PERFORMANCE CHARACTERISATION OF IP NETWORKS

by

BOGDAN VLADIMIR GHITA

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Computing, Communications, and Electronics
Faculty of Technology

February 2004

Performance characterisation of IP networks

Bogdan Vladimir Ghita

Abstract

The initial rapid expansion of the Internet, in terms of complexity and number of hosts, was followed by an increased interest in its overall parameters and the quality the network offers. This growth has led, in the first instance, to extensive research in the area of network monitoring, in order to better understand the characteristics of the current Internet. In parallel, studies were made in the area of protocol performance modelling, aiming to estimate the performance of various Internet applications.

A key goal of this research project was the analysis of current Internet traffic performance from a dual perspective: monitoring and prediction. In order to achieve this, the study has three main phases. It starts by describing the relationship between data transfer performance and network conditions, a relationship that proves to be critical when studying application performance. The next phase proposes a novel architecture of inferring network conditions and transfer parameters using captured traffic analysis. The final phase describes a novel alternative to current TCP (Transmission Control Protocol) models, which provides the relationship between network, data transfer, and client characteristics on one side, and the resulting TCP performance on the other, while accounting for the features of current Internet transfers.

The proposed inference analysis method for network and transfer parameters uses online non-intrusive monitoring of captured traffic from a single point. This technique overcomes limitations of prior approaches that are typically geared towards intrusive and/or dual-point offline analysis. The method includes several novel aspects, such as TCP timestamp analysis, which allows bottleneck bandwidth inference and more accurate receiver-based parameter measurement, which are not possible using traditional acknowledgment-based inference. The the results of the traffic analysis determine the location of the eventual degradations in network conditions relative to the position of the monitoring point. The proposed monitoring framework infers the performance parameters of network paths conditions transited by the analysed traffic, subject to the position of the monitoring point, and it can be used as a starting point in pro-active network management.

The TCP performance prediction model is based on the observation that current, potentially unknown, TCP implementations, as well as connection characteristics, are too complex for a mathematical model. The model proposed in this thesis uses an artificial intelligence-based analysis method to establish the relationship between the parameters that influence the evolution of the TCP transfers and the resulting performance of those transfers. Based on preliminary tests of classification and function approximation algorithms, a neural network analysis approach was preferred due to its prediction accuracy.

Both the monitoring method and the prediction model are validated using a combination of traffic traces, ranging from synthetic transfers / environments, produced using a network simulator / emulator, to traces produced using a script-based, controlled client and uncontrolled traces, both using real Internet traffic. The validation tests indicate that the proposed approaches provide better accuracy in terms of inferring network conditions and predicting transfer performance in comparison with previous methods. The non-intrusive analysis of the real network traces provides comprehensive information on the current Internet characteristics, indicating low-loss, low-delay, and high-bottleneck bandwidth conditions for the majority of the studied paths.

Overall, this study provides a method for inferring the characteristics of Internet paths based on traffic analysis, an efficient methodology for predicting TCP transfer performance, and a firm basis for future research in the areas of traffic analysis and performance modelling.

Table of contents

Table of Figures

Acknowledgments

I would principally like to acknowledge the contributions of the following people:

- Professor Emmanuel Ifeachor, my director of studies, for his guidance in research and for setting a high standard, both for this project and for my entire research.
- Dr. Steven Furnell, for his technical and professional advice, his personal support throughout the project, and for his patience while reading countless draft versions of the thesis.
- Dr. Benn Lines, for his technical advice and encouragements.
- Dominique LeFoll, from Acterna, for providing me the opportunity of pursuing this research and for his input and guidance during the monitoring part of the research.
- My colleagues from the Network Research Group for their friendship and for their help, whenever requested.
- My family and Oana, my girlfriend, for their support and encouragements, without which this thesis would never have been written.

Glossary

ACK	Acknowledgment
ANN	Artificial Neural Network(s)
AMP	Active Measurement Project
BSD	Berkeley Software Distribution
CBR	Constant Bit Rate
IDA	Intelligent Data Analysis
ITU	International Telecommunication Union
ISP	Internet Service Provider
LBNL	Lawrence Berkeley National Laboratory
MoIP	Multimedia over IP
MSE	Mean Square Error
MSS	Maximum Segment Size
NIMI	National Internet Measurement Infrastructure
NAI	Network Analysis Infrastructure
NAT	Network Address Translation
NLANR	National Laboratory for Applied Network Research
NS	[ISI] Network Simulator
OS	Operating System
PMA	Passive Measurement and Analysis
PSTN	Public switched telephone network
RFC	Request For Comments
RTCP	Realtime Transport Control Protocol
RTO	Retransmission TimeOut
RTP	Realtime Transport Protocol

RTT	Round Trip Time (delay)
RYL	Random Yahoo Link
SLA	Service Level Agreement
SNNS	Stuttgart Neural Network Simulator
TTL	Time To Live
VoIP	Voice over IP

AUTHOR'S DECLARATION

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

This study was financed with funding from Acterna and the University of Plymouth.

Relevant scientific seminars and conferences were regularly attended at which the work was presented. Contacts from Acterna provided technical advice and guidance, particularly in the early stages of the project. Details of the published papers are listed in the Appendices

Signed ....

Date 3/06/05

Chapter 1. Introduction

Since its inception, the Internet has witnessed an exponential increase in the number of hosts it interconnects. A series of network studies has shown that the number of allocated IP addresses increased substantially between 1993 and 1999, growth that slowed down in the last few years [ISC 2003]; while this slowdown may have a variety of causes, such as connectivity via Network Address Translation (NAT) devices, one of the reasons may also be the saturation of Internet provisioning. This is one of the reasons why, during the past few years, the networking area turned from providing connectivity to providing quality, change reflected in application performance. In this context, Internet Service Providers (ISPs) and customers focused their interest towards observing, understanding, and controlling the performance characteristics of their network and their connectivity to their neighbours [NYI 2003] [MCI 2003], [Pipex 2003].

The increased interest to control network and application performance was hindered by the 'best-effort' character of IP (Internet Protocol), which does not provide any quality guarantees. This is why the focus of the research moved towards observing the network paths features and estimating performance.

Current monitoring approaches use active monitoring or passive offline analysis of stored traffic to obtain a precise image of the network status. Although successful in their purpose, such methods present several disadvantages. They require an infrastructure of active probes and capturing devices deployed throughout the studied internet and typical tests involve traffic exchanged between these probes, traffic that adds to the network congestion. As the Internet has became more complex, such infrastructures become less viable due to their scalability issues and the resulting amount of traffic. Also, these infrastructures provide information regarding the aggregation and core segments of the network, where the probes are likely to be located, rather than the quality parameters for the end-networks, which are of interest for the end-user. With regards to offline trace analysis, the result lacks timeliness: the characteristics of the studied internetwork are likely to change between the time when the traces were collected and the present.

One of the aims of the project on which this thesis is based is to propose a novel, online, non-intrusive method to analyse traffic, using a single-point of capture, in order to reduce or eliminate the disadvantages of current network monitoring solutions. The proposed method aims to overcome the above-listed limitations and to allow a new approach for observing network conditions such as loss, delay, and bottleneck bandwidth.

In parallel with studying the properties of the Internet, there is a growing interest in the area of protocol behaviour modelling. TCP is a typical focus for such research, particularly due to its network status awareness and self-adjusting nature. Various studies aimed to characterise mathematically the behaviour of data transfers in order to improve efficiency and to allow performance prediction. The models are all based upon the same theoretical concept: evaluate the throughput by emulating the variations in transmission pace during a TCP connection. This mathematical approach appeared to be accurate when validated, but it was designed for long connections, atypical for current typical Internet traffic, and it requires comprehensive knowledge about the behaviour of the endpoints. Because of this, the method requires prior study for any new TCP implementations and, more important, it cannot provide reliable estimates for senders with unknown behaviour.

This research aims to improve the area of performance modelling by proposing a neural network based model to evaluate TCP performance. The goal is to provide a model that describes accurately, relative to the mathematical models, the relationship between network parameters and the resulting transport performance.

The users' demand for quality is likely to increase in the near future. In such a performance-focused environment, the areas of network monitoring and quality evaluation will occupy a critical role. There is a need to improve existing methods in these areas or to propose novel approaches that offer a better alternative in terms of accuracy. The learning capabilities of the neural network will make it possible for the model to be adapted for future implementations or application requirements.

1.1 Aims and objectives

This thesis advances the understanding of current Internet characteristics and transfer performance. Two main areas of study have been identified: network monitoring and performance prediction.

The first aim of the thesis is to provide a novel approach to the area of traffic monitoring. The proposed method infers the current status of network paths in a non-invasive manner, using online analysis of existing traffic. It will be shown that, although current active measurement architectures are successful in examining and describing network characteristics, the complexity of the Internet is likely to make them costly in terms of network resources.

The second aim of the thesis is to provide a novel robust alternative, using a neural network based approach, to current performance prediction methods. The research aims to observe that current, mathematical models, the only alternative of inferring application performance, although successful in explaining the theoretical behaviour, may perform poorly when dealing with real traffic. The proposed method uses a neural network model to approximate the relationship between network parameters and data transfer performance. The proposed approach will be evaluated in terms of accuracy relative to the existing mathematical models.

To achieve these aims, the research programme had the following specific objectives:

1. To identify the current state of the art in the areas of network conditions analysis (encompassing topics such as monitoring, traffic analysis, or network probing) and TCP performance modelling.
2. To present a novel method of single-point, non-intrusive, online traffic analysis and monitoring that overcomes the disadvantages of existing analysis techniques.
3. To produce and evaluate a prototype for the proposed traffic analysis method, using synthetic and controlled traffic.
4. To produce a snapshot of Internet paths conditions, as seen from the University of Plymouth connectivity perspective by applying the developed traffic analysis method onto sources of uncontrolled traffic.
5. To propose a novel TCP performance model, based on knowledge of prior connections, that overcomes limitations of existing mathematical models and provides better accuracy.
6. To evaluate the accuracy of the proposed model, using a combination of synthetic and uncontrolled traffic.

1.2 Thesis content

The thesis begins with an overview of the current state of the art for the areas of traffic monitoring and performance modelling. This overview, presented in Chapter 2, starts with a brief description of several essential TCP/IP core protocols, which collectively form the focus of the monitored traffic. This is continued with a review of the current efforts in the area of network monitoring, together with the focus of these studies and their purpose, followed by a taxonomy of the techniques described. The third part of the overview discusses the efforts in the area of TCP modelling, starting from early TCP steady state models up to recent models slightly adapted for short-lived connections. The section concludes with an outline of environments that may be used for validating traffic analysis and modelling methods. It is noted that such environments range from traffic with a synthetic element, generated using either a network simulator or a network emulator, through semi-controlled traffic, produced with the aid of a random link generator, up to real traffic, collected from a network backbone

Chapter 3 builds upon the monitoring overview from Chapter 2 in its first section. It provides a summary of characteristics that would contribute to an ideal monitoring method, all aiming to overcome the limitations of existing methods. The chapter then continues by presenting the proposed monitoring method, which uses the issues highlighted in the previous section. The discussion includes the targeted traffic and then follows with the framework and theoretical aspects of the technique. The chapter describes the details of TCP and real-time analysis, the target protocols of the proposed method. The chapter finishes with an overview of the achievements of the proposed method.

The proposed monitoring method is analysed throughout Chapter 4. The structure of the study follows the inferred network parameters. Each section highlights advantages or limitations of the method. The advantages are presented by comparing the method with previous approaches. The limitations consist of particular cases when the inference routines may lead to errors by misinterpreting certain network events.

Chapter 5 presents the validation tests run to evaluate the accuracy of the method described in Chapter 3. The chapter starts by identifying supervised simulated or emulated environments that may be used to validate the accuracy of the method, based on the outline from section 2.6. The validation process uses these environments and takes into consideration the limitations highlighted in Chapter 4.

Chapter 6 reveals the use of the proposed TCP monitoring method to obtain a holistic view of current Internet paths and web page transfer characteristics. The study starts by describing semi-controlled and real traffic sources, together with the scalability issues involved. The remainder of the chapter provides an image of the Internet, as viewed through the analysed traces. Concluding, the discussion highlights the fact that the images provided by the two types of sources are very similar, showing the Internet to be dominated by low-delay low-loss paths, and short-lived TCP connections.

Chapter 7 advances the research to the next level, from studying network monitoring to building the relationship between network conditions and application performance. The chapter proposes a novel TCP performance prediction model, based on Intelligent Data Analysis. After starting with a justification for using IDA, the chapter presents the entire process, starting with data pre-processing, followed by core analysis, and ending with interpretation of the results.

Chapter 8 compares the accuracy of the IDA-based TCP models proposed in Chapter 7 with the results obtained using a traditional mathematical approach. Two separate models are provided, one for connections without losses and one for connections that encountered losses. The validation tests are separately performed on a wide range of traffic conditions, ranging from simulated traffic to real network traffic. The results are encouraging, particularly for connections without losses; the discussion also highlights the simplicity of synthetic traffic and the difficulties encountered when modelling connections with losses.

Chapter 9 concludes the thesis. It highlights the achievements of this research programme in the areas of monitoring and modelling, as well as the limitations encountered throughout the studies performed. Finally, it summarises a number of promising directions for future work.

The thesis includes three appendices to provide further clarification of the issues presented. Appendix A provides details regarding the real-time traffic analysis method; the presentation starts by describing the method itself, it continues with details on its implementation, and it concludes with the validation tests run to benchmark it. Appendix B includes several scripts written during various stages of the research for automating the neural network processing of traces (Appendix B.1), generating synthetic packet traces using a network simulator (Appendix B.2), and retrieving web pages using a command-line web client (Appendix B.3)[1]. Finally, Appendix C includes copies of selected research papers arising from the project.

[1] The implementation of the proposed traffic analysis method was considered too large to be included in the appendices, but exists in electronic form on the attached CD.

Chapter 2. Traffic analysis and performance modelling

2.1 Introduction

The literature survey covers four areas that were involved in this research programme: network protocols, network monitoring, performance modelling, and sources of traffic. In order to understand the domain of traffic analysis, the review will commence with the theory basis for Internet communication – the protocols. For each protocol, the presentation will expand on its functionality and will highlight the characteristics that are likely to influence its behaviour and performance.

The second part of the literature review will focus on the state of the art achieved in studying network traffic. The discussion will start by justifying the increasing need for monitoring, caused by the evolution of Internet provisioning from basic customer connectivity towards offering quality to network applications. The section will then expand on state of the art for network analysis methods from the historical approaches to the present days. These methods will be then grouped in a taxonomy based on three criteria:

- *interaction* between the monitoring tool and the measured environment, i.e. *how* the method obtains its output data
- *timeliness* of the obtained information, i.e. *when* is the method applied onto the studied environment
- *information* resulting from applying the method, i.e. *what* the method provides as a result of the analysis

The third part follows the understanding of traffic by studying the relationship between the achieved performance and the parameters which influence it. After previously presenting the rules of transferring data and describing how current approaches manage to identify the network conditions, the discussion continues with an overview of performance modelling. The section describes the efforts made towards producing a mathematical model to estimate the performance achieved by TCP connections. The presentation will follow the improvements that various studies brought to the initial idea of formalising the TCP transfer based on the changes that loss and delay inflict on the amount of unacknowledged data that the sender may transmit at one time.

The fourth part of the chapter overviews the types of traffic sources that may be used when validating a monitoring method or evaluating the parameters of the Internet. The section identifies several types of sources, ranging from synthetic traffic, produced without releasing any actual packets on a network, through controlled network or endpoint conditions, up to real, uncontrolled traffic, produced by unknown endpoints. As it will be highlighted, each of these sources is used in one of the project stages, either to validate a proposed method, or to infer the current status of the Internet.

The last section of this chapter summarises the gathered information and opens the way towards the proposed methods and models within this research programme.

2.2 The Internet protocols

A series of studies run since 1993 [ISC 2003] has shown that the Internet has become a more and more complex collection of networks through its exponential increase in terms of host numbers. From a human perspective, this success is most likely due to the ease that the Internet has shown in disseminating knowledge, allowing virtually everybody to publicise and access information. This openness was strongly encouraged in the past decade by unifying the access methods through the advent of the World Wide Web.

From a technical perspective, this fast expansion was made possible only due to the robustness and efficiency of the protocols involved in transporting the information. The protocols gathered within the TCP/IP stack, currently running the Internet, were standardised in the early 1980s; their principles, with add-ons to improve efficiency, are still in place, governing present transfers of data. This section discusses only five of them, relevant to the focus of this study. The first discussed is the Internet Protocol (IP) [ARPA 1981a], providing connectivity at the network layer, followed by the main choice for reliable transport protocol, Transmission Control Protocol (TCP) [ARPA 1981b}. The discussion ends with a short overview of HyperText Transfer Protocol (HTTP) [Fielding *et al* 1997], which provides the functionality for WWW.

The following sections assume a basic level of familiarity with the TCP/IP stack and the described protocols. If certain aspects may be insufficiently expanded, the recommended sources for further clarifications are the documents listed as sources in each section.

2.2.1 Internet Protocol

IP, the core of the Internet, provides an addressing infrastructure for the network and basic mechanisms of error detection and fragmentation-and-reassembly [ARPA 1981a]. Although its header includes functionality for preferential service of traffic, due to the complexity of the Internet and the peering character of its carriers (both of them discouraging end-to-end quality of service provisioning), it is only used as a best-effort protocol. From the point of view of this project, IP provided valuable information in two areas. First, it helped defining unique flows, through the combination of IP addresses and ports described in section 3.2. Second, it allows, through its fragmentation functionality, differentiation between packet misordering and packet loss, via an inference mechanism presented in section 3.3.

2.2.2 Transmission Control Protocol

From a technical perspective, TCP is a connection-oriented transport protocol which provides for end-to-end reliable transfer, and includes self-tuning according to the conditions of the network [ARPA 1981b], being the most complex from the protocols described in this section. The positive side of its complexity is its robustness and flexibility; it is based on Jon Postel's philosophy "be conservative in what you do, be liberal in what you accept from others" [ARPA 1981b].

The connection-oriented character of TCP is achieved through a state machine, thoroughly described in the defining Request For Comments 793 [ARPA 1981b]. The state machine includes 11 states: one for inactivity (CLOSED), three for connection initiation (LISTEN, SYN SENT, SYN RCVD), one for data transfer (ESTABLISHED), and six for connection closing (FIN WAIT-1, CLOSING, TIME WAIT, CLOSE WAIT, LAST-ACK). The transitions between states are determined by system calls (CLOSE, passive OPEN), arrival of packets with set control bits (SYN, FIN, ACK, RST), or expiration of timers (Maximum Segment Lifetime). In order to identify the lifespan of a TCP connection based on the packets exchanged, this state machine must be replicated.

The reliable transfer is achieved through data segmenting and sequencing. The transfer of a data object/stream is realised by splitting the information into segments. The sender associates a *sequence number* to each transmitted byte from the data stream; to simplify the mechanism, each transmitted segment carries the sequence number associated with the first data byte it carries. The receiver returns *acknowledgment numbers* to confirm the next data byte expected. This process uses a mechanism known as *delayed acknowledgments* to reduce the number of acknowledgment packets returned to the sender: rather than sending an acknowledgment for each data segment, the receiver produces an acknowledgment for every *n* data segments; in case the sender has less than *n* segments to send, the receiver will send an acknowledgment at the expiration of an associated timer. Current implementations use *n*=2, which is why this policy is also known as *every-other acknowledging*. Loss is identified through duplicate acknowledgments or retransmission timeout (RTO). The first category of losses is detected when the receiver repeatedly acknowledges (through *duplicate acknowledgments*) a data byte that bears a sequence number lower than the last byte sent[1]. The retransmission is triggered when *d=3* duplicate acknowledgments are received (this is to eliminate minor misordering events). The second category of losses is detected when a sent data segment is not acknowledged before a timer expires.

The timer, known as RTO timer, is subject to a relatively coarse resolution (which will be discussed in section 2.5.2) and is calculated using an RTT estimate. This estimate indicates the maximum delay that a data segment and its acknowledgment can encounter as they transit the network path between the two endpoints. The value is computed by the TCP client using the following algorithm, as thoroughly explained in [Stevens 1995]:

- Determine a *smoothed RTT* (SRTT):

  $$SRTT \leftarrow SRTT + g \cdot |RTT - SRTT| \qquad 2.1$$

  where *RTT* is the last measured RTT, and g is a gain factor; set to 1/8

[1] There is the choice, enabled and functional in approximately a quarter of servers – 28% according to [Floyd 2001] or 14% according to [Wendland 2000], of Selective ACKnowledgments (SACK), [Mathis *et al* 1996] where, rather than sending duplicate acknowledgments to indicate the loss of a segment, the receiver sends a SACK packet to indicate the missing data segment.

- Determine the RTT variation (RTT_{var}):

  $$RTT_{var} \leftarrow RTT_{var} + h \cdot |RTT - SRTT| \qquad 2.2$$

  where h is the gain factor for RTT_{var}, set to 0.25

- Compute the *retransmission timeout* (RTO) as:

  $$RTO = SRTT + \beta \cdot RTT_{var} \qquad 2.3$$

  where β is the delay variance factor, specified to be 2.0 in [Jacobson and Karels 1988], then after further research, changed to 4.0 in [Jacobson 1990]

- Apply exponential back-off if the packet is lost more than once.

Even at the sender a loss is inferred rather than announced by the arrival of a packet and section 4.2 discusses some of the difficulties encountered in the inference process based upon observation of the packet arrival sequence.

The success of TCP is due to its network-aware, self-tuning character. The number of data segments a sender may transmit at any time, providing that the sender has a large amount of data to transmit, is limited by an upper-bound, called *congestion window* and measured typically in data segments (some implementations use bytes). This limit is controlled by the acknowledgments received from the corresponding TCP entity, which are regarded as an indication of the current network status, i.e. delay and loss. The congestion window is increased, after a policy dependent on the historical network conditions (described below) every time new acknowledgments are received, and decreased every time a data loss is inferred.

Acknowledgments are generated when new data segments arrive at the receiver and the transport of both the data segments and acknowledgments is subject to the network bandwidth and delay. As a result, the amount of data that a TCP sender injects into the network is determined by the network conditions and limitations in the congestion window imposed by the receiver.

The packet loss is inferred through the mechanism described above, while the network delay results from the fact that an acknowledgment for a data segment will arrive at the sender only after a certain delay, named the Round Trip Time (RTT) delay. The RTT is defined as the time it takes for the data segment to travel from the sender to the receiver plus the time it takes for the acknowledgment to propagate back, from the receiver to the sender. From the TCP perspective, the RTT governs the pacing of packets during the data exchange.

This is why, in spite of the fact that the data transfer is continuous, TCP transfers may be divided into transmission rounds [Padhye *et al* 1998]. A transmission round can be characterised by the data segments belonging to a congestion window and the associated acknowledgments: a sender cannot transmit newer data in a round before receiving (at least some of) the acknowledgments for the data already sent.

The essence of the self-tuning character of TCP lies in its congestion window update algorithms. The congestion window is initialised at the start of the data transfer by the sender with an implementation-dependent value (the initial congestion window) [Floyd 2001]. Another preset variable is the receiver advertised window (*rwnd*), advertised throughout a TCP connection. The initial phase of the update, used from the beginning of the data transfer until a loss occurs, is governed by the *slow start* algorithm. During this phase the sender increments the congestion window for every received acknowledgment. As a result, the congestion window increases exponentially over each round. The second algorithm, *congestion avoidance* allows linear increases (one data segment per round) and is triggered by packet loss. The two algorithms, based on the way that loss is detected, alter the value of the congestion window and its update policy. When a loss occurs, the congestion window is reset to its initial value; slow start is used until the congestion window reaches half of the value it was before the loss occurred, then congestion avoidance is initiated

The two algorithms described above were later refined by two more in order to account for the specific network conditions [Stevens 1997]. The first refinement, *fast retransmit*, stated that a retransmission should occur if three or more duplicate acknowledgments are received, regardless of the retransmission timers. In addition, the second refinement, *fast recovery*, recommended that the congestion window should be halved rather than reset to the initial value and congestion avoidance should be used after that.

2.2.3 HyperText Transfer Protocol

HTTP is the application protocol used by web browsing applications, based on a request (named *method* in the HTTP specification) – response mechanism [Fielding *et al* 1997]. Typically, the method is a (web) client request for an object from a (web) server; the response is the server returning the object to the client. The initial protocol [Berners-Lee *et al* 1996] had a major limitation: it required opening a new connection for each object retrieved from the server. With the additional 6 packets and 3 RTT required for opening and closing each TCP connection, this behaviour had a negative impact on the download speed for e.g. web pages incorporating multiple images. This problem was adjusted in HTTP v1.1 [Fielding *et al* 1997], which allows TCP connections to transport multiple objects, a feature named *persistent connections*. The new version also does not force downloads to be sequential (i.e. having to wait for an object to download before requesting further ones) by allowing multiple requests to be made – *pipelined* *requests*. The impact that these add-ons have on performance was studied before [Heidemann *et al* 1997], but a part of this study, section 6.5, will look at the overall results obtained in practice from HTTP v.1.1 client-server interaction.

2.3 Traffic and network analysis trends: from Internet experiments to monitoring

2.3.1 The need to monitor network status

The growth of the Internet was not followed by similar advances in understanding and predicting its evolution. Until recently, the Internet was still being seen as a developing service due to its novelty and, because of that, the connection / link / network monitoring implied only availability tests. More recently, the nature of the information travelling on the Internet has changed. Technology evolution, expressed mainly in bandwidth increments, and new emerging applications have moved the content of the data exchanged from mainly-text to multimedia-rich and even real-time. The multimedia and real-time applications, aside from opening new avenues for information publishing or broadcasting on the Internet, also brought in new requirements for the network characteristics. Timing and packet loss, which were less of a problem for browsing small web pages or transiting e-mail messages, became an issue when downloading or streaming large multimedia files.

This move impacted on the perception of the Internet service, with beneficiaries requiring and Internet providers offering boundaries for the parameters of their connection [Woods 2000], [Pappalardo 2002], [UUNET 2003b]. It is interesting to note that the need for quality of service reached even the residential customers. A survey run in 2001 on 14,000 residential Internet users found that 70% of the respondents would switch to a different ISP due to the quality that their current provider offers [NNRI 2003].

In recent years, two main directions emerged in the area of Internet quality provisioning: integrated services (*intserv*) and differentiated services (*diffserv*). Intserv, a concept that is behind the Resource Reservation Setup Protocol (RSVP) [Braden et al 1997], is based on a circuit reservation infrastructure. RSVP requires an internet with intelligent nodes, capable of signalling between them and reserve resources based on requests placed by receivers. The approach is currently considered unscalable to larger networks due to the signalling infrastructure and traffic overheads required for its functionality. At the other extreme, *diffserv* [Blake *et al* 1998] has a more simplistic approach, traffic classification: set for each packet a certain priority, then instruct the transiting nodes to obey these priorities. Diffserv is based on redefining the Type of Service field used in the IP version 4 header [ARPA 1981a] and providing functionality to the Traffic Class field in the IP version 6 header [Deering and Hinden 1998], both designed to differentiate between traffic priorities.

In spite of all these initiatives, the current situation is still a best-effort IP governing the Internet, due to complications that quality provisioning would introduce (one of the typical problems being billing between peering networks). This is why, in order to balance the need for quality with the lack of support, the ISPs have to start paying attention to the quality their networks offer to the transiting traffic. In this context, the first step is to evaluate the network parameters in a manner that should reflect the quality that end-users receive. Such tests currently may include network status information such as latency and packet loss, using network probing to peering networks [NYI 2003] [MCI 2003], [Pipex 2003]. This is far from satisfactory, as it provides information only at the core of the network, without expanding on the access segment parameters and, of similar importance, to the parameters for the rest of the path. Starting from these commercial indicators of interest, this section will expand on the research directions proposed in the area of network monitoring and, at the end, will lead to a proposed traffic monitoring approach that aims to satisfy the requirements of current and future network monitoring needs.

2.3.2 The evolution of network performance analysis

The network and traffic analysis started in the late 60s – early 70s with Internet experiments that aimed to provide information about the typical behaviour of the Internet [Kleinrock 1976]. Until mid 90s, the studies aimed to describe the entire Internet; due to its size at the time. Such an aim could be considered satisfied if the study analysed the traffic over a set of international link characteristics, such as [Wakeman *e. al* 1992], [Asaba *et al* 1992], or the behaviour of Internet paths for generated traffic, as in [Bolot 1993]. The international links traffic analysis was limited in the sense that the studies analysed only aggregate information of the traffic, such as link utilisation, data volume, connections rate, percentage of traffic per protocols, etc. This provided valuable information with regards to the amount and nature of the traffic travelling over the Internet, but did little in the area of studying the properties of the paths involved. On the other hand, the method used in [Bolot 1993] aimed precisely to determine the path characteristics, loss and delay, but used generated traffic to achieve this; further, the traffic was constituted from *ping* ICMP requests, which were emulating a Constant Bit Rate (CBR) source rather than TCP traffic (due, for example, to the generic bursty behaviour of TCP traffic). An important study was the one made by Mogul in [1992], where he laid the foundation for offline analysis of TCP traces. The study indicated the issues that may be encountered (most of them relating to the computational limitations at the time) as well as the analysis procedure and provided basic information about the characteristics of the studied paths.

As the Internet evolved, with the web boom starting in the mid-90s, holistic studies of the Internet network properties became more expensive to perform in terms of setup and complexity. Recently, the most well-known individual research attempt was made by Paxson between 1995 and 1997, who probed a mesh of Internet paths. The results were extensively described in [Paxson 1997a] and various aspects were summarised or expanded in [Paxson 1997b-c]. After this large scale experiment, Paxson observed [1997d] the complexity of the Internet then proposed and developed [Paxson *et al* 1998] a monitoring infrastructure (NIMI – National Internet Measurement Infrastructure), as part of The National Laboratory for Applied Network Research (NLANR), to continue the study [Paxson *et al* 2000] of the Internet in a holistic way. All his experiments, including NIMI, were based on an intrusive method, which consisted of multiple file transfers followed by offline trace analysis.

Most of the published studies from the recent years based their results on end network traffic analysis. Starting with 93-94, they looked at end networks, either on a theoretical basis, as in [Leland *et al* 1994], or, most common, trace based, as in [Willinger *et al* 1995] and [Crovella and Bestavros 1996], analysing self similarity in particular. They were continued by trace-based studies such as the ones based on major web servers traffic, either a busy one, such as the Atlanta Olympics server, studied in [Balakrishan *et al* 1997, 1998] or an undisclosed one, monitored for a long period of time (one and a half years) in [Allman 2000]. These studies indicate a definite trend in Internet traffic analysis: while 10-15 years ago a brief experiment was sufficient to define *typical behaviour on the Internet*, the current complexity and dynamics of the Internet [Paxson 1997d] no longer allow for such generic conclusions to be drawn.

Special attention has to be paid to papers analysing the variation of Internet paths properties in time. Studies from this category start with the previously mentioned analysis made by Bolot in [1993], where he looked at the short-term correlation between packet losses, followed by Mukherjee's study [Mukherjee 1994], where he modelled the delay with a Gamma distribution whose parameters depended on the path analysed and time, followed by Yajnik [1999], who applied Markov Chain models to packet losses and found correlations of up to 1s for the loss on the analysed paths. More recent are the studies have been made by Zhang, [2000, 2001], that looked at the stationarity of network properties for a set of Internet paths; the study also included a comparison with earlier measurements made by Paxson [1997c].

Aside from the end network studies, there are currently several initiatives of Internet measurement for research purposes. One of them is NIMI [NIMI 2003], already mentioned above, which includes a mesh of probes that perform intrusive measurements between them to establish the characteristics of the network. NIMI uses Paxson's Network Probe Daemon [Paxson *et al* 1998], running at each of the participating endpoints, to perform the connections, analyse the data, and dispatch the results to the central database. Another two projects, running also within NLANR are AMP (Active Measurement Project) [AMP 2003] and PMA (Passive Measurement and Analysis) [PMA 2003]. The two projects, both running on a mesh infrastructure called NAI (Network Analysis Infrastructure), fulfil complementary purposes. AMP uses the traditional *ping* between probes at end networks and performs intrusive analysis of the paths performance, while PMA has a set of trace agents placed at the core of the network that collect the traffic transiting through those points. These three measurement infrastructures are not an exhaustive list, but only exponents of the current interest for Internet monitoring. Even when using these infrastructures, the results are limited to the size and complexity of the mesh: NIMI has probes placed in 6 countries (with the majority of the probes located in the US), while NAI spreads over the US and has only two probes outside it, one in Korea and one in New Zealand. From this perspective, a study / measurement infrastructure will describe only a punctual behaviour in terms of network topology, depending on the complexity of the mesh, and time, subject to the length of the experiments. The conclusions drawn may be totally erroneous for a different end network or may be outdated at any time, even for the same network[1].

[1] A good example of such dynamics is presented within this thesis in section 6.5. It was observed that the parameters of an end network changed dramatically in an interval of only a few months, all most likely due to a network upgrade.

It may be concluded that the ever-changing Internet requires continuous examination in order to develop its QoS-awareness, promoted in the recent years. This, in terms of traffic analysis, points to a transition from offline trace analysis and / or intrusive methods to passive traffic monitoring. The remainder of this chapter offers an overview of the available tools for network analysis and then closes with a set of requirements for an ideal traffic analysis method.

2.4 Taxonomy of traffic analysis techniques

The evolution of the network analysis was accompanied, as expected, by a wide range of analysis methods and accompanying tools to support the studies. Due to the variety of purposes that led to the development of these methods, a linear classification is impossible to make. This is why this section, aiming to offer an overview of the network analysis proposed methods, is split into three taxonomies (interaction, temporal, and information), with examples that highlight the advantages and disadvantages for each category. The classifications will focus mainly on the methods and tools that relate to performance measurement, in order to follow the focus of this project. A generic overview of existing networking tools may also be found on the Cooperative Association for Internet Data Analysis (CAIDA) website [CAIDA 2003].

2.4.1 Interaction taxonomy

The first characteristic to discuss when considering an analysis method is whether it generates traffic in order to determine the network conditions. From this point of view, there are three main categories of methods and / or tools: intrusive, pseudo-non-intrusive, and non-intrusive.

Intrusive methods cover the majority of the network analysis spectrum. They all use the same principle: send a probe packet / initiate probe traffic to a remote host, receive the response to the probe packet / traffic and analyse. The main purpose of these tools is to determine whether the remote host is reachable and to measure the latency and availability of that host. In the most common case, the probe is an ICMP *ECHO_REQUEST* packet to which the remote end replies with an ICMP *ECHO_REPLY*. The best example of a tool that uses this mechanism is the ***ping*** [Muss 2003] utility and variants, such as ***fping*** [Schemers 2003], ***echoping*** [Bortzmeyer 2003], ***Nikhef ping*** [Wassenaar 2003], etc. A more powerful utility, based on the same principle but using UDP packets with a controlled Time To Live (TTL) value instead of ICMP, is Jacobson's ***traceroute*** [Jacobson 2003], which measures the latency and availability of all the hops between the source and the remote host. Finally, on the highest level of complexity for this class, are the TCP-based measurement tools, which generate TCP traffic and measure the characteristics of the network based on that traffic, such as ***sting*** [Savage 1999]. In parallel with this evolution of tools, from ping to sting, a new class of active measurement tools emerged: bandwidth estimators. These tools, such as ***pathchar*** [Jacobson 2003], ***pathload*** [Jain and Dovrolis 2003], and ***pchar*** [Bruce 2003], worked on the same principle: the packet-pair probing scheme, described by Keshav in [1991]. The scheme uses the assumption that two packets sent back-to-back by the source have a high probability to be queued in sequential positions along their route to the destination. As a result, the time spacing between their arrivals at the destination will indicate the bottleneck bandwidth of the path transited by the flow.

The intrusive measurement methods have two important advantages: accuracy and ease of use. The accuracy comes from the complete control: the tool used controls one of the endpoints, therefore the packets arriving, with the correspondent delays, are guaranteed to inform about the end-to-end, round-trip conditions of the transited network path.

The main disadvantage is the fact that they must generate additional traffic in order to evaluate the network properties. The amount of traffic produced depends on the method used, from low (in the case of *ping* and *traceroute*) through medium (in case of *sting*) and up to very high (in the case of the bandwidth estimators). Aside from the less-obvious Heisenberg uncertainty that some of the methods generating high-traffic may introduce[1], the injected traffic requires bandwidth to evaluate the network. In the case of a low-bandwidth end network, with Internet connectivity in the dial-up to ISDN range, this may end up using a significant amount of the bandwidth, which could degrade the service itself.

The second issue is the actual target (the remote endpoint) of the test. In a generic case, the clients in a network will connect to a variety of servers (web servers in the case of HTTP traffic); an exclusively active method cannot follow all these connections and will attempt to evaluate only the parameters between that endpoint host / network and a single / group of remote hosts. It is appropriate, from this point of view, to monitor the activity on a single specific path, but this will provide no information about the end-to-end quality that users get from the paths they use to connect to the remote endpoints. Finally, the third disadvantage is the type of traffic used for the tests: the majority of methods use ICMP or UDP traffic, which are not TCP shaped, i.e. controlled by a congestion window. Even if the traffic is shaped in a TCP-like form, a QoS-aware environment may treat it differently from the TCP traffic. Also, recent recommendations to configure firewalls, such as [CERT 1999], suggest that network administrators should reject UDP / ICMP packets, in order to avoid possible attacks of the network they manage.

The second category of methods, based on their interaction, is the pseudo-non-intrusive ones. Such methods, relating more to management issues then monitoring itself, require a strong cooperation from the other devices along the way, e.g. using SNMP messages to interrogate MIB databases on routers. Most of the tools available from this category are commercial products,

[1] Once additional traffic is introduced in the network, the actual network characteristics change.

such as Provisio [Quallaby 2003], InterMapper [InterMapper 2003], or TDSLink [TDSLink 2003]. The main advantage of these methods is their accuracy: the results are correct both in terms of figures and in localisation capabilities. Their disadvantage is their dependency[1] on the information they receive from the network devices. This raises no problems if the segments of interest are managed by the same organisation that does the monitoring, but introduces insurmountable problems if some of the interrogated devices are placed outside the managed zone, as it is highly unlikely that a network device will reveal such information to an external party. In addition, there is the inconvenience of additional traffic due to messages exchanged between SNMP entities, minor due to the amount of traffic involved.

The third category of methods is the non-intrusive ones. They do not inject any traffic into the network, but capture and / or analyse packets to infer or measure characteristics of the traffic itself (such as traffic volume) or of the network path transited (such as delay and packet loss). This category covers a wide range of subsets, starting from packet parsers / analysers, followed by traffic capacity analysers, then, at the highest level, performance inference methods. The packet parsers / analysers do not produce any analysis outputs, but only study the content of the packets and output the header fields to the user; most packet capturing programs (e.g. ***tcpdump*** [Jacobson 2003b], ***ipgrab*** [Borella 2003]) may perform such analysis. At the next level, there are commercial packages, such as the ***DataAnalyser*** [Acterna 2003] from Acterna, or the ***Agilent Advisor*** suite [Agilent 2003] from HP/Agilent, that include, aside from packet decoding, complex functionality for monitoring of the overall workload or determining traffic figures for each protocol. Neither of these two subsets includes the functionality provided by performance inference tools. The methods used in this category aim to evaluate the network parameters by inferring the events that led to a certain traffic sequence. For example, a packet loss is inferred to

[1] This highlights the difference between light cooperation, as required by intrusive tools (ranging from an ICMP ECHO_RESPONSE, ICMP_TIMXCEED, or PORT_UNREACHABLE to a web server response), and strong cooperation, which requires the device to reveal internally stored information.

have happened in a TCP connection if a data segment is captured more than once by the capturing / analysing device. There are only a few tools developed in last subset, with ***tcpanaly*** [Paxson 1997b] and ***tcptrace*** [Osternam 2003] being the best known ones, both suited for offline analysis, with ***tcpanaly*** requiring, in addition, a capturing device placed near to / at the endpoints.

The advantages of the tools listed in this category are clear: they do not interfere with the network traffic, do not need any cooperation from the endpoints that produce the analysed traffic, and do not need network resources to draw any conclusions. From the performance analysis point of view, the last category mentioned, the performance inference tools, is the most interesting, as it aims to provide information about the actual parameters of the network from observations of real traffic. On the other hand, the non-intrusive tools, particularly the inference-based ones, have an inherent disadvantage: the inference methods have as input only the captured traffic and the assumptions made may lead to erroneous conclusions about the network events which, in turn, will lead to a lower accuracy. In many cases, as it will be revealed later in Chapter 4, the resulting parameters are the result of a balance between level of information obtained and the assumptions made.

2.4.2 Temporal taxonomy

An important characteristic of a measurement system / infrastructure is its timeliness, i.e. how recent are the results produced by the method. From this criterion, the output of the results can be made either near instantaneously (online) or at a later time (offline). The first category of tools may be used for network / traffic monitoring, while the second is more appropriate for Internet studies based on trace analysis. If considering the products / programs mentioned in the previous section, it can be noticed that the focus was split into two main classes of methods: intrusive online tools, ranging from ping, simplistic, and up to sting, which involves complex analysis, and non-intrusive online / offline tools, starting with the commercial real-time traffic analysers and ending with the *tcptrace-tcpanaly* pair, appropriate for offline study of traffic. From an implementation perspective, the offline analysis, particularly the TCP analysis, is more convenient for complexity reasons, as the fixed amount of data allows multiple parsing of each connection, with fewer concerns regarding computational resources; online analysis does not allow retrospective analysis or any form of accumulation in time (unless the output is logged).

The difference between the two categories comes back to the applicability of network measurement: while in the past performance analysis was geared towards understanding the Internet, future network managers will require pro-active schemes, that will act in a timely manner when the network status changes. It was shown, at the beginning of this range of taxonomies, that the characteristics of an end-to-end Internet path do change in time and that the variations of these characteristics may range from stability [Zhang *et al* 2000] and long-term oscillations [Mukherjee 1994], to short-term [Yajnik *et al* 1999] correlation. This is why timeliness, as previously highlighted, is critical for the validity of a path measurement: a pro-active QoS management infrastructure, relying on the data provided by a measurement scheme, would require real-time results in order to make useful changes in the network configuration.

2.4.3 Information taxonomy

This last taxonomy looks at the type of analysis used by a measurement method, focused on the TCP traffic characteristics. The two categories of tools, based on this criterion, are aggregate and per-flow analysis. The aggregate analysis category encompasses network monitors that provide cumulative information such as statistics on traffic or number of concurrent flows per port numbers. The analysis methods used from this category make no distinction between the characteristics of different flows and have no per-flow information available. The per-flow analysis is based on splitting the captured traffic on flows[1] and analysing each flow individually in order to extract path performance information. The outputs of the two categories are very different: per-flow analysis, based on to the complex analysis performed, provides information such as loss and delay for the transited path, while aggregate analysis offers only throughout information for the captured traffic. The taxonomy separates research tools and current commercial products. Research-lead tools, particularly TCP-based ones such as ***sting***, ***tcptrace***, and ***tcpanaly***, were specifically designed to run per-flow analysis of the traffic captured; on the other hand, current commercial products, aiming to provide a generic real-time picture of the traffic transiting the network, look only at the overall proprieties of the traffic, being, in fact, more appropriate for workload rather than performance analysis.

Overall analysis should not be ruled out as being primitive or inaccurate. It is true that in certain cases, such as real-time UDP flows, the traffic levels or throughput are irrelevant when the flow characteristics are unknown (e.g. a high-bandwidth video streamed over a path with a high packet loss will still have a much higher throughput than an audio stream running on the same path), but for TCP the main quality characteristic remains throughput. In this context, an overall low throughput per stream for an end-network would indicate that the traffic encounters a problem. Unfortunately, with overall analysis, this is the maximum level of information that can be obtained, with no further details of what makes the throughput low.

2.5 TCP modelling – current state of the art

2.5.1 Introduction

The previous sections from this chapter introduced the first step into the process of gaining full

[1] A TCP flow may be identified by the (source IP address, source port, destination IP address, destination port)

knowledge of the network conditions and the performance of the endpoints. The presentation described the current problem facing current as well as future networks: a network administrator has to know what quality the users are getting from the network. This new approach, due to the evolution of the Internet, as well as to the evolution of the services that are being run on top of it, is leading towards qualitative monitoring, as opposed to connectivity ('is the network connected?'), availability ('is the remote host available?'), or quantitative ('is it too much traffic running through the network?') monitoring.

The quality of a TCP transfer is represented by the resulting throughput: how fast the file / data object is transferred between two endpoints. The calculation of the value itself is straightforward (with a few notable exceptions, e.g. idle times, described in section 4.6): divide the number of transmitted bytes by the time elapsed. The challenge is to model the performance of these transfers – to determine the relationship between a throughput and a set of parameters. This section examines which parameters influence the performance of a TCP transfer, along with the currently available models and how well these models can be mapped onto real situations.

2.5.2 The influencing parameters for TCP throughput

The evolution of a TCP connection can be seen as a function of three main categories: the network conditions, which can dramatically affect the TCP transfer through loss, delay, and bandwidth, the behaviour of the TCP sender, which decides the pace at which data segments should be sent, and the behaviour of the TCP receiver, which, through its acknowledgments, returns an indication about the quality of the connection, and forces the sender to adjust. In addition to these three categories, this section will also discuss the impact that file size has on TCP performance.

quadruple, which uniquely specifies the exchange of data at a moment in time.

One of the main attributes of TCP is the fact that it is self-adjusting (as presented in section 2.2.2): the TCP sender endpoints infer the conditions of the network through the acknowledgments they receive from the pairing endpoints and adjust the transmission pace accordingly. This mechanism makes the network conditions to be the main influence factor in the TCP performance. The sum of these network conditions is reflected in the variable that controls the pace of a TCP transfer: the congestion window. This behaviour, while providing awareness of the network status, generates the above-mentioned strong relationship between the network conditions and the resulting throughput.

The network conditions, and the way they are perceived by the endpoints, relate to the TCP sender characteristics and lead to the second category of parameters. It is worth noting that the network conditions *inferred* by a TCP sender do not always coincide with the *real* network conditions, a difference that will be highlighted in section 3.2. Further, *from the TCP client point of view*, it is the *inferred* parameters that really count towards a certain throughput, and not the *real* ones[1]. This is not a big issue for two TCP clients that have the same rules / implementation code: there is, indeed, a difference between the real loss and the inferred one, but the two TCP clients will react the same to such an event, leading to the same throughput. The real issue is that, as noted in several studies such as [Dawson *et al* 1997], [Floyd and Padhye 2001], the TCP protocol was implemented differently within the various TCP/IP stacks. Some of these differences went against the defining document [ARPA 1981b], and refinements [Braden 1989], such as generic 'broken retransmission behaviour' of TCP clients. However, some of them did not violate the specification, but only varied within the proposed limits, such as the retransmission timeout (RTO) clock resolution, discussed next.

A good debate on the accuracy of network conditions inference, focused on the retransmission timeout and available bandwidth parameters, is offered in [Allman and Paxson 1999]. Aside of the conclusions, the authors give an example of the impact that behaviour of the TCP client has on the dynamics of the TCP transfers. They consider two extreme cases of a setting the RTO and explain the consequences. On one side, a client may set the RTO to a very high value, e.g. 1 minute, and never mistakenly retransmit a single packet, but reduce drastically the overall efficiency of the connection. At the other extreme, a very aggressive client can set the RTO to 1 ms and, while reducing to minimum the amount of time spent while expecting for possible timeouts, it would congest the network due to unnecessary retransmissions, locking it completely. It is true that such cases do not exist in real-life implementation, but differences do exist between implementations; for example, the RTO clock resolution of a Linux sender is 100 ms, while a Solaris endpoint has this value set at 500 ms. As a result, an acknowledgment received with a delay in the (RTO + 100 ms, RTO + 500 ms) interval will trigger an unnecessary retransmission in a Linux client, impacting on the congestion window as well, while a timeout-inferred loss would delay the Solaris client an additional 400 ms before taking any action. On the other hand, the Solaris RTO is initialised to a very low value of 300 ms, [Dawson *et al* 1997], which makes it very ineffective for connections with high RTT and produces a large number of *RTOError* packets at the beginning of a connection [Paxson 1997a][2]. The situation perpetuates due to the Karn's algorithm [Karn and Partridge 1991] that forbids RTT estimates for retransmitted packets, so the client might never infer the real RTT within the connection, and erroneously retransmit every single data segment several times. Although it is beyond the scope of the project to determine "what is better" or "what is normal" in terms of implementation efficiency, it must be mentioned that, when it comes to variations from the suggested behaviour

[1] This is very convenient from the perspective of this project, as the measurement method proposed in Chapter 3, due to its characteristics, outputs exactly these inferred parameters.

[2] The two timers are positioned at opposite extremes: while, at the beginning, the sender will throttle the network with unnecessary retransmissions due to inferred losses, overloading the network, at steady state it would react very slowly to the real losses, decreasing the performance of the transfer

and values in [Braden 1989], it is not a question of "ideal implementation". For example, in the above-blamed Solaris, all the studies that incriminated it admitted that, while less efficient for 'normal' delay and loss values the Internet, the implementation was well-suited for LANs, or, in general, for networks with short delays and large bandwidths.

Ideally, this sender modelling requires defining a set of parameters that fully describe the behaviour of each implementation. Previous work, [Paxson 1999], [Floyd and Padhye 2001], proposed solutions to identify the behaviour of a TCP sender, but none of them is suited for a non-intrusive single-point analysis method. The main problem is the amount of inference required while observing the TCP transfer. While this obstacle was avoided, with some notable exceptions for the network parameters[1], the TCP sender characteristics are more intimate and, therefore, their impact on the transfer performance is rather transparent to the monitor, e.g. the delayed acknowledgment policy. Further, it is difficult to separate the actual behaviour of the TCP client from the sender activity: e.g. a data segment produced after a long delay might be due to either to a TCP timeout or due to the TCP server being busy with several other connections (this difference is spotted by network sniffer detectors, such as *antisniff* [L0pht 2001]). Different implementations would react differently to a certain event, with similar short-term impact, but leading to different long-term consequences. For example, in the RTO example above, the Linux client will infer and react faster than the Solaris one for a lost packet when reaching steady state. As a result, both of them will retransmit the packet, but the Linux client will start with a congestion window of one segment and slow start, while the Solaris client will start with a halved congestion window and will switch to congestion avoidance.

Two possible directions can be pursued to achieve the desired level of information about a TCP sender: inferring the TCP implementation from its behaviour or retrieve information that categorises TCP in a certain class

The first option is to define a comprehensive set of parameters that fully describe TCP behaviour. While being the ideal, this aim is virtually impossible to achieve for uncertainty reasons. The nearest proposed method, implemented in ***tcpanaly*** by Paxson [1997b], classifies a TCP sender as belonging to one of a predefined set of known classes or defines a new class if it does not match any known one.

The second option is very similar: define a minimal set of parameters that define a specific TCP implementation. This is in fact the approach used by current sender identification tools, starting with efforts from hacker groups, [Fyodor 1998], up to the research within the ***tbit*** project [Floyd and Padhye 2001]. These methods generate a specific succession of packets, and analyse the response coming from the corresponding TCP endpoint to determine its implementation. Another study worth mentioning here, using a non-intrusive approach is from [Popescu and Shankar 1999], which profiled the throughput versus the TCP implementation of the sender; in this case, the authors modelled the dependency using an empirical formula and validated it only in a synthetic environment.

An alternative derived from the second option and proposed by this research is to obtain information that classifies a server application running at the sender into a particular class. This method is based on the assumption that a certain version of an application runs on a certain platform. An example would be a Microsoft IIS web server, which can run only on a Microsoft Windows operation system. The problem in this case resides at a different level: the analysis becomes application-dependent, as it will require information provided by the application layer.

[1] These exceptions will be described in section 3.4.3

From the above options, only the latter one is appropriate for the proposed method of analysis. The first solution succeeded with privileged positioning of two monitoring devices, while the second one required intrusive analysis. Because of this, the usage of such sender information would restrict the method to analysing a specific type of application; due to its popularity and usage levels, the chosen application protocol to exemplify the technique was HTTP. The client information is laid relatively conveniently within the data segment: the data segment carrying the HTTP response, "OK 200", carries in one of the tag fields the version of the web server that holds the requested web page, as shown in Figure 2.1

```
HTTP/1.1 200 OK
Date: Fri, 05 Oct 2001 18:52:19 GMT
Server: Apache/1.3.12 (Unix)  (Red Hat/Linux)
Last-Modified: Thu, 11 Nov 1999 10:15:07 GMT
ETag: "8139b-bde-382a972b"
Accept-Ranges: bytes
Content-Length: 3038
```

Figure 2.1 – The header of an HTTP response

As can be seen in this case, the `Server` tag indicates the HTTP server type (Apache) and version (1.3.12), and even the operating system (Red Hat/Linux). This tag may be parsed if the captured packet has a *snaplen* large enough to hold the entire HTTP header and may be associated with the corresponding TCP connection for further processing.

This last method, although flexible, has several disadvantages that are worth detailing:

- It is assumed that the owners of the machine did not alter its default configuration. The parameters of TCP clients within current operating systems can be modified by the user (e.g. altering the RTO values or acknowledgment policy), leading to a different behaviour, but this should not happen unless the default configuration generates major performance problems.

- All OSs suffered changes in their evolution. Each flaw noticed in Solaris implementations was corrected in the following versions, the TCP clients evolved within each Linux kemel, and Microsoft implementers improved the core of their TCP client with every version of Windows. All these changes produce a strong variation between different versions of the same operating system, leading to erroneous profile predictions. In this case the simplification is generated by the progress itself: each new version of OS is (or at least claims to be) better than the previous one, forcing users to upgrade; each new version of a web server is likely to be incompatible with, or at least not optimised for, older versions of the operating system, again forcing to upgrade. As a result, a certain combination of web server – operating system should be fairly focused towards the latest implementation at one moment in time

In a similar manner, the behaviour of the receiver has also a strong impact upon the resulting connection throughput. It is reasonable to consider that current TCP implementations use the delayed acknowledgment policy: an acknowledgment should be transmitted for every other data segment. Nevertheless, in the cases when only one data segment is received at a moment in time, the receiver uses a timer expiration to produce an acknowledgment to that segment. Again, this timer depends on the implementation. For example, although both BSD-derived [Stevens 1995] and Microsoft Windows [Microsoft 2000] TCP clients use a 200ms timer, the implementations differ in the way they use the timer. The timer from BSD-derived implementations resets itself every 200 ms, but is not influenced by the actual retransmissions. This leads to delayed acknowledgments being sent with a delay that has a uniform distribution between [0.0ms;200.0ms]. On the other hand, the Microsoft Windows timer for delayed acknowledgments is set by data segments arrival. This leads to delayed acknowledgments being delayed for exactly 200ms. With compulsory usage of this timer at least once per connection for senders with initial sender congestion window of 1 Maximum Segment Size (MSS), the policy will impact on the resulting throughput figure as well.

The last parameter that influences the transfer is the size of the data object (file) that is being transferred. As previously detailed in section 2.2.2, the TCP client requires a certain amount of data segments to increase its congestion window depending on its bandwidth x delay product, until reaching steady state. For a considerable proportion of traffic, as will be revealed in Chapter 6, this does not happen because the objects transferred within connections are too small to allow the TCP client to reach steady state. However, because initial and transient behaviour is more difficult to model, current TCP models assume that the TCP connection reaches steady state and focus mostly on the network-related parameters.

To summarise, there are three categories of factors that affect the performance of TCP transfers: network conditions, endpoints (sender and receiver) behaviour, and file size. The parameters listed within these categories are used by the current TCP models to determine what influences TCP throughput, as will be presented in the following sections.

2.5.3 Current TCP mathematical models

TCP modelling efforts aim to determine the relationship between throughput and its influencing factors by describing the behaviour of TCP endpoints, as affected by them. This section gives an overview of 3 proposed TCP models, each of them with its limitations and assumptions required. These models assume certain network characteristics, as well as client behaviour, and then build up a model of TCP data transfer. Their result is a description of throughput as a function of the network conditions and client characteristics.

The first model that formalised the TCP behaviour appeared in a note within Bellcore Labs in

1996 [Ott *et al* 1996]. It describes the evolution of an idealised congestion window in time during the congestion avoidance phase in order to estimate the value of the resulting data throughput. The authors made a series of assumptions which allowed them to interpret the congestion window evolution as a stochastic process, and to determine its evolution by studying its stationary distribution. The resulting formula, **2.4**, determines the resulting throughput (named *bandwidth*, *BW* in the paper) as a function of Maximum Segment Size (MSS), Round Trip Time (RTT), packet loss (p) and a constant C, dependent on the acknowledging policy (every / every-other packet) and on the loss process (periodic / random):

$$BW = \frac{MSS}{RTT} \cdot \frac{C}{\sqrt{p}} \qquad \textbf{2.4}$$

The model is based on an ideal TCP connection, as stated in the title of the article. In order to model the congestion avoidance exclusively, the model eliminates any other influence in the data transfer by using several simplifications that do not always hold in a real-life situation:

- Low rate loss and independent losses – the first assumption can be considered true in today's traffic, as it was proven by various Internet experiments, but not in relation to the second assumption. A typical drop-tail router drops all the packets which it cannot hold in its queue(s); as a result, there is a high probability for successive losses to be related and, further, to occur within the same transmission round.
- Infinite sender and stationary state – HTTP traffic is not likely to obey such assumptions, as it typically encompasses short data transfers, therefore most of the connections will never reach stationarity.

It is in fact admitted, within the paper, that the proposed model fails in several situations, none of them uncommon in real-life situations:

- The receiver advertises a small window (RWND) – if the connection path has a high bandwidth x delay product, the advertised window is an important limitation that the following models accounted for.

- The sender always has available data to send – it is very common during HTTP 1.1 connections for the sender to ‘pause’ during the connection, either waiting for a request from the receiver or expecting data from the application layer, as will be shown in section 4.6
- The sender never times out (all the losses are signalled by double acknowledgments and the recovering procedure is fast retransmission) – the loss of a data segment may trigger a timeout. The probability of a loss being detected through a timeout depends on the window size, as will be observed and modelled by the next studies.

It is apparent that the model is fairly limited in respect to real traffic. The study included no real traffic validation and the authors limited the tests to simulation traces analysis. However, it represents the basis for the following studies in the area, as each of them came with improvements but they all used equation 2.4 as a starting point.

The second model, described in [Padhye *et al* 1998], extends the analysis of the congestion window evolution in order to account for losses produced by timeouts as well, aside from the ones inferred by double acknowledging. It also accounts for limitations due to the size of the window advertised by the receiver. The resulting formula for the resulting throughput *B(p)*, 2.5, includes, aside from the loss and delay parameters of the previous model, the acknowledgment *rate* (resulting from the receiver policy), *b*, the timeout value, T_o, and the advertised window size, W_{max}.

$$B(p) \approx min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0\, min(1{,}3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \right) \qquad 2.5$$

The model is closer to reality compared to the previous one, due to the encompassed timeout loss events and the reduced number of assumptions made (e.g. only losses from independent rounds are considered independent, while losses from the same round are correlated, due to the drop-tail behaviour of the router). The validation part analyses real network traces and shows that the proposed formula is a better estimator of the throughput when compared to equation 2.4. Within this part the limitations of the model can be noticed, which relate mostly to the length of the connection. Because, as the previous one, it focuses on the stationary behaviour of TCP, the model was validated using connections spanning long periods; the number of data segments sent was in the range of tens of thousands up to hundred of thousands per connection. With an MSS of 1500 bytes, these figures translate into files between 15 MB and 150 MB, both very far from 'usual' network transfers.

The third model, presented in [Cardwell *et al* 2000], expands further on the previous ones, by accounting for the establishment and slow-start phases of a TCP connection. The proposed model additions are based on the observation that most of current TCP data transfers are short-lived and carry a small amount of data. There is a high probability for such flows, when they follow a path with low loss rate, to have a zero packet loss and, implicitly, to remain in slow-start for their entire duration. Further, if they carry an amount of data that can be carried in only a few rounds, the actual connection establishment represents a high percentage of the entire connection life. The model aims to determine the duration of the transfer, instead of the obtained throughput. The resulting formula for the estimated time of a connection, E[T], is shown in equation 2.6

$$E[T] = E[T_{ss}] + E[T_{loss}] + E[T_{ca}] + E[T_{delack}] \qquad 2.6$$

where:

E[T_{ss}] – estimated time spent in slow-start

E[T_{loss}] – estimated time spent recovering from loss at the end of slow-start period

E[T_{ca}] – estimated time spent in congestion avoidance

E[T_{delack}] – estimated time for a receiver to sent a delayed acknowledgment

The model used to determine E[T_{ca}] is the one from equation 2.5, and the E[T_{delack}] time is taken 100 ms for BSD-derived TCP endpoints and 150 ms for Windows TCP endpoints. The model for E[T_{ss}] is given in equation 2.7.

$$E[T_{ss}] = \begin{cases} RTT \cdot \left[\log_{\gamma}\left(\frac{W_{\max}}{W_1}\right) + 1 + \frac{1}{W_{\max}}\left(E[d_{ss}] - \frac{\gamma W_{\max} - W_1}{\gamma - 1} \right) \right] & when\ E[W_{ss}] > W_{\max} \\ \\ RTT \cdot \log_{\gamma}\left(\frac{E[d_{ss}](\gamma - 1)}{W_1} + 1 \right) & when\ E[W_{ss}] \leq W_{\max} \end{cases} \qquad 2.7$$

where:

RTT – the round trip time average

γ - the congestion window increase factor, due to the acknowledgment rate of the receiver (e.g. 1.5 for the delayed, every-other acknowledging policy)

W_{max} – the receiver advertised window size (in segments)

W_1 – initial congestion window (in segments)

E[d_{ss}] – the estimated number of segments sent in the slow-start phase

$$E[d_{ss}] = \left(\sum_{k=0}^{d-1} (1-p)^k \cdot p \cdot k \right) + (1-p)^d \cdot d = \frac{\left(1-(1-p)^d\right)(1-p)}{p} + 1 \qquad 2.8$$

E[W$_{ss}$] – the estimated congestion window (in segments) at the end of the slow-start phase; determined using equation 2.9, based on the E[d$_{ss}$] estimator resulting from 2.8

$$E[W_{ss}] = \frac{E[d_{ss}](\gamma - 1)}{\gamma} + \frac{W_1}{\gamma} \qquad 2.9$$

The time to recover form the first loss, *E[T$_{loss}$]* is determined using the theory behind [Padhye *et al* 1998], based on:

- the probability of a loss occurring, *l$_{ss}$(p,d)*, as a function of packet loss rate, *p*, and number of data segments to transmit, *d*, as shown below in equation 2.10

$$l_{ss} = 1 - (1 - p)^d \qquad 2.10$$

- the probability of a sender to detect a loss with a timeout, *Q(p,w)*, as a function of packet loss rate and congestion window size, *w*, equation 2.11

$$Q(p,w) = \min\left(1, \frac{1 + (1-p)^3\left(1 - (1-p)^{w-3}\right)}{\frac{1-(1-p)^w}{1-(1-p)^3}}\right) \qquad 2.11$$

- the expected cost of an RTO, E[Z^{TO}], as function of packet loss rate and average timeout, *T$_0$*

$$E\left[Z^{TO}\right]=\frac{T_0\cdot\sum\limits_{k=0}^{max\_backoff}2^{k-1}\cdot p^k}{1-p} \qquad 2.12$$

With the variables resulting from the above equations, the *E[Tloss]* estimator results as:

$$E[T_{loss}]=l_{ss}\cdot\left(Q(p,E[W_{ss}])\cdot E\left[Z^{TO}\right]+(1-Q(p,E[W_{ss}]))\cdot RTT\right) \qquad 2.13$$

Q(p,E[W_{ss}]) – the probability of a sender to detect a timeout at the estimated end of the slow start phase

Due to its complexity, the model allowed testing in all the possible configurations: simulations, controlled internet measurements, and live HTTP measurements. The results of the validation tests show that the proposed model is much more accurate for connections that suffer no loss and has similar accuracy with [Padhye *et al* 1998] when the flow encounters packet loss. The study used simplifications and hard-coded values for the live HTTP measurements: the loss rate due to timeout was inferred from mis-ordered packets which were delayed for more than 200 ms or in-order packets spaced more than 1 second.

It is apparent that the TCP model proposed by Ott went through radical improvements until the one introduced by Cardwell, in spite of the similarity of the mathematical support used. While the conclusions from the first study considered exclusively the TCP transfer reaching steady-state, the last one focused specifically on short-lived connections and aimed to describe fully their evolution. However, all authors were reluctant to test their theories on real traffic, which highlights the limitations of the models proposed.

2.6 Classification of data sources

Network data collection encompasses a wide range of techniques, from interrogating routers to establish their queue sizes to sampling traffic and packet capturing. From these techniques, due to its main objective, i.e. to build a robust non-intrusive monitoring and prediction mechanism, this project focused on data obtained by capturing packets. This section gives an overview of the data sources available when studying network traffic, first by classifying these sources in terms of their approximation of real traffic. Chapter 5 and Chapter 6 will describe how and when each of these categories was used with different stages in the research.

1. Synthetic data – packet traces generated by network simulators. This is the artificial type of network data. There is only an approximate relation with reality, as all the components of the simulated transfer are modelled, and each of the models is an idealisation of the real cases. In addition, the simulator, no matter how complex, still has a limited size in terms of nodes and in terms of network topology as well. All the other categories are using *captured* network traces instead of *generated* traces, i.e. the trace is obtained by capturing real packets instead of artificially producing them.

2. Testbed data – packet traces generated using a limited, controlled, and isolated network environment (a network testbed). This is different from the previous category in the sense that it uses *emulation* instead of *simulation* of the network environment. The difference between the two, according to Carson [1997] is that simulation reproduces an environment by modelling the behaviour of the entities, including the TCP endpoints, while emulation

uses real traffic, generated by real TCP endpoints, over a simulated network.[1]

The advantage is that the amount of idealised components is reduced – the endpoints are real, the traffic is actually produced, but the network conditions are reproduced using an emulation agent running within a router. Also, because of the fact that all the traffic is in fact generated and the endpoints are real, the topology of the network can be only as large as the available endpoints (and most of the time limited to two endpoints and one or two routers)[1]. The position of the capturing device is somewhere within the testbed.

3. Controlled data – packet traces resulting from transfers between real endpoints, both known and controlled by the person who runs the experiment. This is the step that makes the transition from the laboratory environment to the real world. It is, if put in a crude way, a testbed with the Internet in the middle. The network conditions are no longer simulated / emulated, the data is being passed through the Internet; the entire route of the packets is over a real network; the improvement is radical when compared to the previous two sources.

Using this type of data does have two drawbacks. Firstly, the results cannot be reproduced as the network conditions change continuously. Secondly, such a measurement topology is limited, implicitly the network paths also form a finite mesh; the best example of such a measurement initiative is afore mentioned Active Measurement Project [AMP 2003]. The AMP infrastructure includes 130 endpoints, spread throughout the US. The problem with such configurations is their stability over time. It was shown by Zhang in [2001] that network conditions for certain paths tend to persist (i.e. their characteristics remain constant in time),

[1] To complete the picture, the real network testbed case should be mentioned. Such an environment would encompass only real components: real endpoints, connected via a real network, all under the control of the person that manages the experiment. While the environment may have controllable delay, by altering the speed of the links that form it, the loss would have to be emulated by dropping specific packets in the routers.

which, implicitly, limits the result in a narrow range of values.

4. Semi-controlled data – packet traces using an automatic retrieval tool. This takes the network conditions one step nearer to reality; the receiving endpoint remains constant, which, implicitly, limits the range of resulting endpoints combinations, but, in this case, the network paths are virtually unlimited – all the paths do have a common end, i.e. the receiver, but they can expand anywhere in the other direction. Due to the characteristics of this method, i.e. single point that makes all the requests, the capturing point is right at the retrieval point[2].

5. Uncontrolled data – packet traces produced by real network traffic. There is no more artificial / controlled element involved; all the connections are due to genuine (human) requests from the endpoints. The network paths are limited only by the position of the capturing device, i.e. the requesting endpoints can be, still, grouped within a single, local network (e.g. University of Plymouth network).

The data collected for this research project spanned through all the five categories, except controlled data, although not in the mentioned order. The trace collection process was performed in parallel with implementing the TCP monitoring methods presented in Chapter 3. In the beginning, testbed and semi-controlled traces were used to ensure the correctness of the results by comparing the manual analysis results with the results provided by the implementation. The necessity to collect larger network traces became clear once the implementation was finalised. The study of these traces had a double aim: to observe the characteristics of current Internet paths, and to provide a database of TCP connections in order to build a knowledge-based TCP

[1] Progress in the area has been achieved by network simulators (such as ***Network Simulator***) which encompass a network emulation agent

[2] The other possible alternative for this category is to monitor the traffic at a web / file server. In this case, there would be an additional disadvantage: the resulting behaviour of TCP depends more on the behaviour of the sender than on the behaviour of the receiver, therefore the diversity of scenarios would be further limited.

model. In Chapter 5 and Chapter 6 the trace collection process will be detailed for each category, as well as the rationale behind choosing each of them.

2.7 Summary

The literature survey revealed a wide spectrum of issues, all relating to the area of traffic analysis and performance evaluation. The presentation started by introducing the basics of data transfer of the Internet. The description focused on the protocols which were used within this research project and expanded on the algorithms that govern TCP behaviour.

The second part of the survey presented the state of the art in the area of traffic analysis. The section started by describing the Internet as a performance-dependent environment, due to emerging applications and increased customer demands. In parallel, the section pictured the lack of end-to-end quality support within the current Internet, in spite of existing initiatives. These two contradictory facts led to the necessity of evaluating the network status in order to determine whether the provided quality is satisfactory for potential customers. The section then continued with a list of the methods and tools for traffic analysis. The first part of the description was historical, starting from the early probing programs and ending with recent TCP emulation methods. The second part, presented in a separate section, distinguished between the presented tools based on functionality criteria. The proposed three criteria considered the defining questions for the output of a network analyser: what, how, and when to measure. The resulting taxonomy will be used in the next chapter to identify the missing characteristics from current tools and build upon them to depict the features of a virtually ideal network monitoring method. In turn, these features will be used as a basis to define the objectives of the monitoring method proposed in this following chapter.

The third part of the survey looked at the next level of the network knowledge domain. It described the existing series of mathematical approaches that aim to model the relationship between the TCP throughput and the influencing network and endpoint characteristics. The presented series ended with the model considered to provide the best accuracy when evaluating the TCP performance.

Finally, the fourth part of the survey overviewed the data sources which may be used in the validation process for monitoring and modelling methods. The description briefly described the advantages and disadvantages for each of the identified types of environment. Such environments will be further presented in later sections, while validating the proposed methods.

The following chapter proposes a novel approach to network monitoring by online analysis of the traffic. The method attempts to overcome some of the disadvantages of current network analysis tools, observed in section 2.3, such as intrusiveness and complexity of the monitoring architecture, by using a single-point method. The method builds upon previous studies, but also introduces new concepts such as inference of network parameters based on TCP timestamp analysis.

Chapter 3. A novel approach to monitoring

3.1 Introduction

The previous chapter discussed the various historical and current approaches towards network monitoring, highlighting their advantages and limitations. This chapter will start by summarising a virtually ideal solution for network monitoring, based on the conclusions drawn from existing techniques. Using this ideal collection of requirements, the chapter will introduce a novel approach to traffic analysis and monitoring, proposed by this research project. The proposed method builds on the area of per-flow traffic analysis, with improvements in regards to the focused traffic, e.g. real-time traffic, the inferred parameters, e.g. bandwidth, the method, e.g. usage of TCP timestamps, and the robustness of the analysis. The method is considered in terms of its novelty, its main characteristics and functionality, the advantages it brings, and its limitations.

3.1.1 Ideal monitoring method

The previous sections presented the networking world as an evolving domain. Under the pressure of the newer applications, Internet provisioning had to develop from simple availability towards quantity, and recent years and present status took this migration towards quality. Due to these changes, network managers also had to shift their responsibilities from observing network availability to measuring overall bandwidth and throughput. Further, ISPs and customers are bound by a Service Level Agreement (SLA), which, if breached, may lead to legal liability. The set of requirements contained by the SLA for the provided connectivity solution may range from availability conditions to network parameters boundaries. In this context, monitoring also has to transit from measuring availability and quantity towards measuring the quality of the network, as transited by the traffic. In the dynamic market that governs Internet provisioning, a network provider should, aside from optimising the traffic through its own network, check whether the providers he connects to also respect their SLAs; the only solution to satisfy all these requirements is to continuously measure or estimate the quality of the network paths.

The previous section presented a wide range of alternatives with regards to measuring or assessing the parameters of a network path, together with their advantages and disadvantages. From this overview it is apparent that monitoring methods also migrate from evaluating availability and basic path characteristics, as in the case of ***ping*** and ***traceroute***, towards complex examinations of traffic to infer as much as possible about the transited path, with tools such as ***pathchar*** and ***tcpanaly***. For each of the taxonomies suggested, the resulting performance information proved to be a compromise between accuracy and one of the following three factors: network interference (intrusive vs. non-intrusive), cooperation of remote endpoints (pseudo-non-intrusive vs. non-intrusive), traffic aggregation (overall vs. per-flow), or real-time computational load (online vs. offline). It is true that accuracy is one of the defining factors for any measurement / monitoring method, but the proposed solution has to be based on reality. From an end-user or access ISP perspective, the reality is:

- high number of users, all connecting to a wide variety of remote endpoints, virtually impossible to profile;
- backbone ISP / ISPs that the access ISP cannot interact with, as the inter-ISP connectivity is based exclusively on peering agreements, with no control over the neighbouring infrastructures;
- increasing need to evaluate the performance for all users, either per-user or overall, and, if possible, to determine what causes a performance decrease.

Future monitoring tools will have to provide a satisfactory level of accuracy when evaluating the network status and when estimating the level of variation if the network parameters vary. All these requirements will have to be satisfied under the restrictive conditions imposed above: dynamic network conditions, large number of paths to test (although all routes may have a common path segment, through one of the international backbone carriers), no guarantees of cooperation from remote endpoints, and no traffic injected. Comparing these requirements with the presented taxonomies, such a tool will have to be:

(i) Online – to provide real-time feedback information to management entities about the network conditions

(ii) Single point – to avoid elaborate configurations and eliminate traffic between devices

(iii) Non-intrusive – to avoid network overloading

With regards to the targeted traffic, the monitoring should focus on the type of applications that generate most of the traffic today and are likely to produce high levels of traffic in the future networks. It has been shown by previous network studies, such as [Thompson and Miller 1997], [McCreary and Claffy 2000], that web browsing accounts for the majority of the traffic. Because of the simplicity of the HTTP protocols, as described in section 2.2.3, the analysis should focus primarily on TCP and interpret HTTP only to identify correctly the data exchange (e.g. observe persistent and / or pipelined connections). Due to the simplicity of HTTP, the monitor will also offer the possibility to evaluate the network performance for other applications that use TCP. A second direction of traffic to analyse is suggested by the recent trends of real-time applications, generically named MoIP (Multimedia over IP). Such applications [ITU 1999] started to replace traditional communication means such as audio or data-over-audio (i.e. the traditional modems), all part of the PSTN existing infrastructure, with an everything-over-IP environment. In such an environment, all the communication (data, audio, and video) will be done over the existing IP network infrastructure, using either a QoS-aware Internet or a separate Internet, with strict quality requirements. In any case, the traffic will have to be monitored to ensure the quality of the system and to respond promptly to any degradation.

The above guidelines should be taken as strict for a proposed monitoring method. In addition to them, if possible, a monitoring system should be able to predict the performance offered by a network (path) to transiting traffic. There are two areas that an ideal method should cover:

- Performance-wise – predict the *resulting* quality of the applications, based on a set of network *conditions*. This type of prediction is based on performance modelling and would allow a network manager to estimate the results of the alternative changes that can be made in the network configuration. A simple example would be to compare the effect of reducing or increasing the queues size in routers. While the reduction would lead to higher packet loss and reduce the pace of the TCP clients, leading to a fairer environment for short-lived flows, incrementing the queues size would reduce packet loss but increase RTT values, leading to greater congestion windows and better environment for long-lived transfers.
- Time-wise – predict the *future* quality of the network, based on *present* status and past experience. This is, indeed, the final aim of network monitoring: infer future degradation of network conditions based on the current and historical information gathered. Several studies were made with regards to network self-similarity of traffic and models were proposed to predict future network conditions, mostly based on Markov models.

This project aims to produce a single-point, non-intrusive, on-line, accurate, robust, and predictive monitoring method. First, it will propose a technique to analyse from a single point, online, the information provided by the packet flows and infer the performance parameters of the traffic. Secondly, an AI-based model will be proposed which, using knowledge about the network, can model TCP transfers and can estimate the performance that a TCP connection would have over a network path with given parameters.

3.2 Proposed online, single-point, non-intrusive monitoring method

3.2.1 Targeted traffic

The technique proposed by this study focuses on two main types of traffic: TCP connections and real-time-based streams. This subsection details the rationale behind focusing on these two categories of network transfers.

The choice for TCP was straightforward and imposed by the current status of the Internet. It has been shown by previous network studies, such as [Thompson and Miller 1997], [McCreary and Claffy 2000], that most of the current traffic is HTTP/TCP; even more recent studies [Fraleigh *et al* 2003] indicate that, in spite of the increase in peer-to-peer usage, HTTP still accounts for the majority of the traffic This indicates large availability of information to process by a non-intrusive monitoring point but also stresses the necessity of such a monitoring point from a network manager's perspective. Also, TCP had proven to be a very good alternative for studying the properties of Internet as a whole, as shown by previous studies which used TCP as a vehicle to estimate the network conditions [Paxson 1997a]. While being a good point to start, the subject of TCP traffic analysis is challenging due to the large amount of previous work relating to TCP performance analysis, as presented in the section 2.3. The success of the TCP study depended on the amount of novelty brought in by the proposed method; this is why, aside from making the analysis online, the main focus was to improve the various prior approaches either in terms of robustness or ability to locate a network disfunction, and to propose alternative ways of analysing TCP transfers.

An additional aim was to evaluate real-time traffic performance, by combining the analysis of the two protocols that typically carry such traffic – the Real-time Transport Protocol and the Real-time Transport Control Protocol [Schulzrinne *et al* 1996]. The research within this study elaborated therefore on the weaknesses of a RTCP-only analysis: the lack of localisation details when using the RTCP stream and the reliability of the RTCP information. The solution proposed is a combined RTP/RTCP analysis, which balances the weaknesses and strengths of both types of performance inference. The only issue that obstructed the finality of this strand was the purpose of IP telephony: the application is used to convey voice between two parties, therefore its main indicator is the audio / voice quality, as perceived at the receiving end, and the interactivity of the dialogue. While the second results directly from the delay between the two endpoints, the first depends heavily on the subjective opinion of the human listener and cannot be mapped directly to network parameters.

The aim of the project was to build a technique that would achieve as many goals as possible from the list shown in section 3.1.1. This is where the novelty of the method stands: gather as much network performance information as possible without injecting traffic or modifying / knowing the behaviour of the nodes. The main characteristics of this method were aimed to be: robust, single-point, non-intrusive, online, and able to localise faults. Each of these features was approached by previous studies, but they have not been grouped yet into a single method, mainly due to the (still) evolving trend of network monitoring. The proposed TCP analysis method has a common ground with two main previous tools, Osterman's ***tcptrace*** and Paxson's ***tcpanaly***, but it differs from them in several areas. Firstly, it is designed for online analysis, unlike ***tcptrace*** and ***tcpanaly***, which are both designed for off-line analysis of network traces. This opens the path for continuous monitoring of the network conditions and it allows closing the loop with continuous network adjustments for optimising the levels of traffic. Secondly, extensive effort was put into the area of TCP timestamp options based analysis, a topic unexplored by any previous monitoring approaches: TCP timestamps proved to bring information that allowed a parallel inference of the network parameters, as opposed to the traditional use of acknowledgments. Thirdly, the method proposed infers as much information as possible from the captured traffic: identify time-out losses and estimate congestion window size, as an indication of the connection's health over time.

The main issue of single-point non-intrusive analysis is accuracy: how much can be inferred about the network events and endpoints behaviour that may have caused a certain pattern in the captured packet sequence at the endpoint? The presentation will highlight in this context the permanent trade-off between the robustness of the method and its accuracy, together with the decisions taken.

3.2.2 General description

The structure of the proposed method is common to most per-flow analysis. The first level is capture and parsing, common for most traffic analysers. The input for the analysis consists of the network traffic passing through the point where the monitor is placed. The monitor captures packets belonging to certain flows, based on a packet filter, and then forwards them to a header parsing module.

The packet headers, at the output of this module, are used for the actual flow analysis, which is specific for the two types of traffic. Besides the header fields of each packet, the per-flow analysis uses an object associated with the flow to which the packet belongs, object that holds historical information of the flow which is vital to the analysis. The analysis is interpreting the headers and emulating the endpoint, with the aim of inferring the network events that lead to an observed packet arrival pattern. The output of the analysis is the group of inferred network and connection-related parameters, as encountered by the TCP connection: the RTT delay and average, loss, bottleneck bandwidth, and congestion window evolution. Discrete sets of results, produced at the end of each TCP connection, were used throughout the TCP analysis. For each TCP connection, the output is a single set of parameters, indicating overall figures of the network performance for each connection, and may be used for further analysis, as will be shown in Chapter 6.

If a problem occurs on a network path, the first step is to highlight its existence through the traffic analysis. However, the purpose of network monitoring is twofold: to *identify* and *locate*, both to a certain extent, degradation of a network segment. While the previous section gave an overview of the parameters that can be measured/estimated using the proposed method, which solves the first issue of the problem, this section will describe how these parameters can be used to narrow the location of a possible degradation of the network path in relation to the position of the monitor.

The issue can be discussed starting from the simplistic concept of a *local* network (LAN / WAN) connected via a link to the rest of the Internet, with a monitoring station placed somewhere on this link, then expanding to a generic case. In the case of a LAN with a single Internet connection, there is no transit traffic through the connection point; all the monitored flows have as either source or destination an endpoint in the local network and the other endpoint on the Internet. Based on this division, the Internet network can be split in two sub-networks – the *East sub-network*, and the *West sub-network*, as shown in Figure 3.1

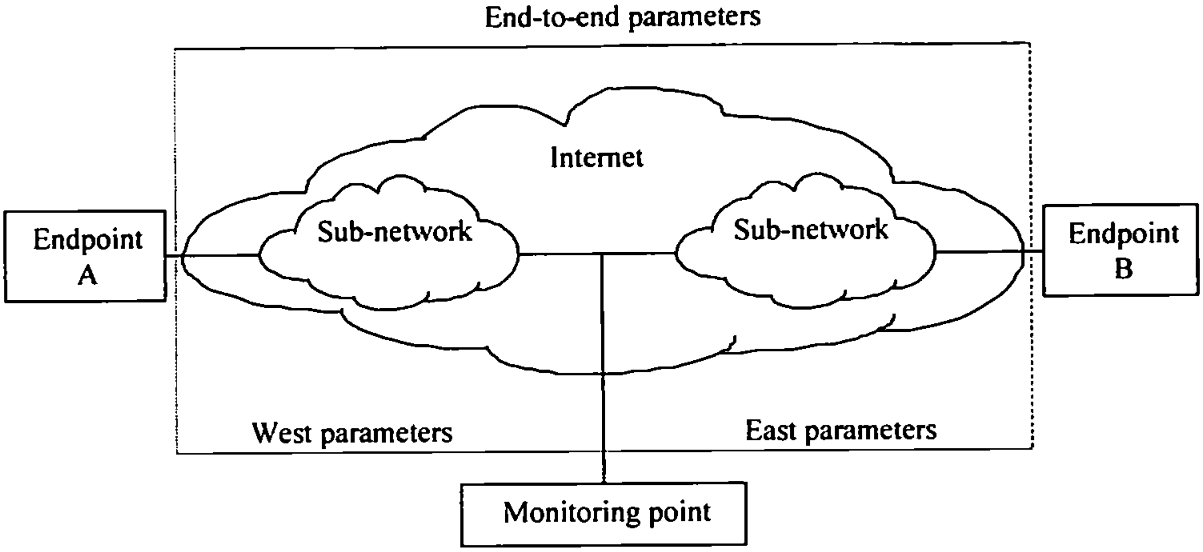


Figure 3.1 - Degradation position in relation to the monitor

With this split in place, the local network may be mapped onto the *West sub-network* and the 'rest' of the Internet, i.e. the collection of network paths and segments that exist outside the local network, would be mapped onto the *East sub-network*. This diagram will be used to explain how the fault localisation is applied when monitoring the real-time traffic (see Appendix A), and the TCP traffic, section 3.4.6.

3.2.3 Impact

The method proposed in this chapter allows a network manager or management system to monitor in real-time the health of the network. The main problem with offline analysis traces is the reaction speed of the management. If a transitory problem is noticed (e.g. the average RTT value doubling for two hours, then coming back to the original figure), there is nothing a network manager can do that would remedy the dissatisfaction of the customers, nor guarantee that whatever measures will be taken will remedy the problem in the future. With online analysis, the management loop may be closed, i.e. decisional factors may be informed continuously about the network status and certain measures may be taken to attenuate / eliminate the problems.

On a different strand, this method may inform systems about the network status for different network paths, so the endpoints, running either a real-time application or a TCP-based transfer, may gather the network status prior to the actual connection. These decisions will all be time based, subject to the network stationarity. Because this is a very wide area, it was reserved for future work, as presented in section 9.2.

3.3 The Real-time traffic monitor

Real-time traffic initially played a very important role in this research project, as the focus was mainly towards assessing the quality of multimedia over IP communications. But, as the project developed, it became apparent that QoS evaluation relates closer to analysis of the vocal signal rather than network research. Therefore, this part was considered to be outside the scope of this project and it was pursued by a parallel research project. The work further studied the relationship between inferred network parameters and the resulting voice quality of a VoIP call and has been summarised in several papers such as [Sun and Ifeachor 2000]. From this perspective, this part would require integration with the voice analysis when looking at the overall aim of this project, i.e. assessing the resulting quality of service for various network applications (where the interpretation of QoS depends on the application studied), but the work included here may constitute part of a future architecture to predict the quality of a voice call based on the network status, measured non-intrusively from a single point. An overview of the real-time traffic analysis, together with the validation tests performed, may be found in Appendix A.

3.4 The TCP traffic monitor

3.4.1 Method – TCP endpoint emulation

The TCP monitor emulates or infers the behaviour of a real endpoint involved in a TCP connection, based on the packets it captures from that TCP connection. As explained in section 2.3.2, there are tools available which attempt to determine the parameters of the TCP transfer by analysing the captured packets and emulate the TCP endpoint behaviour, but the approach used differs in each case. Firstly, the tools mentioned include techniques optimised for offline traces study, such as double pass through each connection, which makes them inappropriate for online analysis. Secondly, they aim to determine the end-to-end parameters of the TCP transfer and do not isolate the location of eventual network degradation in relation with the monitor. Thirdly, both programs are positioned at the extremes from the inference point of view. ***tcptrace*** does not perform any inference on the captured packets to estimate values for e.g. evolution of congestion window[1], timeout losses, or bandwidth. At the other extreme, ***tcpanaly*** includes profiling routines, allowing specialised TCP analysis for each type of TCP implementation. This approach, which is providing very accurate results, relies on prior knowledge of the origin for the captured traffic, unfeasible for monitoring traffic produced by unknown remote endpoints. Finally, it is relevant to mention an additional inconvenient for ***tcpanaly***: some parts of its analysis routines rely on data retrieval from both the sender and the receiver, which makes it inappropriate for single point analysis.

The proposed TCP analysis method is described in the following section. It includes a general approach and architecture designed specifically for single-point online monitoring, aiming to balance between inference and robustness. In particular, the technique introduces several innovative procedures for parameter estimation, such as congestion window inference and timestamp options-based analysis.

3.4.2 State analysis

The TCP monitor is built similarly to a TCP end-client: a state machine, simulating the processing of packets 'sent' and 'received' having as inputs the flags from the captured packets. Meanwhile, the monitor is different from the TCP client for several reasons, mainly due the inputs of a TCP end client, which include user calls, packet arrivals and timer expirations. 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, resulting in some of the transitions, as will be explained, being inferred or unconditional. The presentation will use the term *flow* to describe each of the two directions of a TCP connection. In order to fully understand the transfer of data, the monitor requires the two *flows* belonging to the same TCP connection to be paired in a single entity. The processing will use events such as packet arrivals that appear on one of the directions to modify the current status or / and the parameters of the corresponding direction, named *pair flow* in the discussion that follows.

The states from the original TCP diagram [ARPA 1981b] were maintained, as shown by Figure 3.2, but the transition triggers were modified in order to adapt to the unknown conditions. The transitions in the monitor follow the transitions that happen at the endpoints and they are due to:

A. Packet arrival from the endpoint. Most of the transitions were adapted to this event; in the original TCP diagram, after a transition happens in the client, a specific packet is sent. The

[1] The program evaluates congestion window based on the number of unacknowledged bytes. The technique is accurate if the capturing point is near the sender, but completely unreliable if the capturing is done nearby the receiver.

monitor captures that packet and makes the transition as well.

Example 1. The user sends a 'CLOSE' message to the TCP client, which is in the ESTABLISHED state. The TCP client switches to FIN-WAIT-1 state and sends a packet with the FIN bit set. The monitor, when receiving this packet, also switches the monitored client from ESTABLISHED to FIN-WAIT-1 because the FIN bit is set.

Example 2. The endpoint switches from ESTABLISHED state to CLOSE_WAIT state when it receives a packet with the FIN bit set. After the transition, the endpoint sends an acknowledgment of the FIN. The monitor, when receiving an acknowledgment, checks the state of the corresponding endpoint. If the corresponding endpoint is in FIN_WAIT1 state, it switches the endpoint state from ESTABLISHED to CLOSE-WAIT.

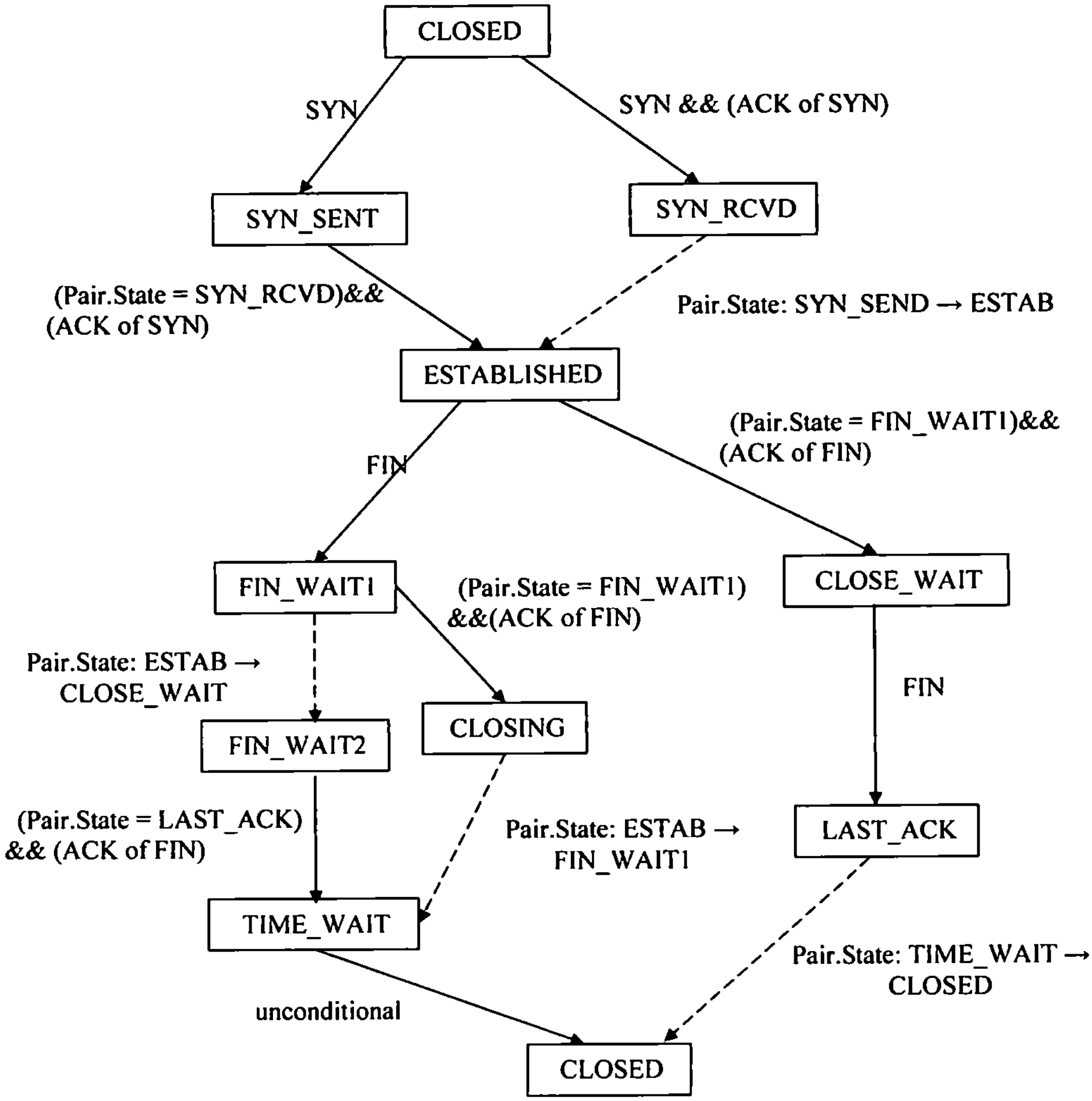
B. Specific transition of the corresponding endpoint. Some of the transitions have a packet arrival or a user call as an input and do not have any outputs, therefore they are performed from the pair flow. Although some of them might not happen as inferred by the monitor, they are bound to happen in the future, prior to any other events, in order for the connection to finish correctly. For example, if a packet that was supposed to trigger a transition is lost after the monitor, it will have to be retransmitted in order to trigger it; the monitor will make the transition when it captures the packet first time, while the receiver will perform it at the second instantiation. Nevertheless, the connection will remain in the same condition until the packet arrives successfully or, in the worst-case scenario, if the packet is lost too many times, it will fail completely.

Example 1. When the endpoint switches from SYN_RCVD to ESTABLISHED, there is no output; this transition is made together with the transition of the corresponding point from SYN-

SENT to ESTABLISHED. If the packet is lost after the monitoring point there will be some supplementary transmissions that indicate this transition but the transition is performed when the sending succeeds. In this case, the monitor transition precedes the endpoint transition.

C. Unconditional. There is one transition (from TIME_WAIT to CLOSED) that was made unconditional, as it is the only one possible from the TIME_WAIT state and, in the TCP client, is due to expiry of a timer. This timer is defined in the TCP specification as 2 x Maximum Segment Life (or MSL), with MSL = 2 minutes. The TIME_WAIT state is only for the endpoint to make sure that the correspondent TCP client received the last FIN packet. While this transition could indicate eventual losses of the last FIN packet, it would keep the TCP object in memory for another 4 minutes, which would affect the memory requirements of the monitor. Additionally, the data transfer finished before this packet was issued, and therefore it does not affect the useful throughput of the connection.

All the above cases can be identified in Figure 3.2, which pictures the state diagram of the TCP monitor



Legend - types of transition triggers:

- SYN /ACK of SYN / FIN – an incoming packet that has the mentioned flag set
- Pair.State = X – the pair flow is in state X
- Pair.State: X → Y – the transition is actually set from the pair flow, when it transits from state X to state Y (that is why these transitions are marked with dotted lines)
- unconditional – the transition is decided at the endpoint by the TCP client and does not have any output

Figure 3.2 - The TCP monitor state diagram

Additionally, aside from the main TCP state diagram, within the ESTABLISHED state, the two parts of the TCP monitor, the 'sender' and the 'receiver' require a state machine:

- Sender state machine:
    - IDLE – the sender received acknowledgments for all the data it sent
    - WAIT_ACK – the sender previously sent data which has not been acknowledged yet
- Receiver state machine:
    - IDLE – the receiver acknowledged all the data it received
    - NORMAL – the receiver is acknowledging data
    - DUPLICATE_ERR – the receiver sent twice the same acknowledgment
    - DUPLICATE_ERR1 – the receiver sent three times the same acknowledgment
    - DUPLICATE_ERR2 - the receiver sent four times or more the same acknowledgment

3.4.3 Sequence analysis

As mentioned, the TCP monitor structure is similar to the structure of a TCP client. Therefore, the analysis performed by the monitor is similar to the analysis performed when an event happens in the TCP client. The phases of this analysis are summarised as follows:

(i) Check flags field and update accordingly the memorised state of the endpoint;

(ii) Check the sequence numbers and update the byte accounting parameters (e.g.: bytes transmitted, useful bytes transmitted). If the packet is apparently out of sequence use the sequence numbers, in conjunction with the Identification field from the IP header to determine if the packet is indeed an out-of-order packet or a retransmission; update the packet accounting parameters (e.g.: lost packets, out-of-order packets);

(iii) Check the timestamp, correlate it with the previous timestamp and with the sequence numbers and update the time-related parameters (e.g.: RTT, jitter).

It is worth mentioning that the Identification field mentioned above has a different functionality in the IP header, where it is used, together with the *More Fragments* flag, to reassemble datagram fragments. Unfortunately, it cannot be a means of identifying losses by itself because it is incremented every time the host sends a datagram and therefore it can be used as a sequencing reference only if the host has only one active TCP connection at a given moment in time (which is not the case for virtually any TCP endpoint).

The analysis applied to the packet depends on its status (whether or not the data from the packet is new, in-order, or out-of-order). The status is determined by comparing the sequence/acknowledgment number in the TCP header of the packet with the sequence variables of the flow to which it belongs. In parallel, the acknowledgment number informs the receiver part of the flow about the status of data sent. In the following, the terms *old* and *future* relate to captured data segments that have a sequence number lower / higher than the expected sequence number of the flow to which they belong.

According to their status, captured data segments were separated into five categories:

- Correct data = segment of data in sequence, following highest sequence number sent, i.e. matching the expected sequence number (IN_ORDER_DATA);
- Future data = out-of-order data; the sequence number of the packet is higher than the expected sequence number (FUTURE_DATA);
- Retransmitted data = old data segment which was transmitted at some moment in the past, and now is retransmitted, probably due to a packet being lost (RETRS_DATA);
- Newer data = segment of data which includes both retransmitted data and in order data;

when detecting a lost segment, TCP can retransmit it by concatenating it with a valid segment; although the number of packets is the same as in a no-loss situation, still, there was a packet lost in the process of transmission, and this event has to be identified (NEWER_DATA);

- Inverted data = old data segment which was mis-ordered (INVERTED_DATA) such that a segment follows after a future data segment, but it is only out-of order, not retransmitted.

In relation to these categories, the method can determine two types of losses: *visible* and *inferred.* A visible loss is detectable using only sequence analysis to estimate a loss. This is possible either because the monitoring point is close enough to the sender in order to see both instances or because the sender transmits other data segments before retransmitting the segment. The second category, *inferred* losses, uses packet spacing instead of sequencing to identify losses. The technique relies on the fact that the sender will always have data to send; because of this, there should never exist a pause larger than $RTT_{average}+4*RTT_{variance}$ (based on a slightly adapted version of [Karn and Partridge 1991]).

The steps of the data analysis are the following: determine what type of data is inside the packet, determine idle periods, calculate delay-related variables, and estimate the congestion window. The processing uses two arrays of information, named *SkipData* and *BufferData* in the following discussion and in section 3.5.1, when discussing the implementation. The role of *SkipData* is to memorise past intervals of sequence numbers that appear to be missing from the captured flow; *BufferData* maintains records of past captured data segments. Throughout these steps, the flow variables are updated. The following list describes each of these steps.

(i) Determine what type of data is inside the packet.

This step uses the expected sequence number and the *SkipData* structure to identify the *visible* losses. The first check compares the sequence number of the captured data segment with the expected sequence number. If the received sequence number is higher than the expected one, then the segment contains in order data or future data. Else, the segment carries old data, and the *SkipData* list in conjunction with the Identification information are required to make a distinction between mis-ordered packets, packets lost before the monitor and packets lost after the monitor. The following checks are performed:

- If the segment is not listed in *SkipData*, then is a lost-after-monitor segment (*LostAfter)*– it was transmitted previously and both of the instantiations were captured;
- If the segment was already acknowledged by the receiver (i.e. the expected sequence number of the receiver is equal to or higher than the last sequence number contained in the data segment), then the packet was erroneously retransmitted by the sender (*RTOError*), although no loss happened. This may be due to the acknowledgment that indicated the successful receipt of the segment being either lost or to a late (arriving to the sender after the retransmission timeout expired).
- If the segment is listed in the *SkipData*, then its Identification field, from the IP header of the incoming packet, is compared with the Current ID (which indicates the most recent value of the Identification field sent) of the flow:
    - If Identification is lower than Current ID, the packet is a genuine out of order packet (*Inverted)*, as the sender had released it before sending the previously received packet (which Identification updated last time the flow Current ID);
    - If Identification is higher than Current ID, the packet is a retransmission of a lost-

before-monitor segment (*LostBefore)*, as the packet is in order from the IP point of view (was transmitted after the previously received packet).

(ii) Determine idle periods.

In addition to the above analysis, additional checks are performed to determine whether or not the current packet arrives after an idle period. An idle period is defined in this context as time during which neither the sender, nor the receiver, have any outstanding data segments or acknowledgments to transmit. A sender reaches this state if there is no more incoming data from the application it serves; the receiver in this state acknowledged all the data it received. It is not a case mentioned in the traditional TCP behaviour modelling literature, where the sender always has data to transmit [Ott *et al* 1996] or it transmits all the available data without any pause. Nevertheless, such behaviour, i.e. with pauses within the connection, is not uncommon in a real-life situation, even for file transfer protocols such as HTTP and FTP. The best example of such a situation is a TCP connection providing data to an HTTP 1.1 web browsing application. HTTP 1.1, as mentioned in section 2.1.3, uses a single pipelined TCP connection to transport all the objects from a page / website, as opposed to HTTP 1.0, that opens a connection for each object. In the HTTP 1.1 case, when a user browses a website, there will be large gaps within the established TCP connection, for example due to the browsing activity: the user opens a page, reads its content, then follows a link to a next page. This does not impact on the overall TCP dynamics, but it affects performance evaluation of the transfer. An example is given in Figure 3.3.

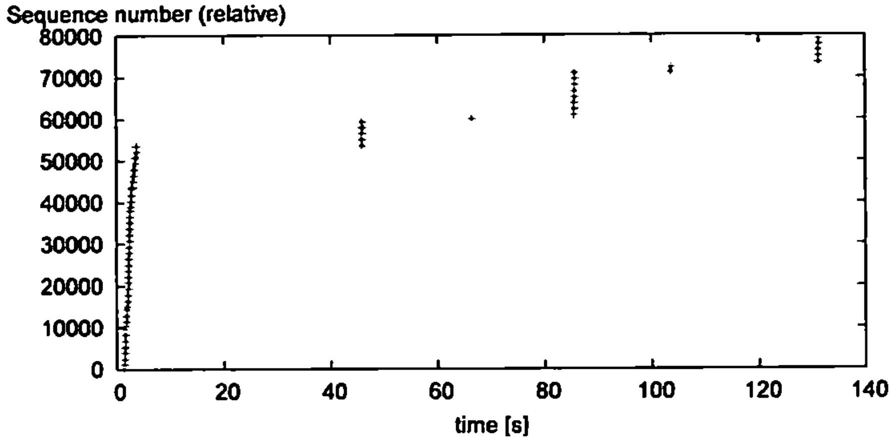


Figure 3.3 – HTTP 1.1 session

The above figure displays the trace resulting from a 6-page web browsing session. Each of the transfers was normal, with a high throughput, but the analysis would reveal a very low figure, due to the pauses that the user made on each page. This is why such a connection must be properly analysed; the monitor, in this case, has to account for each of the mentioned gaps and to eliminate that time from the total time of the transfer.

The method to determine idle periods is based on identifying the gaps between different object transferred. A period between two packets within a TCP connection is considered as idle if:

(a) The time elapsed between the two packets, Δt, satisfies the rule

$$\Delta t \geq max(2RTT_{end-to-end}, 2\ sec) \qquad 3.1$$

The 2 second limit aims to eliminate from this counting the gaps between a train of acknowledgments and the next train of packets, sent due to these acknowledgments, while the 2RTT limit protects the high-delay connections from the 2 second rule. This timing information has an impact on the timeout estimation: if a packet timeouts several times, enough to go above the 2 seconds interval, it will be considered as coming from a different object, instead of being lost. It is an accuracy trade off but, on the other hand, such gaps will happen within a normal HTTP 1.1 web browsing session, while repeated timeouts are rather exceptional events [Paxson 1997a].

(b) Both the sender and the receiver are in IDLE state, as defined in 3.4.2, when capturing this packet. This rule reduces the impact that idle periods can have on the overall performance estimation if a connection encounters a severe network outage; when there is still data pending to be acknowledged or retransmitted, the connection cannot be during an idle period.

(c) The packet has a non-zero data length. A new object transfer has to start with new data being sent.

Special attention was given to the first idle period within an HTTP connection. A typical HTTP retrieval starts with the following data exchange between the server and the client, see Figure 3.4.

```
t1 - Client: GET /index.htm HTTP/1.1 [HTTP tags]
t2 - Server: [empty ACK for the GET request]
t3 - Server: HTTP/1.1 200 OK [HTTP tags, packet content]
```

Figure 3.4 – Initial GET request from a client

The difference (t_3 – t_2), where t_i is the capture timestamp of the packet i, represents the *server response time*. In the ideal case, this time difference should be virtually null if the capturing point is positioned right at or nearby the server. However, if the server is busy, it may be substantial and the server might issue first an empty acknowledgment (either immediately or delayed, depending on the prior evolution of the associated TCP connection) and produce the response only later on. In any case, the estimate indicates the server performance as part of the overall TCP performance; it does not affect the dynamics of the TCP traffic, but it has the same impact on the observed performance (i.e. "How fast a page is loaded") as degradation in network conditions.

(iii) Calculate delay-related variables.

The RTT estimation is based on matching the acknowledgment number carried by the packet with a last sequence number from a buffer location. The rationale behind it is that an acknowledgment is produced by the receiver immediately when it gets a data segment eligible for acknowledgment (in fact it is every other data segment, as receivers - should - use delayed acknowledging). The acknowledgment is valid for RTT estimation if the data segment was not acknowledged before and the segment was not retransmitted; the two conditions ensure that the acknowledgment was produced in normal transfer conditions, and not due to losses or mis-ordering, and they are similar in a real TCP client. The RTT estimation uses the *BufferData* to obtain historical information:

(a) The acknowledgment number is searched in the list.

(b) If the last sequence number of a segment (i.e. the segment's sequence number plus its length) is equal to the acknowledgment number less 1 unit (because the acknowledgment indicates the next expected sequence number), the acknowledgment was sent for that segment; in addition, following checks are performed:

- The retransmission counter of the segment must be zero;
- The acknowledgment counter of the segment must be zero.

(c) If the segment found satisfies all the conditions, the RTT is determined as the subtraction between the capture timestamp of the acknowledgment minus the timestamp of the data segment. Then, the RTT jitter is determined as the unsigned difference between current RTT value and previous RTT value. The formulas for the two parameters are given in section 3.4.5.

(iv) Estimate the congestion window.

The third step, estimating the congestion window, includes an element of uncertainty, as will be further detailed in section 4.2.1. A visual presentation of the congestion window in time may be produced by plotting the sequence number for each segment against the capture time. Such a plot is presented below in Figure 3.5.

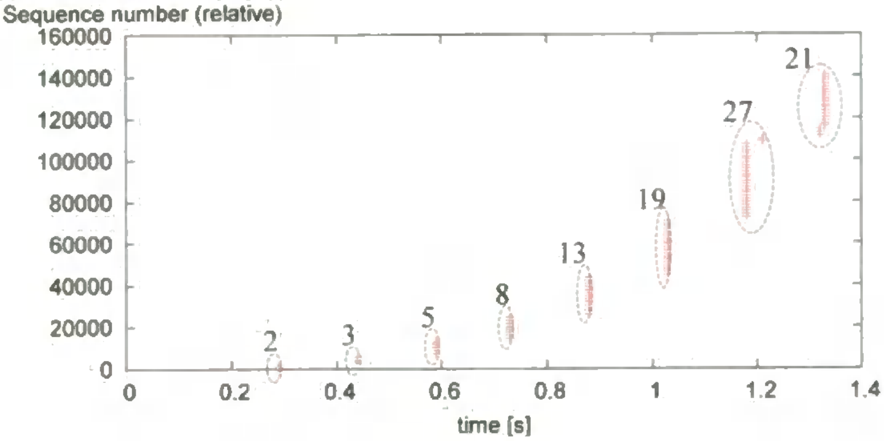


Figure 3.5 - Illustrating connection window evolution through sequence numbers

The dotted ellipses indicate which packets were likely to have been sent in the same congestion window; to visualise better, beside the ellipse is also written the number of packets in that particular congestion window.

The ideal method would be to emulate the four TCP algorithms [Stevens 1997] at the monitor, and modify the congestion window accordingly, but this would be difficult to achieve, mainly because of the unknowns in the sender behaviour. Two other approaches were considered: estimate the congestion window as the number of data packets in flight at a moment or estimate the separation between different congestion windows using the time elapsed between two consecutive captures of data segments belonging to the same flow. The following paragraphs will describe each of the mentioned methods. The notion of *flight (of packets)* will be used, with the meaning of a sequence of packets all transmitted within an RTT, as defined in [Paxson 1997a].

1. Emulate the behaviour of the sender

   TCP, as presented in 2.1.2, includes four algorithms for performance optimisation. Based on capturing the data segments and acknowledgments of a connection, the monitor emulates the evolution of the congestion window and determines a congestion window estimate (CWE), as it happens at the sender. The summary of this type of emulation is:

   - slow start mode: start CWE at 1, when first data segment is captured and every time an acknowledgment / a data segment is captured, increment / decrement CWE;
   - congestion avoidance mode: increment CWE by 1/CWE every time an acknowledgment is captured;

- loss (detected via multiple duplicate achnowledgments): reduce CWE to ((CWE/2)+3) segments and start congestion avoidance. Increase the CWE by one segment for each duplicate acknowledgment received;
- loss (detected via a timeout): reset CWE to 1 segment and then increment in slow start mode.

The above algorithm, offers not an estimate, but the actual value of the congestion window, under ideal conditions (full knowledge of the packets that arrive at the sender and of its TCP behaviour). The method was successfully implemented in [Paxson 1997b], but only by considering two essential aspects. Firstly, in order to correctly follow the evolution of the TCP sender, the capturing point had to be placed right at that endpoint. Secondly, the analysis program was aware about the alterations from this policy existing within the TCP implementation studied. None of these aspects exists within the scope of this project; the proposed method should not be limited to known senders or/and known conditions at the sender. For these reasons, the algorithm emulation was considered to be unsuitable for congestion window estimation within this analysis method.

2. Identify transmission window 'peaks'

The second type of congestion window inference, based on the number of packets in flight, is less demanding compared to the one described above. It is performing inference instead of emulation: it does not make any assumptions about the sender behaviour, but it tries to estimate the number of packets in flight. The rationale behind this is that the size of the congestion window is equal with the number of packets in flight, as the sender can transmit up to the minimum of the congestion window and the advertised window [Stevens 1997]. In order to determine this number, the monitor determines the 'peaks' of the number of unacknowledged packets evolution by comparing each three consecutive values of them. A better understanding of the method can be gained by looking at a graphical example. Figure 3.6 presents the evolution of the number of unacknowledged packets, as seen by the monitor; the updates are performed each time when a data/acknowledgment packet is received.

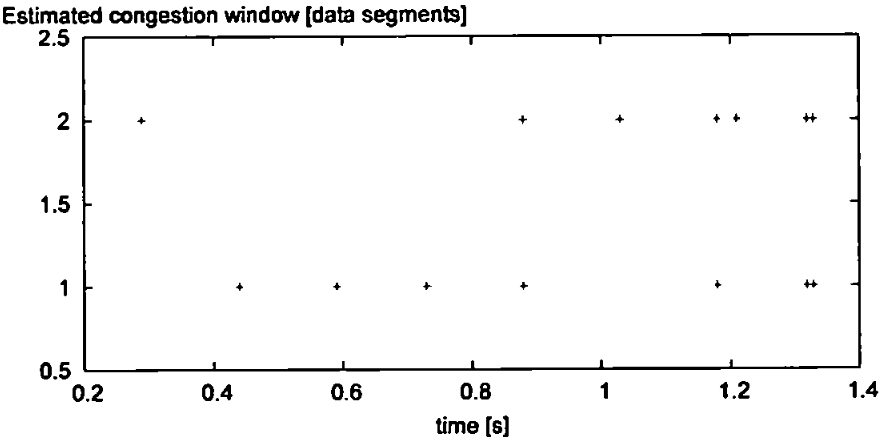


Figure 3.6 - Estimated congestion window based on identifying transmission window peaks

It may be seen that the number of unacknowledged packets oscillates between 1 and 2 packets. This is because, in this case, the position of the monitor was very unfavourable for this method: at the receiver. Because of this, the monitor will see the acknowledgments being sent every other packet the latest, leading to an estimate oscillating between 1 and 2. The method is clearly more robust when compared to the previous but is, clearly, prone to errors, due to the uncertainty introduced by the position of the monitor.

3. Identify packet trains – time based

The third proposed option to determine the congestion window is robust, but less accurate, particularly for long file transfers. It is based on the observation that, for short data transfers and high values of the (bandwidth x delay) product, the sender never achieves efficient utilisation of the available bandwidth, due to the speed of the slow start algorithm that, even though it increases the congestion window exponentially, has a low number of data segments to reach full utilisation. As a result, a sender transmits data segments belonging to the same round in an almost back-to-back fashion, distanced by $\Delta t_{\text{intra-train packets}}$ evolution visible in Figure 3.5 between the circled groups of segments, and then it waits for the receiver to acknowledge these segments. Because of the delay between successive data segments belonging to different flights, $\Delta t_{\text{inter-train packets}}$, they can be visually observed in the figure. Translated in numerical terms, the method must identify this gap by comparing the inter-arrival delay with a defined value, using a relation such as equation **3.2**

$$\Delta t_{inter-train\,packets} >> \Delta t_{intra-train\,packets} \qquad \mathbf{3.2}$$

This best value for $\Delta t_{\text{inter-train packets}}$ was considered to be RTT/2, considering that a flight of packets is all being sent back to back at the beginning of an RTT period, then, approximately one RTT later, when the acknowledgments arrive, the sender transmits the next flight[1]. The result of this estimation, applied on the transfer from Figure 3.5, is shown in Figure 3.7.

[1] Choosing a figure too low may lead to incorrect splits in the middle of a train of packets; a figure too high might erroneously gather segments from more than one packet train.

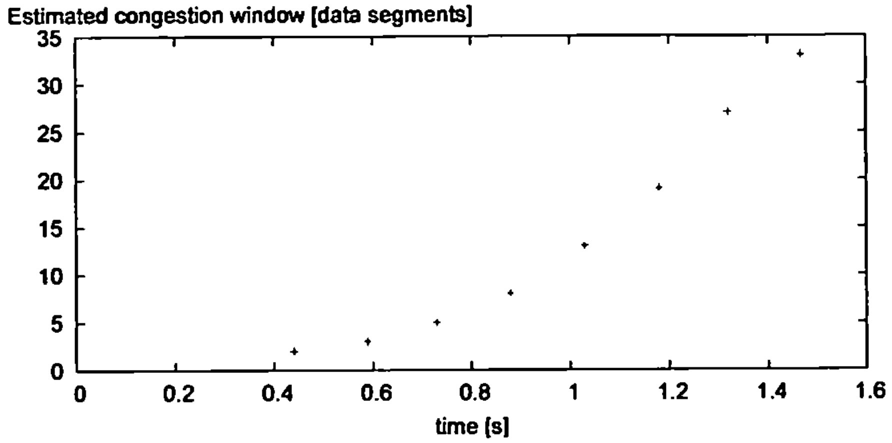


Figure 3.7 - Estimated congestion window based on identifying packet trains

The problems for this method come from a different direction: the TCP functionality itself. It is apparent from the TCP specification [ARPA 1981b], that its primary aim is to fill the available bandwidth for each connection. However, in the real case TCP is not aware of the network conditions at the start of the connection. Therefore it has to probe the available bandwidth through the slow start algorithm, and only then to stabilise at the resulting throughput, not always accurately, during the congestion avoidance phase. It is due to this unawareness that the proposed congestion window estimation works: the TCP client is still in the slow start phase or had a packet loss rate that forced it to keep the congestion window low. As a result, the TCP client does not fill the available *bandwidth x delay* product, which leads to the spacing between windows visible in Figure 3.5

The main drawback of the method is that it is suited only for short connections. As the size of the congestion window increases, the channel reaches its limits, increasing the probability of missing the inter-packet-train gaps. More details about this drawback are presented in section 4.2.1. In addition, another limitation of this method is that it produces the estimate of the congestion window based on the distance between packets *as they arrive* at the monitor. If the monitor is placed by the receiver and the distance between packets belonging to the same train is substantially altered because of queuing, the method will not be able to identify successive packets from the same window. Ideally, the monitor should have information about the distance between successive packets *as they leave* the sender, regardless of its position. This limitation is solved in the following section, 3.4.4.

3.4.4 Timestamp options-based TCP analysis

A parallel analysis of the network performance and sender parameters is performed using the timestamp options from the TCP header. Timestamp options were defined as an add-on of the TCP standard in [Jacobson *et al* 1992], and one of their main purposes is to allow the sender to obtain a better estimate of the RTT. The RFC defined two timestamp fields included in the typical timestamp option: the *TSval* (timestamp value), which is obtained from a local clock, and *TSecr* (timestamp echo reply), which echoes the *TSval* of the most recent data segment received. The monitor uses these exchanged timestamp values in order to determine the RTT, as defined in section 3.4.3, and bottleneck bandwidth.

The timestamp-based analysis uses the same information objects as the previous (the flow-related object, containing the current information, as well as the *BufferData* and the *SkipData* structures). The differences appear when pairing the correspondent packets from the two directions of a connection. While in the 'traditional' method, this mechanism was based on the acknowledgments produced in response to data packets, with this method it is the timestamps that allow identifying which packets were sent as response to certain packets, and determine the RTT value. The *BufferData* must store, besides the boundaries of the data segment, the *TSval* and *TSecr* values for each TCP segment. When a data segment is captured, if it is an IN-ORDER packet, the value of its *TSecr* field is searched among the *TSval* values. The *TSval* values are retrieved from the historical information stored in *BufferData*; the search starts with the most recent segment to minimise the number of searched locations. When the segment is found, the RTT is computed as the difference between the capture timestamps of the two packets. The method has the major advantage of estimating RTT for data segments sent in response to acknowledgments (fact that was impossible for the normal sender-receiver configuration, as all the TCP segments acknowledged the same sequence number, i.e. Initial Sequence Number+1 for the acknowledging flow); this radically increases the accuracy of the overall RTT estimation in the cases where the monitor is placed near the receiver, as will be described in section 4.1.

3.4.4.1. Congestion window

TCP-timestamps-based analysis may also contribute to the congestion window analysis. The values carried by *TSval* are directly proportional to the time when the packet was transmitted by the sender. The only information required to convert the difference between these values into time is the resolution of the TCP timestamp of the sender; because the information is needed during the connection, the SYN-ACK resolution estimate is used. The resulting values for inter-packet spacing are then compared with RTT/2 to identify the trains of packets *as they leave the sender*. This provides a better congestion window estimate,as the TCP timestamps are not affected by the queuing between sender and monitor.

3.4.4.2. Bottleneck bandwidth

One of the studies laying the foundation for bottleneck bandwidth estimation was [Shenker et al 1990], where the distribution of packet inter-arrival times was analysed, then continued in [Keshav 1991], where the *packet pair* method was proposed to intrusively determine bottleneck bandwidth. The method was further refined in [Mogul 1992] and, more recently in studies such as [Lai and Baker 2000]. The theory behind all these alternatives relates to measuring the spacing between packets and estimate, based on this measurement, the bottleneck bandwidth. It considers a pair of (generic) packets of size *x* bytes sent back-to-back from an endpoint A and received at an endpoint B. When the ACKs arrive at host A, due to the bottleneck segment, the spacing between them is changed to $\Delta t > 0$. The rapport between the size of the packets and the resulting spacing gives the bottleneck bandwidth:

$$\textit{bottleneck bandwidth} = \frac{x}{\Delta t} \qquad 3.3$$

There is at the moment a variety of tools for bandwidth measurement, virtually all of them focused on active measurement, as described in section 2.3. The nearest to the scope of this project can be considered the *Packet Bunch Method* (PBM) proposed by Paxson in [1997a], in which he identified 'bunches' of data segments using monitors both at source and destination and fully following the behaviour of the sender. The aim of this project is to have a single point estimation, and, as previously mentioned, this eliminates the possibility of establishing the sender behaviour (additionally, the actual implementation of the sender is unknown, which complicates matters). In all the cases, the problematic quantity is the inter-send time between successive packets at the sender; most of the approaches eliminate it by sending back-to-back packets (which also increases the probability for these packets to be queued sequentially in the routers), or, in PBM case, determine what this value is by capturing the data at the sender as well. This project proposes a novel approach, which is gathering this information from the timestamp options. Indeed, the *TSval* is produced based on the local clock of the sender; although there is no absolute timing information, due to the fact that the clocks of the sender, monitor and receiver are not synchronised, there is still relative information that can be gathered by subtracting the *TSval* values from successive packets. This is how, without producing any additional traffic, the monitor is able to infer which of the data segments sent by an endpoint may have been transmitted back-to-back. For satisfactory results, the resolution must be high enough to differentiate between two separate 'bunches' of back-to-back packets and, if necessary, also low enough to be unable to distinguish between two back-to-back packets belonging to the same 'bunch'. Section 4.4 provides a detailed overview of the resolution estimation for the TCP timestamps, as well as the limitations that the proposed method has.

3.4.5 The measured parameters

This subsection details the actual variables that are measured/inferred using the presented method. There are two main categories of parameters resulting from the analysis: network-related and performance-related.

(i) Network-related parameters. This category of parameters aims to describe the status of the network path transited by the packets belonging to a flow. This category includes:

- RTT average and RTT variation – together aim to inform about the delay encountered by the packets when transiting the network.

$$RTT_{average} = \frac{\sum_{i=0}^{n-1} RTT_{estimation\ i}}{n}, \text{ where n is the number of estimations} \qquad 3.4$$

$$RTT_{variation} = \frac{\sum_{i=1}^{n-1} \left| RTT_{estimation\ i} - RTT_{estimation\ i-1} \right|}{n-1} \qquad 3.5$$

- Bottleneck bandwidth (*MinimumBW*) – represents an estimator of the lowest bandwidth segment from the network path transited by the data segments. It can be determined only if the TCP endpoints are producing the TCP timestamp option (see section 3.4.4.2):

$$\overline{BW} = \frac{1}{m}\sum_{j=1}^{m-1} BW = \frac{1}{m}\sum_{j=1}^{m}\left(\frac{1}{n_m}\sum_{i=1}^{n_m-1}\frac{l_{i,m}}{(t_{(i+1),m}-t_{i,m})}\right) \qquad 3.6$$

where:

m – the number of packet trains observed

$l_{i,m}$ – the length of packet *i* from the mth train of packets

$t_{i,m}$ – the timestamp of packet *i* from the mth train of packets

n_m – the length (in packets) of the mth train of packets

- Lost packets – includes three component losses:
  - *LostBefore* – packets lost on the path segment between sender and monitoring point
  - *LostAfter* – packets lost on the path segment between monitoring point and receiver
  - *Timeout* – packets lost and retransmitted due to a timeout

Each of them represents the loss on one of the path sub-segments; their sum is an inference of the end-to-end losses, either true or due to erroneous behaviour of the TCP sender. In addition, a subset of the *LostAfter* category is determined: *RTOErrors* (retransmission timeout errors). These are data segments that were retransmitted although an acknowledgment to confirm their receipt was captured by the monitor in the time between first and second instance of the data.

- Out-of-order packets (*Inverted*) – the number of genuine mis-ordered packets;
- Total number of transmitted packets (including retransmissions);
- Timing information:

- o Connection time – the time elapsed between the first and the last transmitted packets
- o Data transfer time - the time elapsed between the first and the last transmitted data packets
- o Idle time – the total amount of idle time within the duration of a connection

(ii) Performance information (throughput and duration) – all the parameters below are obtained using the connection / data transfer duration **minus** the idle times. This ensures that the resulting figures reflect directly the performance of the TCP transfer, as described in 3.4.3.

- – Valid data throughput – the amount of valid data that was received (obtained by subtracting last transmitted sequence number and initial sequence number) reported to the duration of the data transfer
- – Connection duration – the time elapsed between capturing the first packet and the last packet from the connection
- – Data transfer duration – the time elapsed between capturing the first data packet and the last data packet from the connection

3.4.6 Fault localisation

The previous section provided a list of parameters that may be obtained from the TCP analysis. In turn, these parameters may be mapped onto the two logical subnetworks defined in section 3.2.2, to give an indication about the *location* of the measured parameters. As most of the Internet traffic today is based on *client-server* communications (e.g. HTTP), the generic model of a connection can be considered the *sender-receiver* case, where one of the endpoints (Endpoint A in Figure 3.1) sends data and the other one (Endpoint B in Figure 3.1) acknowledges it. With this configuration, the loss and delay parameters were split as pictured in Figure 3.8 This model also matches peer-to-peer downloads, which are on the increase in current network usage figures [Fraleigh et al 2003]:

- RTT delay - RTT West and RTT East
- loss – outgoing West loss, outgoing East loss, incoming East loss, incoming West loss

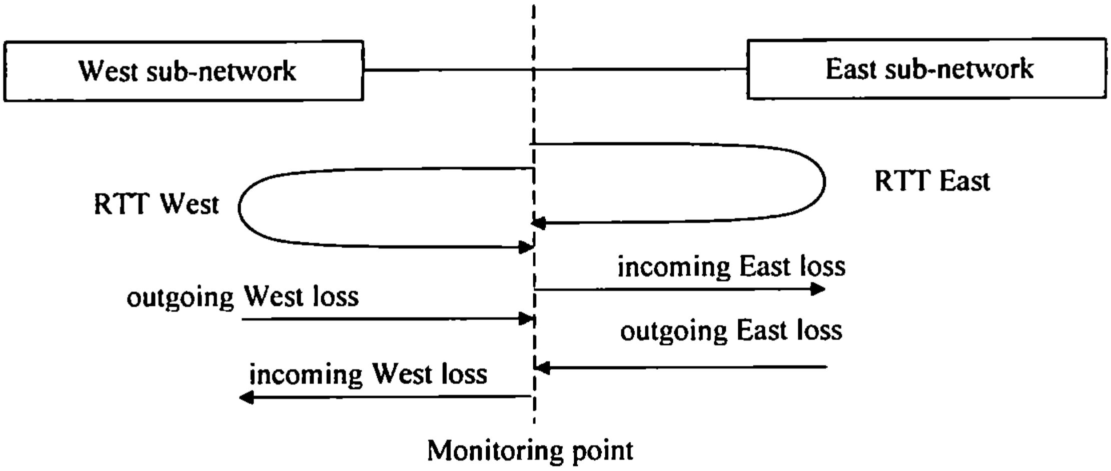


Figure 3.8 – The components of loss and delay for the monitoring configuration

Using the notations from Figure 3.1 and Figure 3.8, for Endpoint A acting as sender, the following parameters may be inferred:

- Subnetwork West A→B packet loss rate ⇔ outgoing West loss ← packets lost before the monitor
- Subnetwork East A→B packet loss rate ⇔ incoming East loss ← packets lost after the monitor
- Subnetwork West A→B packet inversion rate ⇔ outgoing West misordering ← inverted packets

- Subnetwork East RTT ⇔ RTT East ← RTT measured
- Subnetwork West RTT ⇔ RTT West ← RTT measured of the reverse flow (with a lower confidence factor, as described in 4.1)
- Subnetwork West A→B ⇔ West bottleneck ← estimated bottleneck bandwidth

Additionally, if B sends data as well, there can be also determined:

- Subnetwork West B→A packet loss rate ⇔ outgoing East loss ← packets lost before the monitor
- Subnetwork East B→A packet loss rate ⇔ incoming West loss ← packets lost after the monitor
- Subnetwork West RTT ⇔ RTT West ← RTT measured
- Subnetwork East B→A packet inversion rate ← inverted packets
- Subnetwork East B→A ⇔ East bottleneck ← bottleneck bandwidth

3.5 Method implementation

3.5.1 Block diagram

The primary aim of the monitor is to reveal an estimative picture of the network performance, as encountered by the analysed packet flows. The common part of the monitor performs only the header parsing and forwards the information to the analysis blocks, together with prior information, retrieved from a connections database. A schematic view of the monitor is given in Figure 3.9. The analysis is triggered every time a packet is captured.

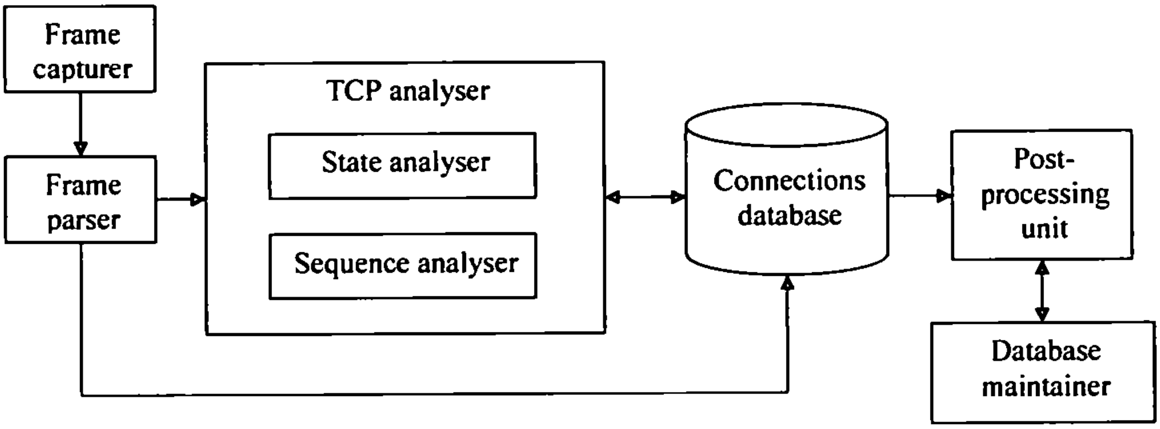


Figure 3.9 – Traffic analyser - main blocks diagram

The **Frame capturer** and **Frame parser** components have the obvious role to capture the frames passing through the monitoring point and decode their content. The output of the **Frame parser** is used for two purposes: to start the analysis of a current frame by the TCP analysis and to retrieve information from the **Connection database** about the flow associated with the current packet.

The **Connection database** is a structured collection of objects, each of them containing information about each analysed flow. It uses the quadruple (source IP address, source TCP port, destination IP address, destination TCP port) as a means to uniquely identify a TCP flow at a moment in time. For each monitored flow, the database maintains an object that memorises its relevant data. The information held can be split into two headings: connection-related parameters and performance parameters,. The connection-related parameters are used in the TCP analysis, which inform about the initial and current status of the connection; the performance parameters are the actual outputs of the analysis and will be forwarded, when requested, to the **Post-processing unit.**

Although a TCP connection has two flow objects associated with it, one for each direction, analysis of a packet belonging to any of the directions requires both objects. The pair flows have to work in conjuction within the monitor, for several reasons. Firstly, some of the transitions of a flow can be made only based on packets sent by the other flow in the pair. Secondly, acknowledgment numbers of a flow acknowledgment data sent by the pair flow. Thirdly, RTT can be computed only by combining the timestamp of the sent data packet with the timestamp of the returned acknowledge. In fact, in the actual TCP client implementations, the two flow objects are paired in a single structure [Stevens 1995], but the monitor has a different view of each connection. While an endpoint maintains information about one sender and one receiver (the ones at that endpoint), the monitor must keep the information regarding two senders and two receivers (i.e. a sender-receiver pair for each of the endpoints). It was therefore preferred, for clarity reasons, to keep the two directions of a connection separate.

The **TCP analysis** block uses the two inputs (the headers of the current packet and the past information) to update the current details of the flow and it uses the sequencing and timing information provided by the current packet to update the parameters of the associated flow from the **Connection database**. The analysis requires two blocks: a **State analyser**, required to infer the current state of the connection according to the TCP state diagram, and a **Sequence analyser**, used to evaluate and update the current status of the data transfer.

The **Database maintainer** periodically polls the database and removes the flows which are in CLOSED state. It uses a timer to decide whether a flow should be kept in the database, using its inactivity period - the time elapsed since capturing the last packet associated with this flow. This is a critical block for the traffic analyser, as it removes unused flows from the database, therefore keeping the memory usage to a low value. The actual value of the timer results as a balance between the TCP and HTTP timers on one side and the memory limitations on the other side. As defined in [ARPA 1981b], a connection can be kept open indefinitely, as long as senders generate periodically packets to check whether or not the connection is still opened at the corresponding endpoint; according to [Braden 1989], the duration between two keep-alive packets should be at least 2 hours (the figure of the monitor trigger was doubled to account for clients with longer keep-alive timers). Based on this specification, the monitor should set the flow removal timer to a value of 2 hours, which would increase massively the amount of memory that the monitor requires. During the implementation, the preferred value was of 5 minutes, which, although it might remove such connections, it considers another type of inactivity periods. These are the ones produced by HTTP version 1.1, which may re-use the same connection to transfer several objects, behaviour detailed in section 3.4.3 (subject to the duration of the monitoring and the amount of traffic captured).

The **Post-processing unit** takes the role to take the information from the **Connection database** and either provides it to a control entity or display it to the user. Depending on the chosen output method, the performance information may be presented either continuously or discretely. For continuous output, the current values for the network parameters are presented every time a packet is received; in this case, the post-processing unit will output performance data after every captured packet. For discrete output, the output is triggered indirectly from the analysis blocks every time a flow is closed. The continuous output is more appropriate for real-time flows, as they may last for a long time and the instantaneous or short-term performance average values are more relevant than the long-term averages. On the other hand, discrete output is the preferred form of sampling for TCP flows, as their performance is best evaluated at the end of the connection.

The **Frame capturer** component was implemented using the *pcaplib* packet capturing library [LBNL 2003], library used for most of today's network analysis programs. The library includes functions to get a handle of the capturing device, filter the captured packets (e.g. retain and forward for analysis only the http traffic - TCP port 80 - from the entire traffic passing a monitoring point). The output of the frame capture is a structure including a pointer to the captured frame, time of the capture and the length captured (for TCP analysis, only the information up to and including the TCP header is used, and the actual data content of the packet is discarded). The **Frame parser** is called with this mechanism and, from the raw frames provided by the Frame capturer, determines the fields of the IP and TCP headers. These two preliminary phases are common for most of the available TCP monitors / analysers / parsers, such as the well-known *tcpdump* [Jacobson 2003b],

As the monitor has to work in real time, it is important that the retrieval of the information related to a connection to be fast, on one side, but also, from the same reason, the database must be efficiently organised, in order to occupy as little space as possible in memory. The first requirement, i.e. speed, points towards an array, with each location uniquely identifying a TCP connection; but this would require an array with, roughly, $255^8*65535^2$ locations to encompass for all the possible combinations of IP addresses and ports quadruples, which is unfeasible for the obvious physical memory limitations. The second requirement, space, can be satisfied using a chained list of the current connections, which would impede dramatically the performance if the number of monitored connections were high. A combined hash-based and chained lists indexing was chosen as a compromise solution to satisfy both of these conditions. As a result, the connection database is organised as an array of hash values, and each of the locations of the array points to the beginning of a chained list of objects. The mechanism of determining the location of the flow is a two-step one. First, a hash value, *HashID,* from the (source IP address, source TCP port, destination IP address, destination TCP port) quadruple is determined. The hashing function treats the quadruple as a 12-byte string split into 6 16-bit integer values. It then produces a number, *HashValue* by summing these integers. The hash value is calculated as the remainder of the division between the *HashValue* number and the size of the table. Second, the chain list at location *HashID* is followed in order to find the requested object. The size of this table, *HashTableSize*, was set to 1024 in order to avoid the chained lists becoming too long.

The **State analyser** module has the role to follow the evolution of TCP connections, as a state machine. The module infers the evolution of a TCP client based on the packet arrival events or timer expirations. The functions of this module implement the method described in section 3.4.2. The need for this module comes from two directions, relating to evaluating the duration of the TCP connection. The first reason is to correctly determine the start of the data transfer. Secondly, knowing the connection lifespan optimises the resources usage of the monitor. Once the connection is successfully completed, its associated object can be discarded and the summary of the connection either saved or forwarded to the **Post-processing unit**.

The **Sequence analyser** module implements the algorithms presented in sections 3.4.3 and 3.4.4. Its purpose is to infer the events, such as latency and dropped packets, that led to the captured sequence of packets and to extract the corresponding network parameters from the monitored connection. In order to fulfil its objectives, the module includes several variables:

- *BufferData* – a circular array that memorises the sequence, timing, acknowledgment, and retransmission information for last 256 data segments captured. The roles of this array are to identify whether there were any (multiple) retransmissions and to provide timing information for RTT calculation (particularly for TCP timestamp-based delay inference)
- *SkipData* – a circular array that memorises the last 10 intervals of data that appears to have been skipped. This array helps to separate between misordered and retransmitted data segments. The array has a complementary role to the *BufferData*. Its functionality may be replaced by *BufferData*, but it was maintained as a separate

structure for speed and simplicity reasons, as the BufferData memory is larger. It also provides a mechanism to determine whether there might have been any packets dropped by the interface during the capturing. This is because, once TCP closes gracefully a connection, all the data transferred was correctly received by the remote end. If there still exist any entries in the *SkipData* list after such a connection, it is likely that those segments were dropped in the capturing process, only by the interface, rather than dropped by a router.

- *PacketStatus* – a variable that characterises the data segment contained in the packet, according to the categories defined in 3.4.3 (IN_ORDER_DATA, FUTURE_DATA, RETRS_DATA, NEWER_DATA, INVERTED_DATA)

A third option, not fully explored during the project but important as a future research avenue, is to determine whether or not the connections follow the correct evolution, in order to identify potential faulty TCP implementations. This direction relates more to the overheads that TCP clients must sustain than to the network conditions. A TCP server that does not implement the TCP state machine correctly will overload its memory and/or the network. Firstly, by not gracefully closing the TCP connections, the endpoint will have to memorise a larger number of associated objects. Secondly, the endpoint will create additional traffic due to forced closing of connections (e.g. through RST – reset packets).

3.6 Summary

This chapter presented a method that allows non-intrusive, single-point, online analysis of traffic. The description of the method started with an overview of the requirements that current network analysis has from monitoring and it built a list of characteristics based on these requirements, which was used as a list of aims for the developed traffic analysis.

The presentation continued with a detailed view of the assumptions and processing involved when analysing TCP connections. The discussion described the two main parts that form the TCP analysis: the emulation of the TCP state machine and the inference of network events. The proposed method is a combination of elements from previous studies, such as TCP client emulation, but it also brings clear improvements, such as the online support, TCP timestamps-based analysis, and means to localise network degradation. As a generic conclusion, due to the objectives of the research, robustness prevailed in front of accuracy within the analysis.

The last part of this chapter provided an overview of a software program that implements the described method. The overview included the block structure of the program and some of the variables involved in the analysis.

The following chapter will expand on the issues raised by the proposed analysis method in terms of limitations and possible sources of errors. The discussion will analyse the problems that may come up for each of the estimated variables, discussing various configurations where the measurement is likely to be less reliable, as well as particular sequences that may not be interpreted correctly by the proposed method.

Chapter 4. Issues related to the proposed TCP analysis

The monitoring method proposed in chapter 3 aims to represent a robust alternative to TCP analysis. It was intended to be generic rather than accurate, especially because it has to predict network conditions encountered by TCP endpoints with unknown implementations. It was observed in previous studies, [Paxson 1997b], [Floyd and Padhye 2001] that current or past TCP implementations do not follow current standards, such as [Braden 1989] or [Stevens 1997], for various reasons. Among the problems relating to TCP implementations, there are:

- 'Bugs', with the meaning of faulty behaviour which does not improve the performance under any reasonable conditions and was probably caused by an error that slipped in the source code (e.g. the Solaris bug revealed in [Paxson 1997a]);
- Tailoring for specific networks (e.g. Solaris delayed acknowledgment policy, [Paxson 1997a], which performs well for LAN/low-delay environments but brings the connection near to a 'stop-and-wait' behaviour for high RTTs);
- Missing algorithms (e.g. Windows 95/98/NT, which do not perform fast retransmit [Floyd and Padhye 2001]);
- Diversions from the standard (e.g. generic broken TCP retransmission, observed in [Floyd and Padhye 2001] which leads to a range of erroneous behaviours).

Besides these implementation issues, there are also issues related to uncertainty due to position of the monitor, that will be detailed in the subsequent sections. These two are the main reasons why some of the detailed directions of analysis, such as TCP algorithms inference, were not pursued.

Aside from the multiple sources of errors associated with each of the estimated parameters, this section presents various improvements to the previously described algorithms, improvements that should increase the overall accuracy when certain assumption are made in regards to the network behaviour. Therefore, it is aimed to be more than an enumeration of the cases in which the presented method fails to provide correct figures for the inferred parameters.

4.1 Round Trip Time

RTT inference accuracy depends on the number of estimations obtained (the more estimations produced, the higher the accuracy in assessing the actual value). The discussion will be based on a Server - Client configuration, as depicted in Figure 4.1, in which one of the stations (i.e. Server) mainly sends data, and the other station (i.e. Client) mainly sends acknowledgments to the received data. This model is considered because the majority of the traffic is produced in the current networks by client-server protocols, such as HTTP. The client connects to the server and requests files, then the server sends the files back to the client; in any of the cases, the client-to-server amount of data sent is very small (or non-existent) in comparison with the server-to-client data (the model also applies for peer-to-peer downloads).

As mentioned in the previous section, the monitor relies on the acknowledgment mechanism to determine RTT values. The main problem arising is that TCP senders, in order to avoid 'acknowledging acknowledgments', do not confirm the empty acknowledgments and, if the receiver does not send any data, there are no segments to confirm. As a result, in the worse case scenario, when the receiver does not send any data, the sender produces only two acknowledgments that can trigger RTT estimation: one in the synchronisation sequence and one in the closing sequence.

The end-to-end round trip delay, RTT_{total}, can be split, in relation with the monitoring station, into two components, as depicted in Figure 4.1:

- RTT_{west} – based on acknowledgments produced by the Server in response to data sent by the Client;
- RTT_{east} – determined from acknowledgments produced by the Client in response to data sent by the Server.

With the observations above, in a Client-Server configuration, the monitor can obtain a number of RTT estimations for RTT_{east} and only two measurements for RTT_{west}. The limited number of estimations itself does not lead to errors[1], but, in conjunction with other factors, can lead to major errors. The worst-case scenario happens when both of the packets that should trigger the RTT $_{west}$ measurement (the acknowledgment to SYN and the acknowledgment to FIN) are lost before the monitor (i.e. in the West Subnetwork). In this case, there is no apparent out-of-order in the TCP arrival sequence (simply because the actual data transmission either has not started or finished already), and the monitor erroneously interprets the second instantiation of the packet as good-for-RTT and the resulting estimate is in fact (RTT_{west} + retransmission timeout).

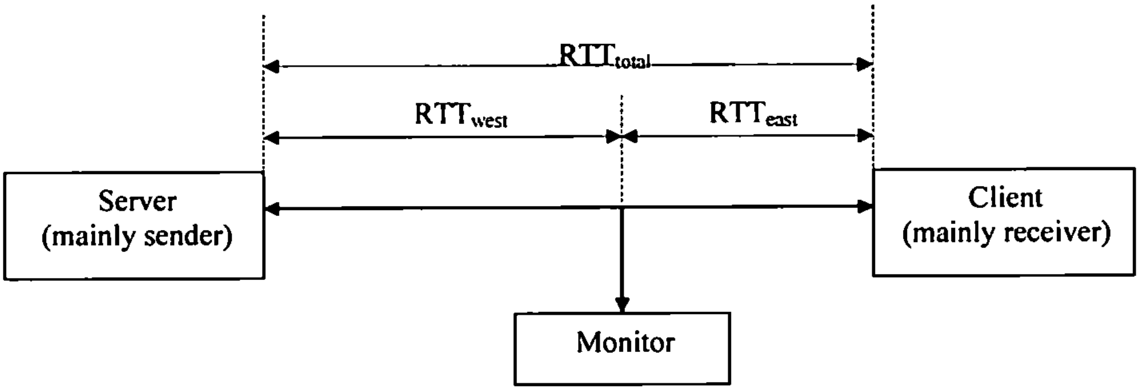


Figure 4.1 - Configuration example for RTT measurement

Uncertainty due to the position of the monitor, as described above, represents a technical limitation of the described monitoring method. The monitor, due to its position, cannot determine properly the pairs of sends and acknowledgments, in order to calculate the RTT. But, even in an ideal situation, when all the packets would provide 'pairing' information, is the resulting figure the actual delay introduced by the network? The answer is NO. In order to analyse the obtained figure, the whole route of the information must be considered:

Inferred delay = *network* delay + *processing* delay + *implementation-related* delay

The *network* delay refers to the time the packets spend in the network to reach from the source host to the destination host. It is, under 'normal' conditions, the term with the highest value in the equation, and therefore the inferred delay can be approximated with it. It has three components:

- *Propagation* delay, caused by the finite speed of the electromagnetic field (e.g. a bit of information travelling through a copper wire for 1000 km is delayed for 1000/(2.5*10^5) = 2.75 ms). It reaches high values for satellite connections;
- *Transmission* delay, due to the finite capacity of the path of links between the endpoints (e.g. a full Ethernet frame, 1514 bytes sent over a 10Mb LAN is delayed 10000/(1.514*8) = 1.21 ms). It introduces high figures for delay if the capacity of the bottleneck (the smallest capacity link from the path) is low, mainly in modem-connected endpoints;
- *Queuing* delay, due to congestion - n sources of packets that have to be forwarded by a router on the same segment have to queue and wait forwarding (e.g. 10 sources producing each back-to-back trains of 10 Ethernet full-sized 1.5 KB frames each,

[1] The higher number of estimators is required so that the TCP endpoint may follow more closely the changes in the network. Therefore, for a connection with a low or non-existent RTT variance, the number of delay estimators does not influence the evolution of the state machine associated with a TCP connection.

queuing at the router to be forwarded on a 10Mb link can introduce a delay of up to 100*1.21 = 121 ms). It represents a generic introduced delay, dependent on the traffic levels at the moment when the connection takes place. As there was no term to categorise the overall traffic level, Paxson defined [1999] the concepts of *busy* and *quiescent* network to define the overall volume of traffic, based on his observations that the Internet traffic follows 24 hour and 7 day patterns, due to the human activity.

The second term, *processing* delay, depends on the activity that takes place at the endpoint, mainly sender. If HTTP is considered, as it is the predominant type of traffic in current Internet (section 3.2.1), the web server has, depending on the task required by the client, to perform a certain number of actions. These actions can vary from a simply file retrieval from the hard disk (for a simple text-and-images page) to a complex query, followed by formatting of the resulting web page (for a search on a search engine). Each of these sets of actions requires a certain amount of time to be processed by the machine on which the web server runs. Although optimised for such operations, a large number of concurrent clients can produce a high processing delay, leading to additional time required by the web server to provide the responses. There is a clear difference between data acknowledging and page retrieval, as detailed in section 3.4.3:

- A server can, if configured, first acknowledge the GET request, then, at a later time, produce and send the requested page to the client; in this case, the RTT will be correctly estimated, and the monitor will be able to identify the server response time, as in 3.4.3. Otherwise, if the server does not produce any empty acknowledgments, the RTT estimation will be erroneous, as it will include the server response time within it;
- The server can be overloaded due to concurrent TCP connections. The delay resulting from this is transparent to the monitor in all the circumstances.

The third term, *implementation-related* delay, indicates the fact that the TCP senders themselves can introduce delay due to their operating rules. As specified in [Braden 1989], a TCP endpoint must produce delayed acknowledgments, i.e. an acknowledgment is produced either after not more than *x* ms (where *x* is e.g. 200 for Windows and Linux and 500 for Solaris) from the last data segment received or for every two full MSS segments, in order to lower the number of acknowledgments produced (for large data transfers, the number of produced acknowledgments is halved) and to avoid undesired behaviour such as the silly window syndrome [Clark 1982]. This does not affect the timing measurements for large data transfers, as the TCP sender has always data to transmit and the connection reaches full utilisation, but it can introduce an additional delay for small congestion windows values or, in general, for underutilised paths, when the window includes an odd number of packets. Unfortunately, this is not unusual for current picture of the internet, with average figures for from page web files of 10-20 KB, fact noticed in [Cardwell *et al* 2000] and analysed using a larger study in Chapter 6, for both experimental and backbone traces.

As presented, none of the terms has a definitive contribution to the overall figures, but they all depend on various factors, all of them transparent to the monitoring process, such as overall Internet activity and TCP implementation at the endpoints. All these suggest that the monitor provides a very crude estimate of the actual network performance, but informs about the sum of the network parameters AND the adjacent terms. The remark is correct; if all the 'worst case scenarios' are combined, the actual network contribution to the value of the measured delay can be very low. But, even in these conditions, the actual view from the TCP endpoint perspective must be considered. As with the monitor, the TCP endpoint obtains all its information from the arrival sequence of the data segments / achnowledgments, it does not know if a delay is due to network congestion or due to high load at the server, and it adjusts its parameters based on this seen sequence. Summarising, *the monitor, in extreme conditions, does not provide information about the actual network delay, but it offers an accurate image of the network as seen by the TCP endpoint.*

4.2 Packet loss

There are several categories of estimation errors relating to packet loss measurement that can occur in the network parameter estimation.

The first category of error sources are the ones that do not exist within current implementations, yet can appear in future ones. A good example of such a source is the Identification IP header field, used to determine apparent mis-ordering as mentioned in section 3.4.3. The TCP protocol specification includes an enhancement that can obliterate this algorithm: "TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment" [ARPA 1981b][1]. Although beneficial, this would require tighter cooperation between the transport and network protocols (in order for the first to inform the second about a segment being a retransmission or not) and the enhancement, or behaviour generated by it, was never discussed as a possible alternative in the literature.

While possible future error sources do not affect loss estimation for current TCP implementations, there is a range of errors produced by insufficient information at the monitor, due to its position in relation to the losses. As mentioned, there are two indications of a packet being lost: 'visible' retransmission and 'apparent' out-of-order. In the following paragraphs, a hypothetical connection evolution shown in Figure 4.2, will be discussed.

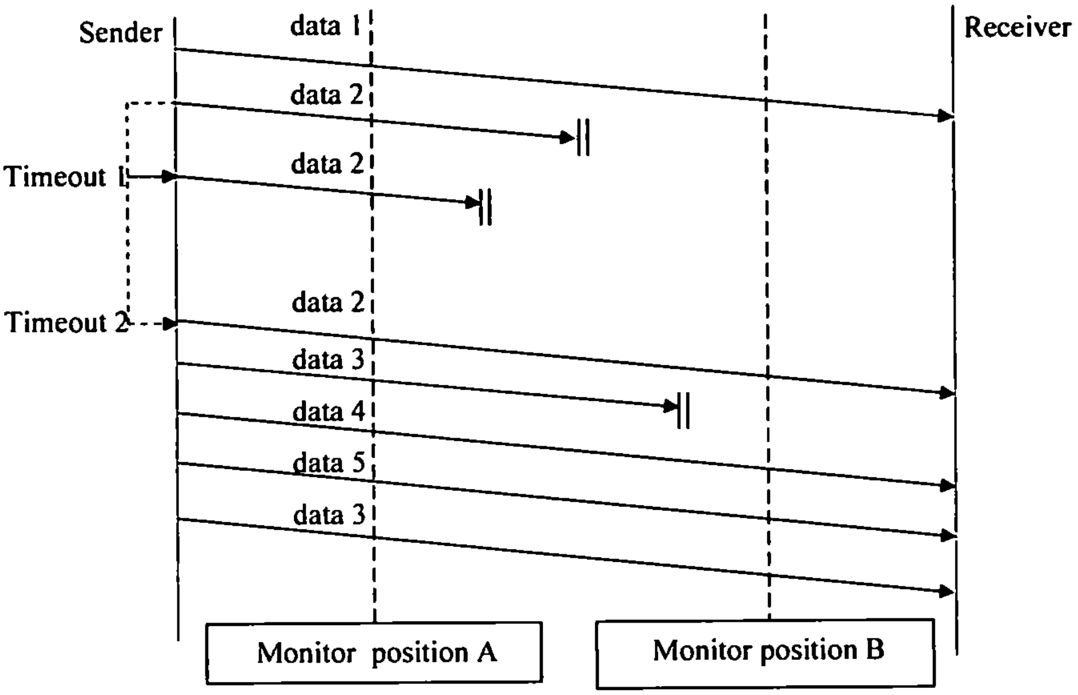


Figure 4.2 - Position of the monitor in relation to the packet loss

Note: There are several possible cases for retransmission, and they include acknowledgments from the other direction, but this is the most generic case if neglecting the acknowledgments.

Based on Figure 4.2, in order to detect all the losses, a monitor has to:

- capture both of the transmissions of the packet; (e.g. the first two transmissions of *data 2*, as captured in the monitor position A)

and

- identify an apparent out-of-order sequence of data segment arrivals as an actual packet retransmission; e.g. the *data 2, data 4, data 5, data 3* sequence, as captured in the monitor position B)

and

[1] Although no implementations appear to use it at the moment, this recommendation was repeated in [Braden 1989].

- identify a retransmission timeout by following the sender time-out procedure; e.g. (the *data1, data 2* – third retransmission - captured in the monitor position A).

Loss is successfully identified in most of the cases for the first two categories:

- the retransmission is identified by comparing packets containing 'past data' with the segments from the *SkipData* array
- the apparent out-of-order sequence is identified by comparing the Identification of a presumed out-of-order packet with the last Identification captured on this flow

The third category of losses requires, as mentioned, inference of the retransmission timeout. An endpoint starts / restarts a timer after releasing each packet on the network; if it does not receive any acknowledgment for the sent data until the timer expires, the sender considers the packet lost and retransmits it. In the following paragraphs, it will always be considered the case when the loss happens before the monitor, because, if the first instance of the data segment is visible, the monitor would be in the position A (with the conventions from Figure 4.2) and the case reduces to identifying repeated instances of same data segment in the packet arrival sequence.

It is important to understand under what conditions a timeout occurs, and in which of the cases it is undetectable using out-of-order identification. A TCP sender retransmits a data segment in two cases:

- If it receives more than a number of n (with $n = 3$, as recommended by the standard) duplicate acknowledgments for that segment (duplicate acknowledging);
- If it does not receive an acknowledgment for that segment in a time longer than the RTO (timeout).

There are three types of situations when a loss can occur, in relation to the status of the TCP connection at that moment in time:

- Beginning of a connection. When the connection begins, the congestion window is small, therefore it triggers a small number of achnowledgments, and the sender is more likely to retransmit due to timeout than due to duplicate acknowledging. For example, considering a 'correct implementation', a sender must receive 3 duplicate acknowledgments to a lost segment in order to retransmit it, as mentioned above. Therefore, for example, if the current window is 2 or 3 packets, there will not be enough double acknowledgments to trigger a retransmission, and the sender will have to timeout;
- Steady state (concept used in earlier studies such as [Padhye *et al* 1998]. In this stage, the TCP connections have a large(r) congestion window, due to the slow-start increase. The available bandwidth is 'filled' with data segments and achnowledgments. If a packet is lost, the event is flagged to the TCP sender by multiple double acknowledgments and it is unlikely in this case for a timeout to happen;
- End of a connection. The sender has no more data to send, and waits for the acknowledgment to the last data segment sent. Because there is no more data ahead, there are no double acknowledgments and the sender times out.

In relation to all the above cases, a loss will be transparent to the monitor if there are no more data segments between the two transmissions of the lost data segment. From the three situations presented, the second one is the only that creates no confusion: the monitor identifies the loss by observing the apparent mis-ordering in the data segments sequence. In the first case, it depends on the size of the congestion window whether or not the loss will lead to a timeout or to a transmission due to double acknowledging. In the other two cases, the sender will either timeout and stop transmitting any further data, as it had used the entire available congestion window, as in the first category, or simply has no more data to be sent, as in the third one; it is these two cases that require a solution to identify timeouts.

During the testing stages of the method, attempts were made to follow the above algorithm, but, besides problems interpreted as 'broken retransmission' in senders or the previously mentioned errors in estimating the RTT, there was still an essential unknown variable: the time when the first transmission occurred; without it, there is nothing to compare the RTO with. A slightly different approach was therefore considered, relating to time spacing between consecutive packets. It was mentioned that the monitor is unable to detect the timeout only if there is no data being sent between the two transmissions of the packet. In addition, as long as the sender has data to send, there should be no gaps longer than an RTT between transmitting two consecutive packets. However, this does not guarantee identification of all losses, due to TCP's backoff retransmission policy. Without having thorough knowledge about the sender, it was impossible to infer the value of the initial timeout and policy used at the sender. To exemplify from prior studies, the solution chosen by Cardwell in [2000] was rather coarse: hardcode the delay between packets and identify gaps longer than 1 second as retransmissions due to timeouts. Due to the complexity of the traces used, which may have included HTTP1.1 connections or, slow-responding senders, the inference mechanisms were slightly expanded. Based on the above assumptions, the monitor follows the gaps between consecutive packets and determines whether or not there could have been drops between two apparent normally transmitted packets. The analysis also ignored delayed first-packets in a connection, as they could be caused by a busy sender and it also eliminated gaps when the receiver had acknowledged all the data transmitted.

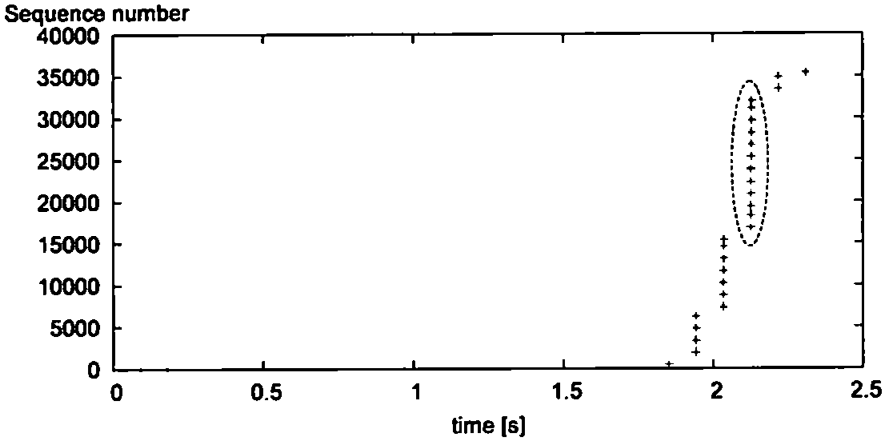
This method to determine retransmission timeouts should not be considered 100% reliable by any means, and this is why it is presented here, as an issue together with its possible solution, instead of being part of the previous section. There is, first of all, the obvious case in which there are simply delayed packets, without any timeout, in which a high variation in the network conditions is interpreted as a timeout. Another source of errors, with an even higher probability of happening, is usage of HTTP v1.1 protocol for web transfers which, as previously mentioned within the idle time analysis, section 3.4.3, uses *persistent connections*. While the idle time identification can flag inactivity periods that are longer than 2 seconds, the monitor will interpret shorter periods between object retrievals as timeouts. In fact, if the first data segment within a new retrieved object is lost before the monitor and timeouts, the event is completely transparent for the monitor, even if an additional module to decode and interpret the HTTP dialogue is implemented.

At the end, it is worth mentioning that sometimes even the TCP sender erroneously estimates the RTO, and does unnecessary retransmissions. These retransmissions can be observed by the monitor and flagged as *RTOError* packets, as mentioned in 3.4.3; the closer the monitor is to the receiver the more chances it has to determine such a condition (at the other extreme, if the monitor is right at the sender, it cannot see any such event, as the sender itself would adjust its values correctly). Unfortunately, the resulting figure for RTO errors can be higher than the real one due to, again, uncertainty generated by the position of the monitor. The monitor sees the acknowledgment that shows that the packet arrived at the receiver and was confirmed but there is no guarantee that also this acknowledgment arrived at the sender to inform about the successful transmission. In fact, due to the fact that acknowledgments are not confirmed, there is no way to determine whether or not this one arrived at the sender or not; the only solution, used by Paxson in his analysis [Paxson 1997b], would be to have two devices, one placed at the sender and one at the receiver, and to use them to determine *what* packets are lost or, if they arrive, *when* that happens.

The main problem faced during the Internet traces analysis phase, in relation to this timeout inference, was validation. In the 'visible' case of (possible) loss, the proof of packet retransmissions exists (the multiple occurrences of the same data segment or the apparent data mis-ordering) but, in this situation, even the retransmission (not the loss) is *inferred*, so there are no obvious means to check the validity of the assumption. The method used was to randomly select the connections which may have contained timeout inferences and determine, based on the timestamp numbers and on the (visual) evolution of the window whether or not the assumption was correct.

4.2.1 Limitations while identifying loss and misordering

It is true that, according to TCP specification [ARPA 1981b], retransmitted packets may have the same identification number as the previous instances, as the transport layers *may* control how the Identification field is incremented. However, the practical implementations do not reuse the identification numbers, but increment them for each IP datagram sent [Stevens 1995]. This is why a second instance of a packet with the same identification was considered to be due to duplication rather than loss-and-retransmission. An example of such behaviour is presented in Figure 4.3



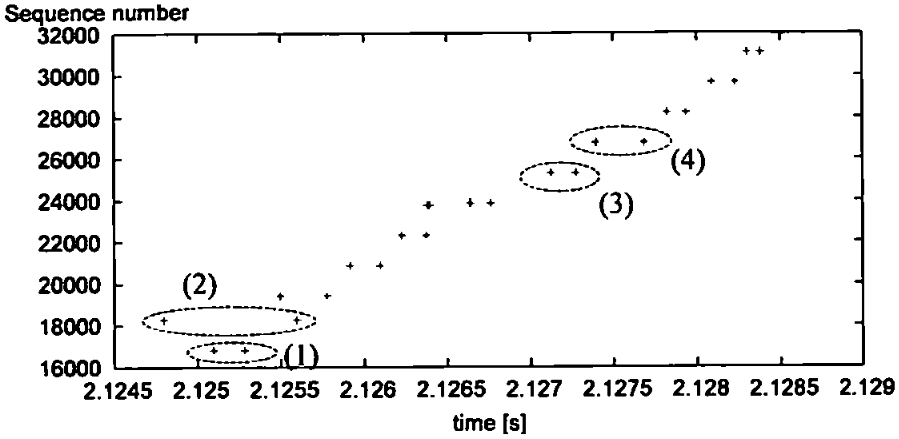


Figure 4.3 – Example of sender experiencing a mixture of packet duplication and misordering (top); a zoomed view of the circled area (bottom)

Judging by the spacing of the packets in Figure 4.3 in the top graph, it is clear that the circled packets are not part of a retransmitted window: the trains of packets are equally spaced and the congestion window is larger than the previous one. Also, the bottom graph shows the pairs of packets that appear to be retransmitted are closely spaced and in total there are 13 pairs of packets, figure that matches a slow start increase from the previous window (7 packets long).

The mechanism implemented to verify the ordering of the packets uses the *Identification* IP header field of the last captured packet. Because of this, it does work only for strong-coupled duplication, i.e. where duplicate packets are closely spaced. An example of such behaviour exists e.g. between the pairs of packets (3) and (4) from the connection presented in Figure 4.3, which the algorithm is able to correctly identify as duplicate. However, the pairs of packets (1) and (2) from the connection do not obey the previous rule and, as a result, the second packet from pair (2) is reported as retransmitted.

4.2.2 Avoidance of estimation errors due to Identification field wrap-around

In relation to the usage of the Identification field to identify mis-ordering, there is a potential wrap-around problem. The field is only 16 bit long, so it takes values in the interval 0-65535. Therefore, considering a web server, with three flows, *i*, *j*, and *k*, (between the server and one, two, or three different clients), the sequence shown in Figure 4.4 is possible:

```
t1 - flow i; ID = 66534; seqno n
...
t2 - flow j; ID = 65535
t3 - flow k; ID = 0;
...
t4 - flow i; ID = 1; seqno n+1
```

Figure 4.4 - Example of wrap-around sequence

It can be noticed that the datagram sent at t4, although in sequence, is interpreted by the monitor as an apparent out-of order, when it compares the IDs of the two datagrams. In the implementation, a light protection mechanism was introduced, see Figure 4.5, to avoid this happening, which works if the server sends less than 32768 (= 65536/2) datagrams between two consecutive segments which belong to a specific flow

```
if(Pkt.PID >= CurrentPID+1)
{
//in-order PID
     CurrentPID = Pkt.PID;
} else {
     if((CurrentPID - Pkt.PID) < 32768)
{
//inverted packet
          InvertedIP++;
     } else {
     //in order PID, wrap around
     CurrentPID = Pkt.PID;
}
}
```

Figure 4.5 - Procedure to avoid false out-of-order sequences caused by wrap-around

4.3 Congestion window

The estimation of the congestion window includes, by far, the largest amount of guesswork from the inferred parameters. The method used by Paxson in *tcpanaly*, i.e. follow the evolution of the congestion window based on the specification, is far from achievable in a single-point monitoring configuration, especially if the capture point is positioned 'somewhere' along the path of the packets, i.e. not even at one of the endpoints. In the single-point configuration, there is no guaranteed information about whether or not:

- a packet was transmitted – the sender might have transmitted it, but it was lost in the sender-to-monitor segment and the loss was transparent even to the timeout identification routine;
- an acknowledgment was received – the receiver sent it and it might have been lost in the monitor-to sender segment;
- a packet was received – the receiver might have received it, but the acknowledgment was lost in the receiver-to-monitor segment;
- an acknowledgment was sent – the receiver might have sent it, but it was lost in the receiver-to-monitor segment.

The first two transparent situations described above trigger a different, unknown, transition in the sender, leading to a particular evolution of the congestion window: each timeout switches the sender to slow start sending and each acknowledgment received updates the size of the congestion window. The last two cases, although not affecting the sender directly, can have an impact on the evolution of the congestion window: if the lost acknowledgment was flagging an apparent mis-ordering due to a previous loss, the sender will timeout instead of detecting double-acknowledging. In addition, the receiver is anonymous, therefore has an unknown acknowledgment policy. Under certain conditions, i.e. a low value of the delayed acknowledgment timer at the receiver, together with a low bandwidth x delay product, the receiver acknowledges every segment instead of the desired 'every-other' policy, situation highlighted in [Paxson 1997a] and [Floyd and Padhye 2001].

With all these unknown quantities in place, it becomes apparent why Paxson's method cannot be applied for single point observation. A novel approach, described in 3.4.3, was considered: identify the trains of packets, either based on the number of unacknowledged packets, or based on the spacing between packet trains. In the following paragraphs there will be highlighted the issues that each of these methods raises.

The second method to determine congestion window, described in 3.4.3 is based on counting, every time a packet is captured, the number of unacknowledged data segments at that moment. The accuracy of the method is affected mostly by the location of the monitor in relation with the sender and the receiver: the closer the monitor is to the sender, the higher is the accuracy of the estimated congestion window; at the other extreme, the congestion window estimate is inaccurate if the monitor is near the receiver. The reason behind this uncertainty can be discussed by comparing the packet exchange, as seen by the monitor, in the two extreme cases. If the monitor is near the sender, it will be able to identify correctly the 'peaks' in the variations of the number of unacknowledged segments, peaks that mark congestion window rounds, because the acknowledgments arrive only after the entire congestion window was transmitted. On the other hand, when the monitor is at / near the receiver, it will capture, at most, groups of two packets and their correspondent acknowledgment, inferring continuously a value of not more than 2 packets for the size of congestion window (which is due only to the delayed acknowledging of the receiver). The situation was exemplified in Figure 3.6.

The conclusion is that the position of the monitor in relation to the endpoints must be determined in order to enable or not the usage of this method. As all the near-far relationship mentioned before relate to the network delay between the respective hosts, the best way to determine these 'distances' is by comparing, either continuously or retrospectively, the RTT for the two sub-networks, East and West, see 3.4.6. If Subnetwork West RTT >> Subnetwork East RTT then the monitor is 'near' the sender, and the method can be used; otherwise, the monitor is 'near' the receiver, and the inferred value will be inaccurate.

The third method, identifying the packet trains based on the spacing between consecutive packets, is not affected by the position of the monitor in relation to the endpoints. This is due to the fact that the relation between time differences from equation **3.2** is not affected by the absolute network delays.

It is clear that the third proposed method will not work in the stationary case, e.g. for large file transfers, but it may succeed for smaller transfers. The following example will examine a connectivity example and try to determine the approximate size of a connection for which the method is still usable.

E.g. Ethernet LAN connected via E1 link to Internet:

- bottleneck bandwidth: BBW = 2 Mb/s = 256 KB/s
- delay: RTT = 250 ms
- maximum segment size: MSS = 1460 bytes/frame

For the above values, the resulting congestion window (CW_{max}) that would produce a contiguous stream of packets is:

$$\mathrm{CW}_{\max} = \mathrm{BBW} \cdot \mathrm{RTT} = 256 \cdot 0.25 = 51.2\ KB \cong 36 * MSS \qquad 4.1$$

The congestion window has to reach 36 MSS in order for the sender to transmit continuously. Because the inter-train time value used in 3.2 is RTT/2, the transmission window should be halved too. Even using the slow start algorithm, it will require 48 MSS (2+3+5+8+12+18, each of the figures representing the number of packets transmitted during a single round) to reach this limit. Based on the Ethernet MSS, the figure translates into approximately 70 KB. Above this size, the method would stop working for the given conditions; however, as shown in Chapter 6, typical bottleneck bandwidth figures are higher, leading as well to longer transfers. Also, this rationale applies in the ideal case where no loss occurs; if losses appear, the congestion window will be reduced, allowing the method to work even for larger data transfers.

Further, in the generic case of an unknown congestion window increase policy at the sender, the monitor determines continuously if the current (estimated) value is a valid one by comparing it with the possible maximum. The monitor can determine both the total RTT delay and the bottleneck bandwidth; therefore it can calculate the maximum detectable congestion window. If the currently estimated congestion window is larger than the maximum, this is probably due to the algorithm not being able to identify anymore between two consecutive congestion windows, and the value is discarded.

4.4 Bottleneck bandwidth

There are two types of issues in regards to bottleneck bandwidth estimation, from the error source point of view. The first category of errors relates to the unknown nature of the network, while the second is due to the TCP implementation at the endpoints.

4.4.1 Errors due to network conditions

Estimation of bottleneck bandwidth may be affected by a network condition called *queue draining*, [Zhang *et al* 1991], which may lead to *ACK compression* or/and *data compression.* Queue draining is caused by the way packets are being queued in the routers. In normal conditions, two packets sent with a certain delay Δt_1 between them, due to going through bottleneck links with a speeds lower than the bandwidth of their source point, arrive at the destination with a spacing $\Delta t_2 > \Delta t_1$. However, when a group of packets arrives at a router, its spacing will be affected by the size of the queue and the speed of the outgoing link. For example, two distanced data segments which are placed in successive slots in the queue will be forwarded back-to-back at the speed of the next link; as a result, their spacing due to the link they came from is cancelled. Depending on the position of the bottleneck links and draining queues, the inter-arrival time of the packets, Δt_2, will be smaller than the one created in normal conditions, leading even to the case when $\Delta t_2 < \Delta t_1$. In, [Mogul 1992], and [Bolot 1993], the authors observed the phenomenon of ACK compression, where the affected packets were the returning (empty) acknowledgments. On the other hand, *data compression* was observed in [Paxson 1999]. This time, the affected packets were not ACK but data packets, with the same consequence: due to queuing, under certain circumstances, the spacing between successive packets can be lost as the train of packets travels to destination.

The proposed method relies only on data packets to estimate the bottleneck bandwidth, because the packets need to be transmitted back to back from the sender As a result, data compression does affect the measurement. However, a prior study found this phenomenon to be "relatively rare and small in magnitude" [Paxson 1999]. Nevertheless, to reduce the impact of such a condition onto the accuracy of the estimation, the processing will involve outlier removal from the initial estimation and averaging.

4.4.2 Limitations and errors due to implementation issues

The second category of issues, implementation-related, is of higher concern. The bottleneck bandwidth estimation, based on the TCP timestamp option, depends highly on the TCP implementation of the endpoints. The main source of errors in this case is the resolution of the internal clock, defined as "a (virtual) clock that we call the 'timestamp clock' " with values "at least approximately proportional to real time, in order to measure actual RTT." [Jacobson *et al* 1992]. This definition allows the TCP implementation to choose any resolution for the timestamp clock, as long as it is "proportional to the real time". What is not mentioned in the specification, and makes room for differences in the implementations, is the minimum / recommended / maximum values for the timestamp clock resolution. The following presents two cases of such clocks that might, in fact, impede the accurate measurement of the RTT values, for a generic case RTT=400 ms.

- 100ms resolution. In this case, the *tsval* (the TCP timestamp echo value carried by a packet) will differentiate between packets belonging to different transmission rounds, but the packets belonging to the same flight of packets will carry all the same *tsval.* As a result, the other endpoint will be able to determine the average values for RTT, but, due to packets with the same tsval spanned over an interval of up to 100ms, these values will include an error factor and, further, the RTT variations will be virtually undetectable;
- 1s resolution. Applying the timestamp RTT estimation does more harm than good in this case; packets belonging to different rounds will carry the same *tsval*, and they will not be able to provide any reliable estimate of the actual values for the RTT average or variation.

If considering only the sender timestamp the estimate for the bottleneck bandwidth can be only as accurate as the resolution of the internal clock. For a certain resolution of the TCP timestamps, the maximum reliable bottleneck bandwidth estimator BW_{max} would be:

$$BW_{\max} = \max\left(\frac{pkt\_size}{resolution}\right) = \frac{\max(pkt\_size)}{\min(resolution)} \qquad 4.2$$

For an Ethernet environment (where the maximum frame is approximately 1500 bytes), and a resolution of 10ms, the resulting maximum for correctly measurable bottleneck is:

$$BW_{\max Ethernet} \cong \frac{1500 \bullet 8}{0.01} = 1200000\, b/s \cong 1\text{Mbps} \qquad 4.3$$

Using such an environment, anything above this speed, e.g. an E1 link (2Mb/s), would lead to an unreliable estimate. To overpass this limitation, an assumption was made that the sender is using delayed acknowledgments (current implementations of Microsoft and Linux to satisfy this assumption). If this is the case, the sender will transmit back-to-back a pair of data packets, every time a new acknowledgment is received. The distinctive mark of these pairs will be that they will carry the same *tsecr* (TCP timestamp echo reply) value. This allows for the inference mechanism to work as long as the combined sender and receiver TCP timestamps include a reasonable timestamp.

The original theory behind packet-pair measurement, [Keshav 1991], suggested that packets should be sent back-to-back at the sender, both to simplify the calculations, but, more important, to increase the probability of the packet pair to be queued at successive positions in the routers; it is aimed to maintain this requirement in the proposed analysis method, but there are two obstacles:

- The resolution of the TCP timestamp clocks at the endpoints has to be known; otherwise, no proper decision can be taken regarding the accuracy of the measurement;

- The only information about spacing of the sender is gathered through the TCP timestamps values. If their resolution is too low, there is no direct method to determine which of the packets were sent back-to-back and which belong to different back-to-back trains.

These two requirements still exist even in the case where the sender and receiver timestamps are combined. This is because, if both the sender and receiver resolution are too low, then the measurement might still fail.

The first problem relates to determining the timestamp resolution of a remote point. First, an implementation case study about the resolution on a few systems will be discussed, followed by a method to determine the resolution for a pair of endpoints, from a third point, i.e. the monitoring point. Several main sources of information were identified for the implementations used during the various experiments conducted within this project, i.e. Linux and Windows. Linux is an open source operating system, therefore its TCP/IP implementation is freely available. In addition, the actual source code was presented in several publications, such as the in-depth analysis made by Stevens in [Stevens 1995] for 4.4 BSD. Although there are slight implementation changes between the various versions of the Linux kernel, the principles, as well as the majority of the code explained, remained the same: in 4.4BSD, TCP resolution was set to 500 ms, while in newer versions of the Linux kernel, e.g. 2.2.x and 2.4.x, the variable was reduced to 10 ms.

At the other extreme is the Windows TCP/IP implementation: its code is not publicly documented (hermetic development is one of the reasons why, until recent versions, the Windows TCP clients had several bugs, highlighted particularly in [Floyd and Padhye 2001]).

The most comprehensive source of information regarding the Windows implementation of the TCP/IP stack was found in [Microsoft 2000]. Nevertheless, the document provides little information about the algorithms used to implement various TCP functionalities, and insists on listing information on how to tune their values.

None of the above mentioned sources includes an overview of various operating systems and their correspondent resolutions. The most detailed information regarding this subject was found in [McDaniel 2001] and [Securiteam 2001][1]. Therefore, amongst other information, the article gives a list of the timestamp resolutions for various operating systems, list summarised in Table 4.1

Operating system	Ticks / second	Resolution [ms]
4.4BSD, Irix, Solaris*	2	500
Linux 2.2+, Windows 2000	100	10
Cisco IOS	1000	1

*More recent versions of Solaris use a 100 ms resolution.

Table 4.1 – TCP timestamps resolution for various operating systems (compiled from [McDaniel 2001])

4.4.3 Evaluate TCP timestamps resolution

To overcome the variety of available sources, a method was proposed to empirically determine

[1] It is interesting to note that the subject of the two articles was not TCP performance, but IT security. The author noted in both sources that the usage of timestamp resolution may represent a security flaw. By observing the timestamp values and their resolution, an attacker can identify the operating system of a remote computer as belonging to a certain subset of alternatives.

the resolution of the Linux / Windows TCP options timestamps. The resolution may be obtained by comparing the timestamps of two transmitted packets (*pkt1* and *pkt2*) with the timestamps of their capture. As a result, the resolution of the TCP timestamp clock is determined using the formula in 4.4

$$\text{Resolution} = \frac{timestamp_{pkt2} - timestamp_{pkt1}}{tsval_{pkt2} - tsval_{pkt1}} \qquad 4.4$$

The method was tested on short file transfers (10KB) between Linux 2.2.14 and/or Windows 2000 hosts, using the first and the last packets in the connection as *pkt1* and *pkt2*. The calculation indicated a resolution of 10 ms for Linux and 100 ms for Windows, which matches the figures from Table 4.1.

The problem is that, in order for the analysis method to function online, the resolution must be identified *during* the connection, not *after* it. In the above formula, the resolution was determined after the connection, by reporting the difference of capture timestamps for the first and the last packets to the difference of *tsval* values for the same packets. The advantage in doing the difference between the last and the first packet is twofold: the resolution is averaged for the duration of the connection and the two packets are distant enough in order to neglect the differences caused by changes in the packet spacing due to bottleneck bandwidth. The following assumption is made:

The resolution of a timestamp clock does not vary in time during a connection[1]

[1] It is known that errors of clock timing may appear which lead to variations in the actual values produced. The assumption does not refer to these variations, which should be very small relatively to the TCP timestamp resolution, but to the actual value set in the TCP client to use with a connection.

The implication is that the timestamp resolution can be determined using equation 4.4 on two packets. The only recommendation is that the time elapsed between the two captures should be high enough to eliminate the errors due to packet spacing changes (e.g.: if the two packets are successive, the impact of bottleneck bandwidth on the result is very high). The best alternative to satisfy this condition, while performing a useful measurement for the evaluation of the connection, is to use the first two packets produced by an endpoint during that connection: a SYN packet and its correspondent acknowledge. The advantage of this particular choice is that, after transmitting the SYN, each of the endpoints must wait for the other end to acknowledge the packet before transmitting anything else. In this situation, the time elapsed between the two packets will be 1 RTT, enough to eliminate the impact of the bottleneck bandwidth.

To compensate for any errors that may appear, the apriori resolution, obtained at the beginning of the connection from the SYN packets, is compared at the end with the value obtained from the first and last packet of the connection. If the difference between the two figures is too high, the bandwidth estimation should be considered unreliable for that particular connection.

Based on all these observations, the bottleneck bandwidth can be predicted with a good accuracy using the TCP timestamp, allowing the measurement of bottlenecks of up to 100Mb/s, as will be detailed in the trace analysis in Chapter 6.

4.5 Fault localisation

As section 3.4.6 discussed, the results gained from the network analysis are twofold. First to identify and measure/infer the current transport parameters of a network transited by the monitored traffic, then to localise the extent of the possible degradations from the plurality of paths to a subset of the Internet space. The main problem in interpreting the monitoring results is represented by data summarising: the large number of independent flows must be converted into a limited number of classes, which would allow displaying them as a whole.

Furthermore, the network (Internet) may be split into two domains, such as the monitored network (inbound domain) and the rest of the internet (outbound domain). The monitor can work out the distinction easily, as, in the typical case, the local network has a defined range of addresses (e.g. University of Plymouth has allocated the subnet 141.163.0.0). The separation between the two domains is straightforward in this case: a single 'if' rule, which compares a given IP address with the specified range and determines to which of the domains it belongs.

Expanding the case for any link in the Internet is more cumbersome due to IP localisation issues. Considering, for example, the above.net infrastructure [MFN 2003], a core network provider that connects Europe and the United States, the traffic passing through its segments is only transit traffic. An IP address, in order to be mapped to a West/East configuration, should be localised geographically into one of the two continents, which would involve a lookup in the table of allocated sub-networks. Even in that case, a company could register an IP network in US and have an office in Europe, a fact that would confuse the matters even more. Concluding, this division, although not fully scalable, is satisfactory for monitoring in points that connect a defined network to the Internet.

At the other end, it can be argued that this type of mapping is too generic. The 'rest of the Internet' can be either just a hop away from the monitored link, leading to very low figures for loss and delay, or remote, as far as 20-30 hops away, which would lead to high values for the same parameters. But the purpose of this domain division is to provide a first-degree localisation of possible network condition degradation to the monitored network or the rest of the Internet.

As will be shown in later chapters, for the real network traffic collection, the collection host was connected either in a local network or hierarchically above it, in a local backbone segment. Because of this positioning, one of the logical sub-networks presented in Figure 3.1 encompassed only the local network. Implicitly, the localisation analysis identified the sources of loss and delay mostly in the East sub-network (i.e. the path between the collection point and the remote endpoints). However, the theory was successfully tested in Chapter 5 using a controlled environment.

4.6 Separation between HTTP v1.1 sessions and timeout losses

The loss timeout technique is reliable for a simple HTTP 1.0 retrieval, where the reply is a single object. Problems arise if HTTP 1.1 is used, due to the spacing introduced between retrievals of successive objects within the same connection. For such transfers, there are two choices: involve HTTP analysis or infer object boundaries. The first solution requires combining the TCP analysis with information provided from analysis of the application layer (HTTP). Full parsing and interpretation of the HTTP headers was considered to be beyond the scope of this project and was reserved for future work (see Chapter 9). The second choice, infer object boundaries, was the solution preferred for this project.

The solution used to avoid confusion between HTTP 1.1 sessions and timeout losses was based on the assumption that a receiver should acknowledge all data before the sender would start transmitting the next object. As a result, the sender should be in IDLE state between transmitting consecutive objects with longer pauses between them, such as the session pictured from Figure 3.3. This is why the monitor, besides doing all the comparisons with the RTT values, also checks whether the sender is in an IDLE state. If the sender is not in an IDLE state and a gap longer than $RTT_{average}+RTT_{variation}$ appears between two successive data packets, the delayed packet is considered a retransmission due to timeout. Otherwise, the monitor interprets the sequence as normal.

4.7 Summary

This chapter presented a list of issues associated with the TCP analysis. Most of the discussed topics emphasised limitations of the proposed method or possible sources of errors, either due to certain network events or due to uncertainty. The encountered problems were ordered according to the inferred variable.

The discussion started with the errors that may appear in the RTT and loss estimation, highlighting the components of the RTT value and the uncertainty sources when inferring the loss events. The argument focused then on the congestion window issues, insisting on the limitations of the proposed monitoring method when analysing long-lived connections. The last inferred variable to be discussed was bottleneck bandwidth; the section observed the two types of factors that may influence the accuracy when measuring this variable: network conditions and implementation specifics.

The last two sections examined how fault localisation may be applied to a real network connectivity case and how to separate between successive HTTP v1.1 objects and timeout losses, events that lead to similar sequences of packets.

Chapter 5 will use some conclusions drawn from this discussion in order to evaluate the accuracy of the method through validation tests.

Chapter 5. Validation of monitoring methods

5.1 Introduction

This chapter presents the tests run to validate the TCP analysis method proposed in Chapter 3. The tests consisted of three stages:

- Produce connections using controlled network conditions and endpoints with known characteristics. The validation required a reference that can be compared with the proposed method in order to determine the accuracy of the estimation
- Apply the method on the obtained connections.
- Compare the input (known) network and endpoint parameters with the ones resulting from the analysis.

The first step, producing connections with known parameters, required a controlled network environment. As briefly introduced by section 2.6, there are two alternatives that may be used for such requirements: simulation or emulation of network conditions. The first section of this chapter will provide a detailed overview of the implementation choices used for this study. The second step, applying the method on the resulting packet traces, will use the implementation described in section 3.5. The comparison between the reference and the inferred parameters will take into consideration the observations made in Chapter 4.

The validation of the TCP analysis method required network traces since its implementation phase, in order to provide validation data. The traffic had to be generated within an environment with controlled network parameters in order to allow comparison between the output of the implementation and the actual network conditions. In addition, a controlled environment could introduce network conditions/impairments over a wide range of values; in comparison, the available real network did not offer that much diversity, as will become apparent from the results presented in section 6.3. The two choices of controlled environments to satisfy the above requirements were network simulation (resulting in synthetic traces) and network emulation (resulting in real traces over a simulated environment). The following two sections will introduce these two types of environment, while the rest of the chapter will discuss the validation tests results.

5.2 Testbed data – the *NIST Net* network emulator

Two network emulators were identified as appropriate for this study: *NIST Net* [Carson 1997], a freely available product, and Shunra [Shunra 2003], of commercial origin. *NIST Net* was preferred because of the flexibility offered (the software, although under development at the time when the experiments were made, was continuously improved by the research community) and financial reasons (the price quotations for Shunra were around the figures of £5-20K). *NIST Net* is a software program that runs under Linux and emulates various network conditions (e.g. satellite delay, congestion, loss) by forwarding packets between the interfaces of a router. The program emulates all network impairments:

- packet loss, by dropping packets, either randomly or in a correlated manner, based on a loss rate and a correlation factor (*losscorr).*
- network delay, by deferring the forwarding of packets, using a delay distribution with specified mean and standard deviation values; the values of delay may be either uncorrelated or correlated, using a correlation factor (*delaycorr*).
- bandwidth, by limiting, on a per-second basis, the amount of data being forwarded between its network interfaces

In addition, the program also emulates DRD (Derivative Random Drop) router queuing policy [Gaynor 1996], through two defining parameters: *drdmin* and *drdmax* (the minimum and maximum thresholds for which a queue drops packets with a probability of p, $0.0<p<1.0$). However, for simplicity and consistency with current Internet conditions, the traditional DropTail policy was used for queues.

The built testbed consisted of 2 endpoints connected via 2 routers, as shown in Figure 5.1. All the machines (*Routers*, *Monitor*, and *Endpoints*) were running flavours of Linux (either Suse 6.4 or RedHat 7.2).

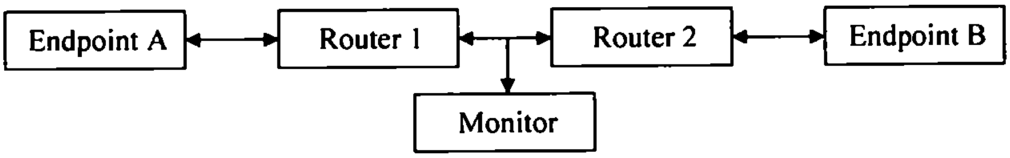


Figure 5.1 - The *NIST Net* testbed configuration

Both *Router 1* and *Router 2* were running *NIST Net*. The purpose of the testbed was to emulate a range of network conditions using the two routers running *NIST Net*, and to transport traffic between the two endpoints through this emulated network path. The rationale of using two emulation boxes was to ensure a higher complexity of the path (by combining the network parameters emulated in the two routers) and to offer the possibility of capturing packets *along the path* as opposite to *right at the endpoints*. Figure 5.2 presents the logical perspective for the above configuration.

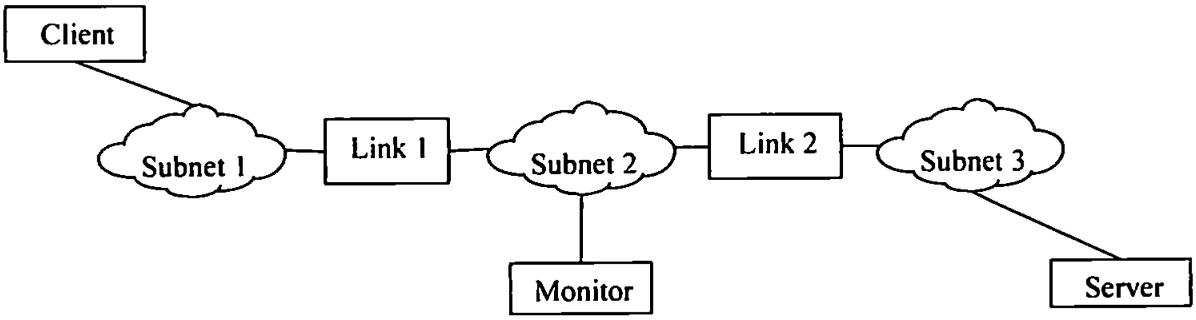


Figure 5.2 - Test configuration

As the monitoring station was placed in the middle, Figure 5.2 can be mapped onto the previously-mentioned East-West configuration from section 3.1.2, Figure 3.1. The West Subnetwork consists of (Link 1 + Subnet 1), and East Subnetwork is formed by (Link 2 + Subnet 3).

The traffic was generated using a command-line HTTP retrieval tool, ***wget*** [wget 2003] at one of the endpoints and an Apache web server at the other end. The tool was running on one of the endpoints and was requesting files placed on the other endpoint, which was set-up as a web server. The resulting traffic was collected at the Monitor and at the two endpoints using ***tcpdump***. The parallel packet capturing from the three points allowed full comparison between inferred network events (e.g. loss), and the actual conditions (e.g. determine whether or not a packet loss actually happened).

The generation / collection of ***NIST Net*** traces was script based and allowed full control of the process remotely, from the monitoring station. Due to the functionality of ***NIST Net***, independent network conditions could be defined for each direction at each router. As a result, four sets of parameters had to be specified for each experiment, each of them containing values for delay, jitter, delay correlation, loss, loss correlation, bandwidth, queue parameters, and the file size to be transmitted. These parameters ranged as follows for each segment:

- Delay: [10ms; 1500ms]
- Jitter: [0ms; 400ms]
- Loss: [0%; 20%]
- Bandwidth: [40000 b/s; 10 Mb/s]
- File length: 1KB, 10KB, 100KB

Initially, *NIST Net* proved to be a very good testing environment as it provided the project with *testbed data*, as defined in 2.6. However, the results from the tests highlighted several problems. The main issue was the size of the emulated environment: *NIST Net* required hardware endpoints for extension, endpoints that required either remote or manual control. In the testbed configuration used, i.e. with only two endpoints, the amount of traffic produced could not replicate a large network with a large number of aggregating connections. Of similar importance was the realism of the network conditions: for each incoming packet, *NIST Net* determined whether or not the packet should be dropped (according to the drop percentage and correlation), delayed (due to delay or/and bottleneck bandwidth emulation). This approach, although aiming to reproduce real network conditions, introduced errors when producing delay and bandwidth.

The delay emulation represented the highest source of errors. *NIST Net* calculated the delay for each packet separately; by treating packets independently, the program produced out-of-order packets when emulating variable delay. The simplest example of such behaviour was when two back-to-back or closely spaced packets arrived at the router and the first one was delayed more than the second one; in that case, the two packets would come out reordered from the router. This behaviour had no impact for the monitoring part (the method was able to account for reordered packets), but it was affecting the response of the TCP endpoint, which could interpret certain reordering events (i.e. a data segment being shifted by more than two slots) as losses and start congestion avoidance without an actual loss happening. This situation was, in fact, flagged by other users on the *NIST Net* mailing list [*NIST Net* 2003].

NIST Net also introduced errors when emulating bandwidth, because the bandwidth estimation was performed on a per-second basis. This produced non-realistic packet spacing, i.e. different from how it should be when packets pass through a slower link. This problem was less of an issue for the TCP endpoint behaviour, as the emulated figure for bandwidth was correct macroscopically, but could potentially lead to erroneous figures when trying to determine the bottleneck bandwidth using the TCP monitoring method.

5.3 Synthetic data – the NS environment

All the above experiments provided data from real transfers, encompassing transmission of real packets through either a controlled or uncontrolled environment. From this point of view, they were all superior when compared to a synthetic environment, which only simulates the transport of the packets, which makes them much more appropriate for training a trace-based model. Nevertheless, they all suffered to some degree from at least one of the two limitations: traffic aggregation and diversity of network environments, depending on the controlled or uncontrolled nature of the environment. Both of these limitations may be removed using a simulated environment, which allows reproduction of a virtually unlimited number of sources, connected via any combination of networks.

There are several choices for a network simulator (an extensive but not exhaustive list can be obtained from [Kennington 2003]), but the choice was rather simple, based on the prior work in the area of TCP protocol design and modelling; the majority of the authors working in this domain used the Berkeley Network Simulator (ns) suite to implement, test, and / or validate their concepts. NS is, as described by its authors, 'a discrete event simulator targeted at networking research' and provides 'substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks' [NS 2003]. Beyond this short description, *NS* is a collection of models that simulate the behaviour of various protocol entities, (named *agents*, e.g. TCP agent), network components (e.g. links and router queues) and specific network environments (e.g. wireless links, satellite links, etc). The user builds a script with specific *NS* commands to create a network topology and to introduce traffic through it using the available agents. The script is interpreted and run by the *NS* environment. The result is a trace of the simulated transfers, consisting of the events that happened during the simulation, e.g. queuing, de-queuing, and dropping. The *NS* suite includes several TCP agents that can be used to simulate the behaviour of TCP endpoints. From the existing choices (Reno, NewReno, Tahoe, etc) the *FullTCP* agent was preferred, because it was the nearest to a complete TCP client, including features such as TCP timestamp options, vital for the functionality of the trace analysis.

A problem was encountered when using the *NS* package to produce network traces: the output of *NS* was different from the pcap capturing structure. *NS* tracing support has a very simple design: the resulting output is oriented towards indicating the packets as they travel between endpoints rather than their headers. *NS* produces a plain text trace, with fields corresponding to packet fields (e.g. there is a field for the number of the packet, which, by multiplying with the size of the packet, gives the sequence number). This is satisfactory for the main purpose of ns, which is protocol efficiency or model accuracy validation, but improper for trace analysis. The only attempt to transform the output of *NS* into a usable form was a contributed module within tcptrace [Osterman 2003], but, at least at the time, it was both incomplete (it did not account for all the fields) and out of date (the format of the *NS* traces changed along the version). This module, was, nevertheless, used as a starting point to add to the traffic analysis tool the ability to use *NS* traces.

5.3.1 Topology

NS has the ability to simulate large topologies, but does not include the support for generating them. Although there are several programs which perform this task, it was preferred to integrate the topology generation with the simulation scripts for simplicity.

As a result, the *NS* traces were obtained by randomly generating network infrastructures. The simulation aimed to include a complex structure, with three levels of connectivity: a backbone running at a random speed (1), having two sub-networks connected to each of its ends through random speed links (2). Each sub-network had a random number of clients connected to it, all having the same access speed (3), but random delays. TCP connections were established at random start times over this infrastructure between clients from all four subnetworks. The trace collection gathered data from the backbone link and, for the TCP analysis, only the arrival events were kept for one of the backbone ends, in order to simulate packet capturing by an interface. Details about the program used to generate the simulation may be found in Appendix D. An example of such a structure may be seen in Figure 5.3.

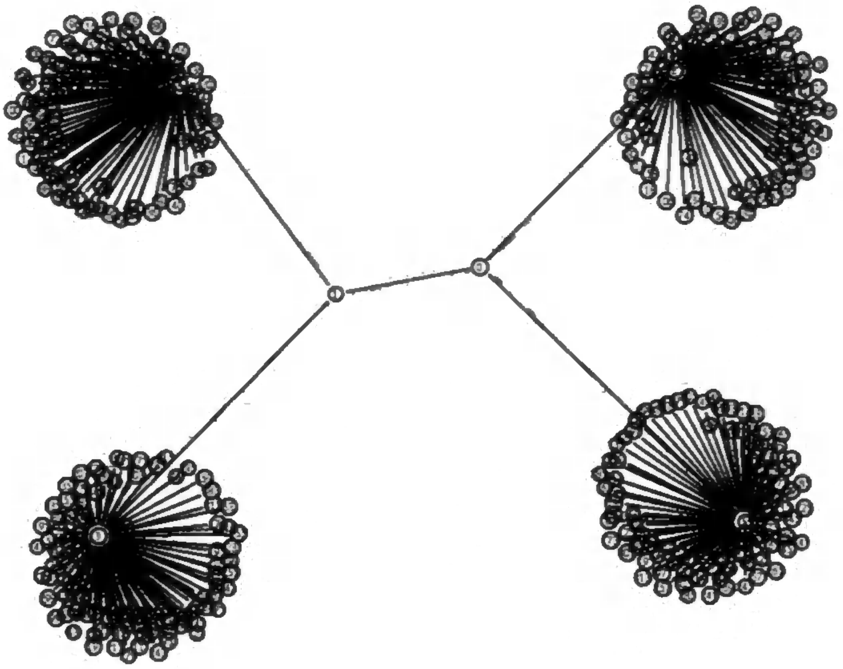


Figure 5.3 – NS generated network structure

The preferred solution to obtain more traces was to loop the script rather than keep the simulation running for longer. This was because, at the time, *NS* had no means to start more than 1 transfer on a TCP client throughout one simulation session (i.e. close the connection and open another one). The looping also offered the advantage that, although the generic connectivity remained the same, each run used different a different number of endpoints, connected via links with different characteristics, leading to a wider variety of environments and traffic conditions.

5.4 *NIST Net* tests

The ***NIST Net*** tests used the testbed described in section 5.2. The aim was to test the capabilities of the monitoring method using traces consisting of actual traffic rather than simulated one. The VoIP validation included a single round of tests, using a smaller version of the testbed, with only 1 ***NIST Net*** machine. The TCP tests were more comprehensive, with variations in the network conditions, in order to thoroughly verify whether the method infers correctly the TCP behaviour.

The combination of figures for file size, network delay and jitter, loss, and bottleneck bandwidth described in section 5.2 led to a number of 456 usable connections. Each test consisted of a single transfer between the two endpoints using a certain set of emulated network values (delay, jitter, loss, and bandwidth). The transfer was captured using *tcpdump*, then analysed with the method described in Chapter 3. The results from the set of traces produced were then filtered to keep only the useful data (some of the connections failed due to timeouts or finished abnormally and produced no results). At the end, the results of the inference were compared with the results of the measurement.

The jitter and bottleneck bandwidth estimations, initially part of the validation, were removed in the final stages, for different reasons. The jitter estimation was removed due to the measurement characteristics: any measured jitter metric would have been different from the metric introduced by *NIST Net*. For *NIST Net*, the jitter was applied to each packet and it depended on the set parameters (distribution and correlation) and the jitter applied to the previous packet. On the other hand, the monitor did not measure the jitter for every packet, but only for the *data segment - acknowledgment* pairs that satisfied the RTT measurement conditions. In addition, the measured jitter is obtained by combining the jitter from two sets of *NIST Net* emulations. For example, in the case of A acting as sender and B as receiver, the RTT measured on the route monitor - host B - monitor, will be affected by the link 2, direction AB (the data segments) and link 2, direction BA (the acknowledgments). Depending on how each of the two jitter instantaneous values combine, the resulting jitter may be anywhere between 0 and the double of the value set. The bottleneck bandwidth calculations were removed due to the errors introduced by *NIST Net*. The analysis of the resulting data showed only minimal correlation between the set bandwidth limitations and the estimated figures. To overcome this problem, connections were made over a controlled path that included a low-speed (ISDN) link, followed by analysis of the results.

5.4.1 RTT measurement

The RTT measurement section included, after filtering the experiments that did not yield any meaningful results, a number of 72 samples, using various figures for the delay. During this round of experiments there were no bandwidth impairments introduced via the ***NIST Net*** boxes. This was because, by introducing bandwidth impairments, it would have been impossible to obtain the real delay figures (produced by the combined action of added delay and bandwidth-limitation delay). Previous studies and tools [Paxson 1997b], [Ostermann, 2003] used measurement of round trip times based on TCP acknowledgments as a reliable estimator. However, as shown in Chapter 3, the TCP timestamps based measurement has the potential to be more accurate, as it allows a collection of more RTT samples, regardless of the position of the monitor. This is why, as well as determining the validity of the method for actual traffic (rather than synthetically generated one), the purpose of this test was to compare how accurate the two methods are when estimating the RTT. Figure 5.4 below shows the cumulative distribution for the two evaluation methods throughout the dataset.

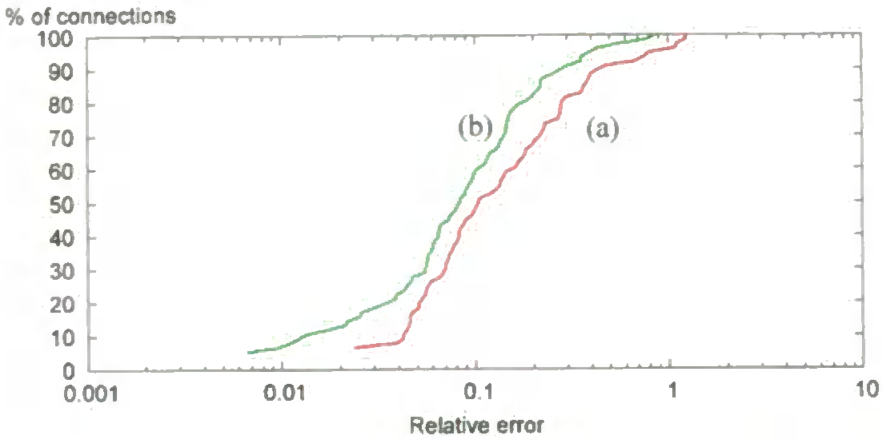


Figure 5.4 – Cumulative distribution of the (a) acknowledgment-based and (b) TCP-timestamp–based RTT estimation errors for the *NIST Net* delay experiments

It may be seen that the proposed TCP timestamps inference method outperforms the traditional acknowledgment-based method throughout the dataset. Aside from the visual representation, the two methods were also compared using statistical analysis of their results. Paired testing [Cochran and Snedecor 1980] was used to determine whether the timestamp-based method leads to lower relative errors when measuring the average RTT for a connection. A hypothesis was made that the acknowledge-based measurements lead to higher relative error values when estimating the real values of the RTT, compared with the timestamp-based measurements. The test involved calculation of:

- An estimate $s_{\overline{D}}$ for the standard deviation $\sigma_{\overline{D}}$ of the sample mean difference $\overline{D}$

$$s_{\overline{D}} = \frac{s_D}{\sqrt{n}} = \frac{1}{\sqrt{n}}\sqrt{\frac{\sum\left(D_i - \overline{D}\right)^2}{n-1}} = \sqrt{\frac{\sum D_i^2 - \frac{\left(\sum D_i\right)^2}{n}}{n(n-1)}} \qquad 5.1$$

  where $D_i = \overline{relerr\_ack_i} - \overline{relerr\_tsopt_i}$, with:

  - ○ $\overline{relerr\_ack_i}$ - the relative error between the actual average RTT for connection *i* and the average acknowledgment-based RTT estimate for connection *i*
  - ○ $\overline{relerr\_tsopt_i}$ - the relative error between the actual average RTT for connection

i and the average TCP timestamps RTT estimate for connection *i*.

- The *t* quantity from Student's t distribution:

$$t = \frac{\overline{D}}{s_D} \qquad 5.2$$

The resulting value for t was then compared with values from lookup tables (the above mentioned [Cochran and Snedecor 1980] includes such tables in the appendices section) in order to evaluate the statistical significance of the hypothesis.

Based on the data from Figure 5.4, the resulting values were: $\overline{D} = 0.0825434$, $s_{\overline{D}} = 0.0315989$, and $t = \frac{\overline{D}}{s_D} = 2.61222$. Using the lookup table, this value is situated between the 2.5% probability, $P_{2.5\%} = 2.29$, and the 1% probability, $P_{1\%} = 2.648$, for a measurement with 70 degrees of freedom. It may be therefore concluded that acknowledgment-based RTT measurements lead to relative errors 0.08 higher than the TCP timestamps-based RTT measurements with a statistical confidence of more than 97.5%.

To further clarify the results, it was investigated whether the high error values may have been produced by other factors rather than errors in the inference mechanism. Examples of such factors may be additional delays, such as processing delays, which become significant for low values of the introduced delays. To determine whether this is the case, a graph was produced to observe how the inference relative errors varied as a function of the introduced delays.

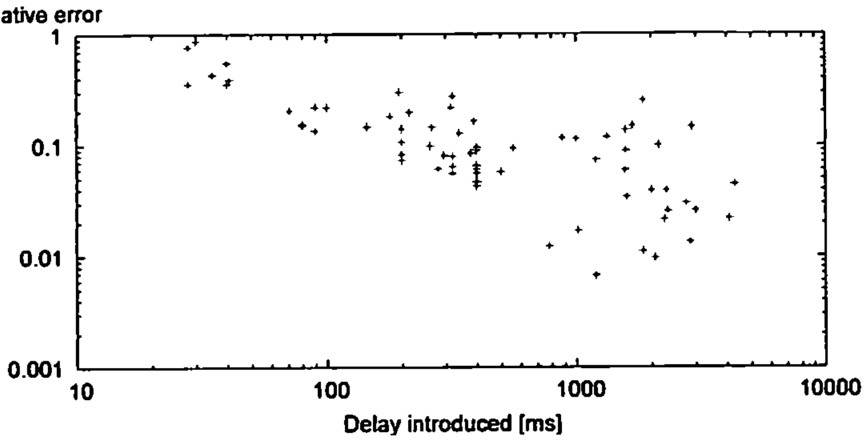


Figure 5.5 - Plot of relative errors of TCP timestamp based inference as a function of delay introduced by *NIST Net*

Figure 5.5 shows the relationship between the introduced delay and the relative errors when using TCP timestamps-based inference. It may be seen that higher error values appeared only for low delays, where it is likely that the influence of other delay sources was relevant for the result. Overall, for the entire dataset, the traditional, acknowledgment-based inference had an average relative error of 23.1%, while the TCP timestamp based method produced errors of only 13.8%. One of the causes for this difference in accuracy is the advantage that TCP timestamp inference brings in terms of number of samples, as highlighted in Chapter 3. To illustrate the difference between the number of samples, Figure 5.6 below shows the distribution of the ratio between the number of samples obtained with each of the two methods for the entire set of experiments (the whole set was used for these statistics because the actual RTT values were not relevant, but the number of RTT samples obtained for each connection).

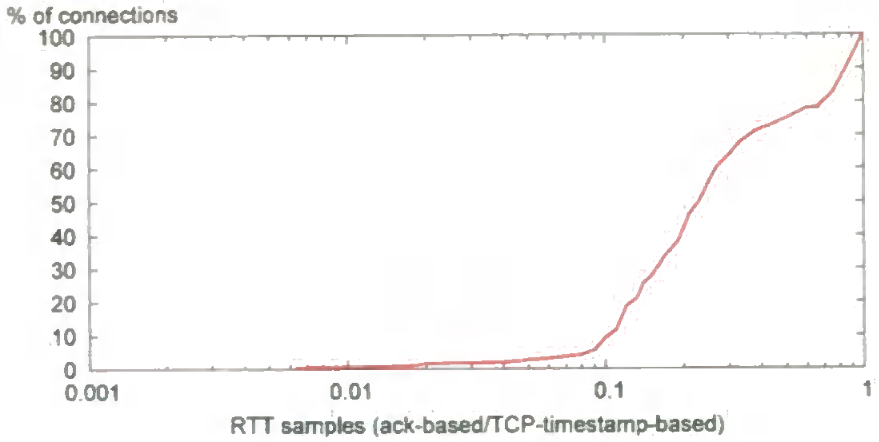


Figure 5.6 - Cumulative distribution of the *acknowledgment-based RTT samples / TCP-timestamps-based RTT samples* ratio for the *NIST Net* dataset.

The graph shows that the TCP-timestamp-based method always yielded more samples than the traditional method, with an average of 0.38 for the ratio between the two. It is true that the improvement is better for longer transfers, which, as will be seen in the trace analysis section, is not always the case for real traffic. However, regardless of the size of the file, the timestamp-based method will provide more measurement samples and better estimates of the actual delay values.

5.4.2 Loss measurement

A subset of 326 connections (from the 456 generated), included packet loss varying between 1% and 62%. The loss was distributed amongst the 4 segments either symmetrically (same for all four segments – 2 routers x 2 directions) or asymmetrically (e.g. introducing more/less delay on the forward/return direction). Aside from loss, various amounts of delay and delay variation were also introduced to simulate a realistic environment. ***NIST Net*** maintained the number of lost packets, a facility which was very useful in the analysis as it allowed comparison between the reported number of discarded packets (by ***NIST Net***), and the estimated number of lost packets, as determined by the TCP analysis. The results of the TCP inference were compared on a per-segment basis with the *NIST Net* data in order to be able to isolate degradations in terms of packet loss, equivalent with an East-West divide, as explained in Chapter 3. The results are presented below in Figure 5.7.

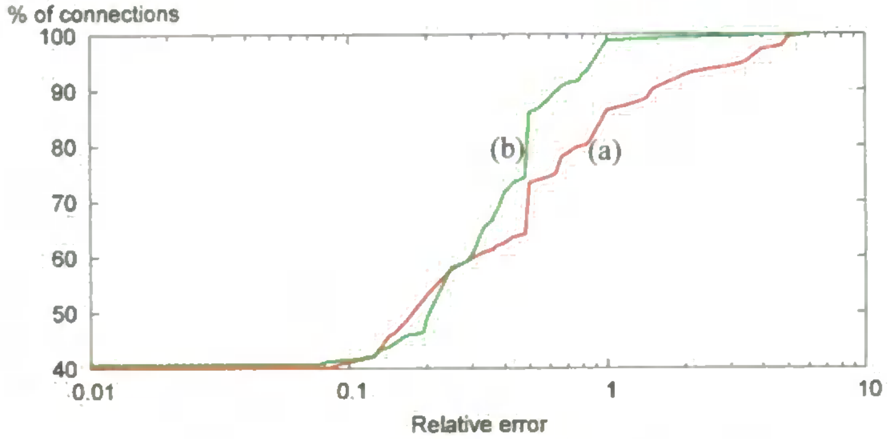


Figure 5.7 - Relative error between the inferred and reported losses for (a) *LostBefore* and (b) *LostAfter* loss events.

Figure 5.7 indicates similar shapes for the distribution of the two measured parameters. Statistical analysis was employed to determine which of the two types of loss is more accurately identified. The method was the same to the one used for Figure 5.4: use paired testing to determine which variable has higher values. The hypothesis was, based on the shape of the two distributions, that LostBefore events lead to higher relative measurement errors in comparison to LostAfter events. The resulting average difference between the two resulting type of errors was $\overline{D} = \overline{(LostBefore\_err_i - LostAfter\_err_i)} = 0.299314$. The calculation returned $s_{\overline{D}} = 0.0890647$ and $t = 3.36064$. Using the lookup table, this value is situated beyond the 0.1% probability $P_{0.1\%} = 3.2905$. Based on this, it was concluded that LostBefore measurements can lead to higher relative errors than LostAfter events with a statistical confidence of more than 99.9%.

The graph does indicate that, although for 40%-50% of the connections the method appeared to be accurate, errors appeared throughout the rest of the samples. In fact, 20% of connections appeared to have more than 100% errors in the estimation. It is true that some of these errors may have been genuine mis-interpretation of the TCP monitor. However the basic support that *NIST Net* provided for reporting loss, cumulative number of losses, did not allow any further analysis into the sources of the errors. To determine the problems, some experiments were run with data collected at the sender. Based on manual analysis of these traces, it was concluded that the errors may have been the result of other events that generated packet loss unaccounted for by *NIST Net*:

- events leading to higher estimates than the reported figures: packets dropped by the routers, losses on the reverse path (dropped acknowledgments, forcing the sender to timeout and retransmit), erroneous retransmissions (an example of such retransmissions will be presented in section 5.5.2)
- events leading to lower estimates than the reported figures: multiple drops of the same packet, drops of non-data packets.

Due to these multiple uncertainty sources it was decided to expand on the accuracy of the loss inference in the *NS* tests, which offered a better controlled environment, with fewer sources of error and comprehensive reporting functionality.

5.4.3 Bottleneck bandwidth measurement

As mentioned in the introduction section, the bottleneck bandwidth experiments failed to show any relevant results. To overcome the lack of results, a round of 50 experiments was run within the Acterna company network over a network path that included a 64Kb/s ISDN link, as shown in Figure 5.8, in order to determine the efficiency of the bottleneck bandwidth estimation.

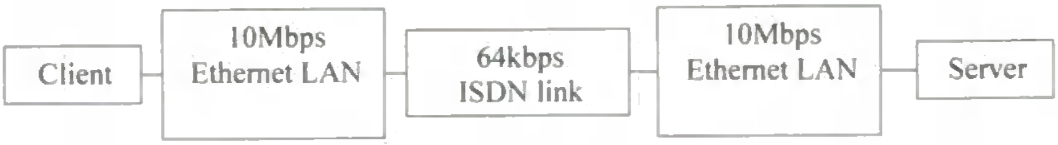


Figure 5.8 – Block diagram of the bottleneck bandwidth testbed

Each experiment consisted of sending a file between the two endpoints, one acting as a server and one acting as a receiver, and capturing the resulting packets at the client side. During the experiments the size of the file being transferred was varied (10KB and 100KB) in order to observe the efficiency of the method when transferring small/large objects. The results of the measurements are presented below.

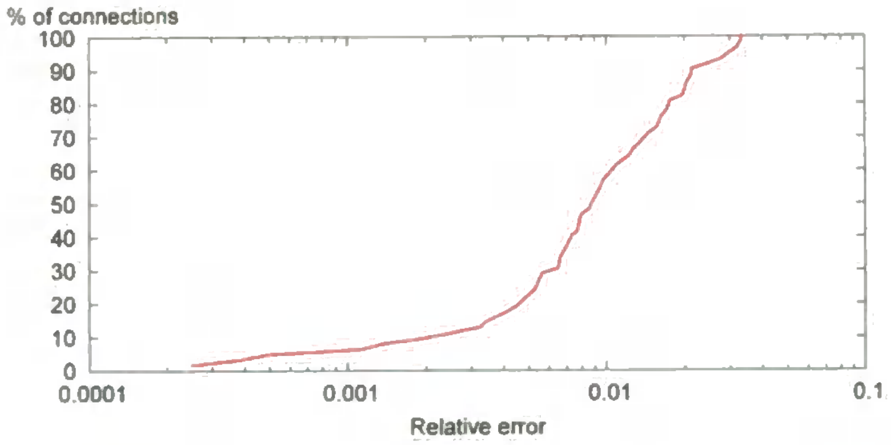


Figure 5.9 – Cumulative distribution of relative error for the bottleneck bandwidth estimation using a 64kb/s bottleneck

The resulting distribution from Figure 5.9 shows that the bottleneck was accurately identified throughout the batch. The accuracy was reflected also in the average figure – the average relative error was 1.14%. The number of samples was also relatively high. The traces produced 41 samples/transfer for each of the 100KB files and 4 samples/transfer for the 10KB files allowing observation of any possible variations in the average per-connection figures.

5.4.4 Conclusions

The ***NIST Net*** experiments offered a comprehensive image for the evaluation of the RTP monitor. The analysis indicated that the monitor was able to identify the location of the degradation in relation to the monitoring point based on the combined RTP-RTCP analysis. Also, both measured parameters (packet loss and jitter) were synonymous with the introduced degradation.

The results were less encouraging when evaluating the TCP connections using the TCP monitor, mainly due to the various sources of error that ***NIST Net*** introduced. The delay experiments encountered variations in the RTT estimation error, but indicated that the TCP-timestamp-based estimation produced a more accurate estimate of the RTT. This was due to the higher number of estimation samples for each connection. The errors recorded for some of the experiments appeared to increase for smaller values of the introduced delay. It was concluded that they may have been caused by factors relating the actual network used for testing. The loss experiments were more affected, with high errors between the estimated losses and the actual figures recorded by ***NIST Net***. A list of likely causes behind these errors was presented, all indicating that synthetic traces may prove to be a better alternative for loss evaluation. Finally, as described in Chapter 5, the bandwidth estimation did not produce any reliable results due to the microscopic behaviour of ***NIST Net***. However, a batch of connections was used as an alternative means of testing and indicated that the bandwidth estimation was reliable, generating a relatively high number of samples even for small file transfers and producing accurate estimates throughout the batch of connections.

5.5 NS tests

5.5.1 RTT tests

The structure from Figure 5.10 was used to generate 100 different batches of connections. The network reproduced a 3-tier structure: access (links 4-2 and 5-3), aggregation (links 2-0 and 3-1), and core (link 0-1). The links were set with the following parameters:

- Access / aggregation / core bandwidth – variable 1-10 / 10-100 / 10-100Mb
- Access / aggregation / core delay – variable 0.01-0.05 / 0.01-0.05 / 0.05-0.1 s
- Gateway / Core router queue limit – 10 slots

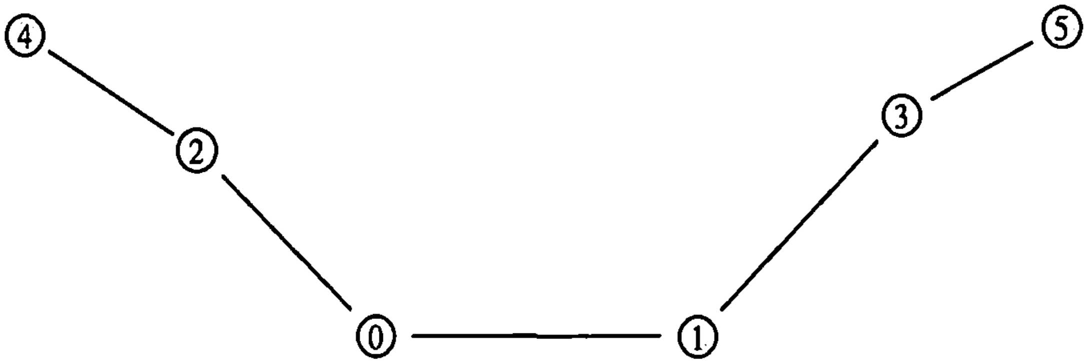


Figure 5.10 - NS configuration used for RTT tests – path comprising three types of links: access (4-2 and 3-5), aggregation (2-0 and 1-3), and core (0-1).

The word 'variable' indicates that the parameters were generated randomly with a uniform distribution for each experiment, using the internal *NS* random number generator. The two figures for each variable indicate the boundaries between which the respective variable was generated. The bandwidth was set to relatively high figures (Mb/s) in order to reduce the impact of queuing delay (relative to the propagation delay) on the results (this is why the queue limit for the gateway was also set to a low value). The purpose of this setup was to equal the total RTT with the sum of the simulated link delays, which was easier to compute from the script; this would allow comparison between the estimated total RTT and the sum of simulated link delays.

After generating the traces, they were filtered to keep the trace as captured in a single node and then analysed using the proposed method. The estimated delay and bandwidth figures were compared with the set link delays and the bottleneck bandwidth. The results of the analysis are shown below in Figure 5.11.

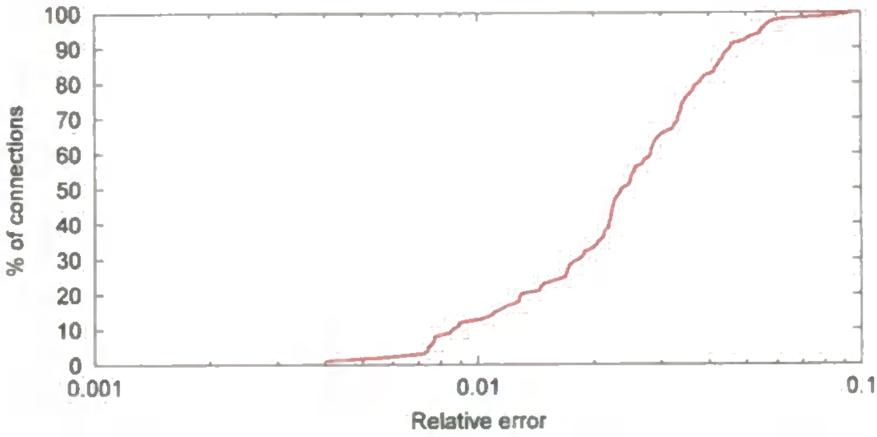


Figure 5.11 – Cumulative distribution of the relative error between the sum of links delay and the estimated RTT

The distribution of the resulting RTT relative error shows that the method estimates the delay accurately. The resulting average relative error was only 2.61%. It is worth noting that the errors were probably even lower due to ignoring the queuing delay when computing the real delay values. Support for this, not shown in the above distribution, is that all the RTT estimates were higher than the actual RTT values.

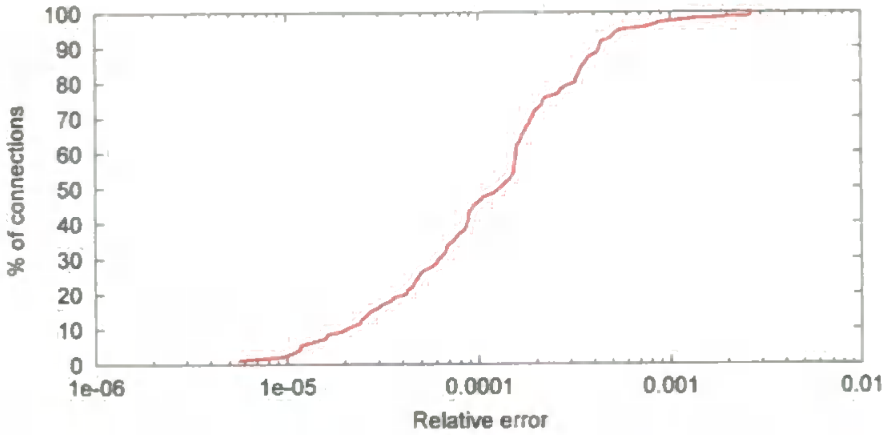


Figure 5.12 - Cumulative distribution of the relative error between the actual bottleneck bandwidth and the estimated bottleneck bandwidth

The bandwidth estimation also appeared to perform very well, as pictured in Figure 5.12. The method estimated correctly the bottleneck bandwidth throughout the batch, with an average relative error of only 0.02%. It must be said that this virtually perfect match may have been also due to the fact that the trace in question is a simulation; therefore all the timings are ideal and not affected by any external factors. Also, the TCP options timestamp mechanism, as implemented in the TCPFull agent, uses the simulation clock, therefore it has a 1µs resolution.

5.5.2 Loss tests

A network topology likely to lead to congestion, shown in Figure 5.13, was used to produce the traces for loss testing. The end networks, running at 10Mb/s, having each 10 nodes, were connected via two routers with 10-slot queues to a 2Mb/s backbone link. The endpoint nodes connected to node 0 were set as the senders and the nodes connected to node 1 were the receivers. The simulation lasted for a random time between [10s;20s] and had connections starting at random moments and running for random periods between [5s,15s]. The queues of the two backbone nodes (node 0 and node 1) were set to 20 slots.

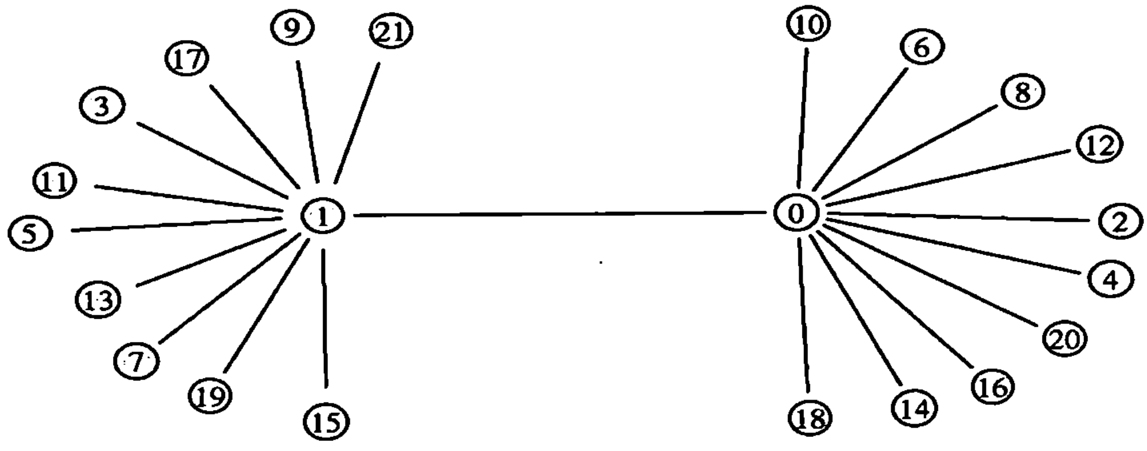


Figure 5.13 - Network topology used for losses – two 10-host 10MB/s networks connected via a 2Mb/s backbone link (between nodes 0 and 1)

The above topology was used to run several simulations, leading to a total number of 100 connections collected. The traces were collected from the receiver side of the backbone (node 1). Due to the logging capabilities of *NS*, it was possible to include in the trace the *packet drop* events. The results from the TCP analysis of the *NS* traces were then compared with the number of dropped packets, as reported by *NS*. The results are shown below in Figure 5.14

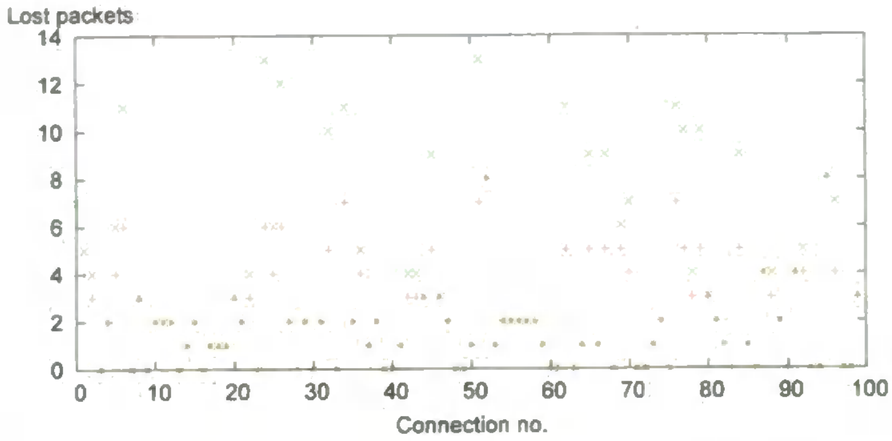


Figure 5.14 – Plot of the lost data packets for each connection (x) as resulting from the TCP analysis and (+) as reported by the *NS* trace

It may be noticed that the method estimates correctly low values, but it appears to overestimate for higher loss rates. The over-estimation cases are in fact due to avoidable retransmissions. An example of such a connection is presented in Figure 5.15; the graph uses data from Figure 5.14 (connection no. 25), but collected right from the receiver rather than from node 1. The chosen case is one of the worst in terms of discrepancy between the estimated figures for lost packets vs. the number of dropped packets, as reported by *NS*.

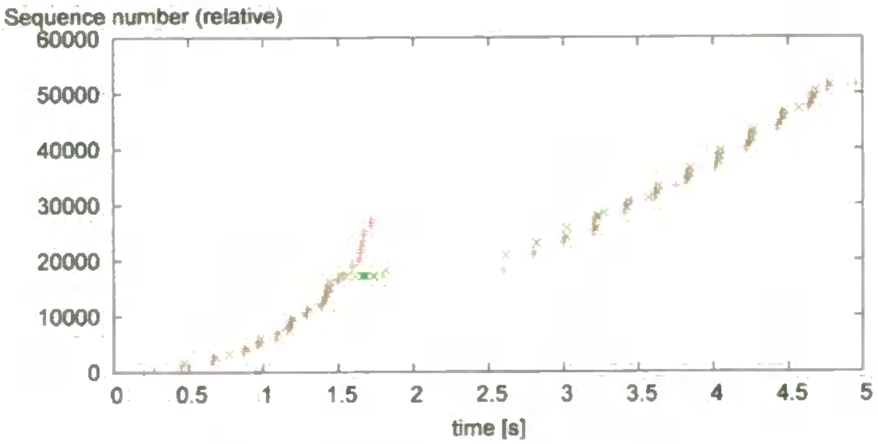


Figure 5.15 – Plot of (+) sequence numbers from sender, (x) acknowledgments numbers from receiver within a connection exhibiting erroneous retransmissions

Out of the last 20 segments transmitted in order after 1.6s, only 6 of them were dropped. The 6 dropped packets are the ones reported by *NS* (1st, 4th, 7th, 12th, 14th, 17th), while some of the rest, although not lost, were incorrectly retransmitted by the sender due to receiver not returning any acknowledgments. It is worth noting that the 7 erroneous retransmissions were avoidable, but they were caused by to the position of the lost packets rather than faults in the retransmission timeout calculation. For example, the 2-segment window arrived at 2.8s, included 1 dropped segment (the 7th from the original window), plus one erroneous retransmission (the 8th segment, not lost during the first transmission).

Summarising, the method provides an estimate of the *loss rate as perceived by the receiver* rather than the *actual loss rate*, as existing in the network. However, as visible in Figure 5.14, the two figures coincide for small transfers and reduced number of losses.

5.5.3 Congestion window

There is no monitoring mechanism within *NS* that allows monitoring of the congestion window; the trace support outputs only header fields, not agent status. However, the TCP client used for the simulations (TCPFull) did include an internal variable *cwnd_* that indicates the size of the current congestion window. In order to observe the evolution in time, the source code was modified to print in a separate trace the values of the *cwnd_* variable. Unlike RTT, bandwidth, and loss values, which require an average figure, congestion window estimation should reflect the evolution of the variable in time. This is why the best choice to compare the inferred values with the actual figures was considered to be graphical comparison. To illustrate the comparison, a connection example was used below in Figure 5.16

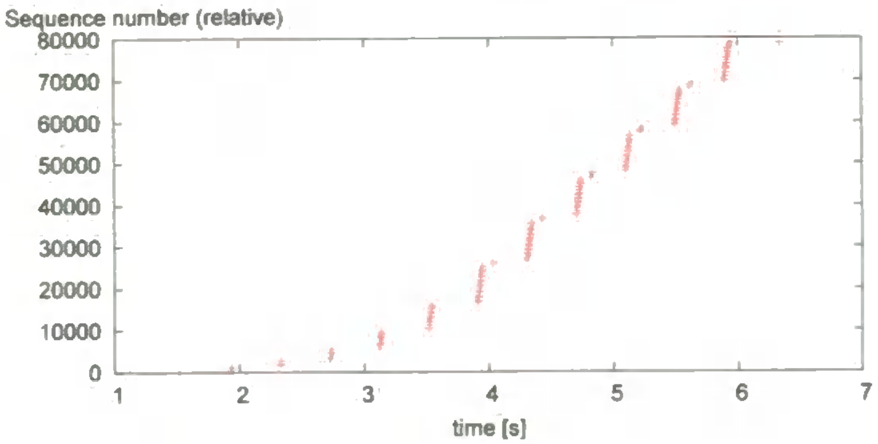


Figure 5.16 – Plot of the relative sequence numbers for a connection with congestion window limited by the receiver

The connection evolved in slow start, then, from 4s onwards, the congestion window was limited by flow control mechanisms. The result of the comparison between the internal congestion window value and the inferred values is presented below in Figure 5.17

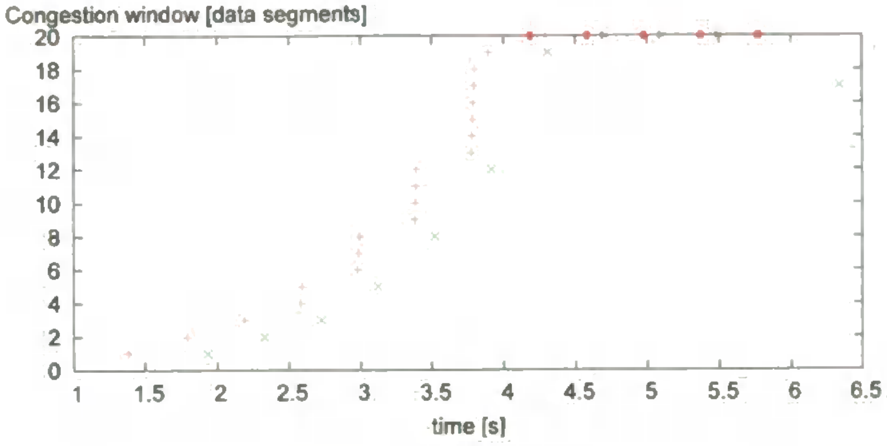


Figure 5.17 – Plot of congestion window values, as resulting from (+) *NS* internal variable cwnd_ and (x) TCP analysis inference (based on the connection from Figure 5.16)

The graph indicates that there is a phase shift of approximately 1 RTT between the real values and the corresponding inferred ones. For example, the first congestion window (1 segment), appears at 1.4 seconds; however, the corresponding inferred value appears 0.5 seconds later. The delay is maintained throughout the connection. This delay is due to the way the method works: a certain congestion window is inferred only when the first packet from the next round arrives, which is 1 RTT later. Considering this note, it may be seen that the inference follows the actual value throughout the connection. The only apparent error is the last value, which was recorded as 17 instead of 20. This was because the last round of packets did not include 20 data segments, but only 17.

The method does work even in the case of loss, but with a few limitations. An example of a connection that exhibited losses is given in Figure 5.18.

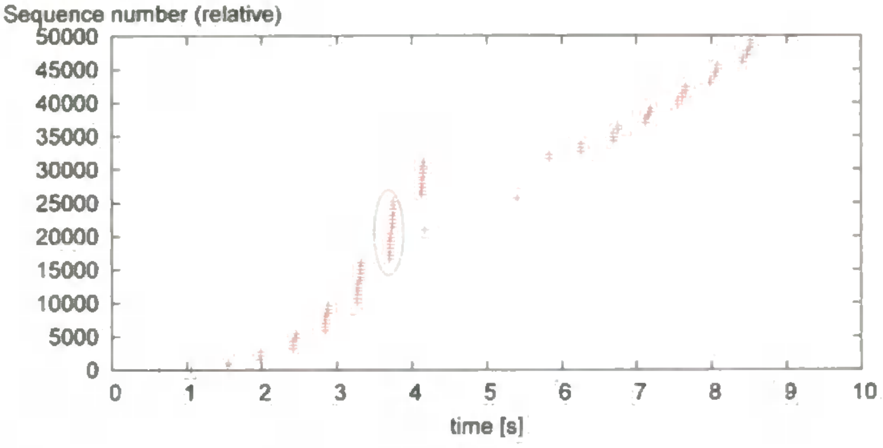


Figure 5.18 – Plot of the relative sequence numbers for a connection

The connection shown above had two lost packets, both around 3.75s. Both losses were part of the same round (appearing circled in the figure). The size of the congestion window at the end of that round was 18 data segments; the dropped segments were the 9th and the 18th. The first loss was retransmitted due to the duplicate acknowledgments received for the 10th – 17th segments. This led to a window of 10 segments (inflated from 1 segment to 10 segments by the duplicate acknowledgments). The second loss was retransmitted due to timeout because the receiver issued only one cumulative acknowledgment after receiving the missing segment. As a result, the congestion window was reset to 1 segment and the sender switched to slow start. A comparison between the real values and the inferred ones is given below in Figure 5.19

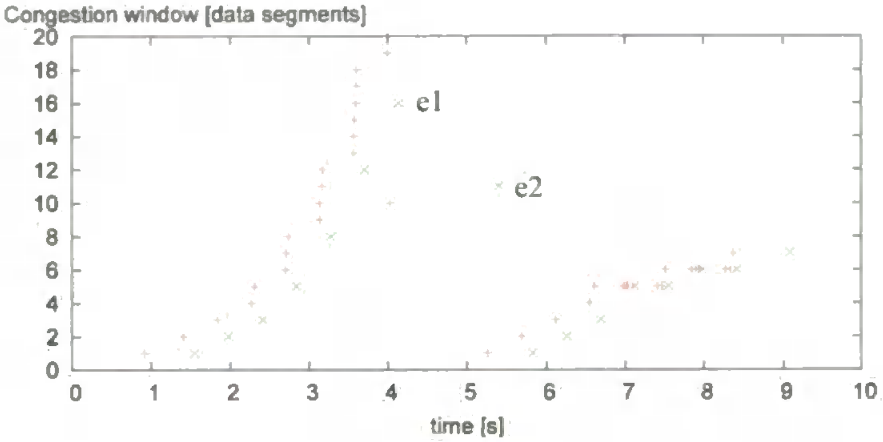


Figure 5.19 - Plot of congestion window values, as resulting from (+) *NS* internal variable cwnd_ and (x) TCP analysis inference (based on a connection with losses)

It may be seen that the inferred values followed the internal variable in all but two cases, marked as *e1* and *e2* on the above figure. The first error (*e1*) was produced due to the two lost segments missing from their correspondent window. The monitor counts the number of segments rather than the covered sequence space and, because 2 segments were lost from the 18-segment window, it produced a figure of 16 segments. The second error (*e2*) was produced due to the retransmission of the first lost segment, together with the 10 data segments which formed the actual window, resulting in an inferred value of 11 instead of 10 segments.

5.6 Summary

This chapter presented a collection of various tests that aimed to evaluate the accuracy of the TCP monitoring method proposed in Chapter 3, using two types of environments: a controllable real environment, produced with a network emulator and a variety of synthetic environments, based on a network simulator (*NS*).

The network emulation round led to a mixture of results for the TCP analysis. The RTT tests led to low relative errors, increasing only towards lower delay values, likely to be due to the network propagation and additional processing delays. According to the tests, the method appeared to estimate less accurately the packet loss, with only less than 50% of the connections leading to correct estimates. It was observed that, aside from the incorrect interpretation of the packet sequence by the TCP monitor, there were multiple sources of errors, all likely to affect the result of the packet loss figures reported by *NIST Net*. Finally, the bandwidth tests were inconclusive due to emulation limitations within *NIST Net*. To compensate, a round of tests was run in a real environment with controlled endpoints. The results were very good, indicating very low relative errors in the bottleneck bandwidth estimation.

The results obtained from the synthetic traces indicated high accuracy for delay, bandwidth and congestion window estimation. The trace support provided the means to study the differences between the estimated packet loss and the logged loss events. The analysis revealed that the estimated loss is in fact, as highlighted in section 4.2, a better approximation of the network loss as inferred by the client. This figure, although it does not indicate the real number of lost packets, is more relevant when evaluating the performance achieved by a connection.

After benchmarking the accuracy of the method proposed throughout this chapter, the next chapter will apply the developed implementation on real traces. The discussion will present the characteristics and limitations of the environment used, then will describe the findings of the trace analysis, aiming to offer a holistic image of Internet paths and web transfers.

Chapter 6. Analysis of Internet traces

6.1 Introduction

This chapter presents the results obtained when applying the TCP analysis method proposed in Chapter 3 onto real network traces. The aim is to obtain an image of Internet paths parameters, as observed from a single collection point.

The traffic produced employed two different methods: either provide a web client that includes scripting support with a list of web pages to visit, or use actual traffic, as produced by real users. The two alternatives correspond to the "semi-controlled data" and "real data" cases introduced in section 2.6. The traffic was collected and stored in packet traces for offline analysis. It is true that offline trace analysis does not make use of the online capabilities of the developed method, but it allows repeatability of the analysis and changes if certain algorithms appear to introduce errors.

The study of the results will follow the range of network- and endpoint-related parameters that the method can infer: packet loss, round-trip delay, bottleneck bandwidth, and congestion window values. It is aimed to observe whether influencing factors are reflected in the resulting distribution, particularly when analysing the variation of delay. The images obtained from the two sources of data (semi-controlled and real) will be compared to determine whether the overall network characteristics are affected by the amount and diversity of data collected.

6.2 The RYL traces

The semi-controlled data collection was based on the idea used by Neil Cardwell to prove the efficiency of his TCP performance model in [Cardwell *et al* 2000]. He used the *Random Yahoo*

Link (RYL) [Yahoo 2003], a Common Gateway Interface (CGI) script within the Yahoo website that redirects the HTTP client to a random web page, to connect to a few web-sites and use the resulting traces to validate his model. The same principle was used for this project, i.e. the functionality of RYL, but on a much larger scale, as the purpose was not only to validate the TCP model, but also to build it.

The RYL-based experiment was run in two rounds, the first during the autumn 2001 and the second during spring 2002. The resulting traces were filtered repeatedly to remove unfinished, reset, and incomplete connections, (which were considered inappropriate for TCP analysis), as well as 1-packet object transfers (which were producing inconsistent throughput figures and did not indicate any TCP behaviour). The filters applied depended on the purpose of the analysis; as will be shown in section 7.1, bottleneck bandwidth analysis, for example, required TCP timestamps present, and any connections that did not use them had to be removed from the bandwidth statistics. However, RTT estimation did not require them; therefore such connections were kept in the delay statistics.

The retrieval used ***wget***, the command-line HTTP client used previously in the ***NIST Net*** experiments, and was automatic, controlled with a script described in Appendix B.3. The capture was performed, as in previous cases, using ***tcpdump***; all the processes (control, retrieval, and capture) were running on the same computer. When the first round of experiments was run, the latest version of ***wget*** at the time did not allow for a full download of the web pages (e.g. for a page with one or more images or embedded objects, only the HTML file was retrieved). At the second round of experiments, the newer version of ***wget*** had the facility to parse web pages and download the objects hosted on the same server with the page), which allowed a rough estimation of the actual content of the page. In the case of an HTTP1.1 client, these objects would be downloaded in a single connection, which gave an approximate indication of the actual length, in terms of size, of a TCP connection making full use of HTTP1.1. Even at the second round of experiments, after all the enhancements, ***wget*** still proved to be restrictive for the RYL experiments due to its limited capabilities. Its latest version did not include some major functionality such as the support for frames, persistent connections, or pipelining[1], all because the program did not support HTTP v1.1

The resulting traces were successfully used in the model training and testing, as they contained all the necessary data for analysis (the ***snaplen*** variable within ***tcpdump***, which controls the number of bytes saved from each captured packet, was set to 300 to retrieve, aside from the TCP/IP headers, some of the HTTP tags).

6.3 The real network traces

NIST Net introduced, as seen in the previous section, several errors when reproducing real network environments, in spite of *emulating* network conditions, errors that affected the accuracy of both monitoring and prediction methods. In order to overcome this problem, the data collected had to be real traffic, produced by real clients in real networks – real traces or *uncontrolled* data, according to the above taxonomy.

The procedure of traffic collection is straightforward for 10/100Mbps Ethernet networks: connect a computer at the tapping point and capture all the packets going to the uplink. The capture requires only a computer equipped with an Ethernet network card and it can be done using, as in previous case, ***tcpdump***, the Linux packet capturing program.

The main problem relating to network trace analysis is gaining access to an aggregation point.

[1] According to the author, there are no plans to expand it in the future in these areas

The two main concerns of a network administrator when allowing such a device to be connected in his/her network relate to service disruption and privacy. Service disruption may occur only at the hardware setup time, if a hub must be connected to the network core, and along the monitoring process, if the hardware used to interconnect the capturing computer (the cables and the hub) is of inferior quality and degrades the system. These issues are less important when compared to privacy: the saved traces will contain all / parts of user data that travelled through the network.

The trace collection required a trusting relationship between the network administrator and the person collecting the data. Due to the limited contact with ISPs, the only place it was possible to collect traces was the University of Plymouth network backbone. The capturing used port mirroring at one of the routers that connect the UoP end-network to the JANET uplink. The program used for capturing was ***tcpdump***, with a *port 80* filter applied, in order to collect only the HTTP traffic

The UoP trace collection, in spite of appearing to be the ideal solution, had several limitations, relating to the size of the environment, OS characteristics, and diversity of the studied environment.

It may be argued that the browsing behaviour and the type of websites accessed may be considered rather limited, because the network users (students, lecturers, and researchers), may have had similar interests. The question is whether conclusions drawn from this network are comparable with results provided by larger traces, collected from higher aggregation points.

The second issue, OS characteristics, impacted only one of the estimated parameters, network bandwidth; the problem was caused by a limitation in the monitoring method rather than a problem of the network. The method uses the TCP timestamp options header to estimate the bottleneck bandwidth, as explained in section 3.4.3.2. Windows 2000 was the typical desktop OS at the time when the traces were collected. Although the TCP client includes implementation of TCP timestamps, their use was not enabled by default. However, recent versions of Linux distributions have all TCP timestamps implemented and enabled in the TCP client. As a result, the resulting bottleneck bandwidth estimation is likely to have been produced only by the requests from the Linux machines.

The third problem encountered was the variety of the environment. Regardless of the amount of aggregate traffic collected, the topology of the environment was rather static: the University of Plymouth network was connected to the Internet, through the Joint Academic Network (JANET) – the network that connects the education organisations throughout the UK [UKERNA 2003]. As a result, the first 8 hops of all paths were within the JANET infrastructure, as shown in Figure 6.1 by a ***traceroute*** run from the computer on which some of the RYL tests were run to www.hotmail.com. Because of this, the first 8 hops were common for all connections, but the routes diverged at the exit from JANET, depending on the destination host.

```
bogdan@tester:~ > traceroute www.hotmail.com
traceroute to www.hotmail.com (64.4.54.7), 30 hops max, 40 byte
packets
1  141.163.77.253 (141.163.77.253)  1 ms  0 ms  0 ms
2  141.163.7.14 (141.163.7.14)  0 ms  0 ms  0 ms
3  141.163.58.250 (141.163.58.250)  1 ms  1 ms  0 ms
4  man-gw-1.bwe.net.uk (194.82.125.177)  6 ms  5 ms  4 ms
5  bristol-bar.ja.net (146.97.40.101)  4 ms  4 ms  4 ms
6  po13-0.bris-scr.ja.net (146.97.35.29)  4 ms  4 ms  4 ms
7  po0-0.read-scr.ja.net (146.97.33.10)  5 ms  5 ms  5 ms
8  london-bar3.ja.net (146.97.35.126)  6 ms  6 ms  6 ms
[...]
```

Figure 6.1 Part of the traceroute output from UoP network to www.hotmail.com

As a result, the performance of the resulting traffic was exclusively influenced by the position of the remote endpoint. For remote endpoints connected in 'similar or better' networks (i.e.

networks connected to UoP through paths with comparable or larger bandwidth and low RTT), the distributions obtained for loss, delay, bandwidth, and, implicitly, for throughput, are fairly narrow. More details on the network environment limitations and on the impact they may have had on the resulting statistics are offered in section 7.1.1. It is worth mentioning here that the connectivity parameters did change in time, as the UoP network went through a major internal upgrade in the autumn-winter of 2001. This upgrade had a radical impact on the quality of the network, improving the end-to-end parameters as it will be seen throughout section 7.1

With regards to the public traces, the only source identified was the Passive Measurement and Analysis initiative [PMA 2003]. The PMA holds an archive of relatively large network traces collected from 31 different locations using various physical network technologies to connect, all 'sanitised' to protect the identity and privacy of the users that created the traffic. The advantages are obvious with regards to this source of data: variety of environments, amount of collected traffic (which lead to a higher possibility to find traffic generated from non-Windows clients). However, there were three main issues when using this traffic for analysis: the data formats, the way the data flows were collected, and the content of the packets in the trace.

1. Data formats. The traces from NLANR (the umbrella of PMA and AMP, as explained in section 2.3) come from different network environments (e.g. ATM, FDDI, Packet over SONET) and were captured using different programs / hardware (coral, DAG3, etc), which lead to a variety of formats for data. This obstacle was easy to overcome, as the NLANR website provides a wide range of tools that allow conversion from any of the capturing formats to pcap, the format used by the implemented traffic analyser.
2. Data collection. Because they were collected more for overall analysis (e.g. number of flows, number of bytes per flow, number of bytes per packet), a lot of the traces, especially from ATM, include the traffic only one direction, rendering any TCP analysis

impossible. Some of the traces, the ones captured using the CoralReef suite [CoralReef 2003] were bi-directional but, nevertheless, they had a clock drift between the two interfaces, which lead to erroneous figures for the delay statistics.

3. Captured content. All the traces were 'sanitised' for privacy purposes; as a result, none of them included any TCP extensions or data content. This affects the performance estimation, as no inference can be made in regards to the bottleneck bandwidth (using the TCP timestamps options) or the server that generated the traffic (using the HTTP headers).

For the above reasons, the real trace analysis was limited to studying the UoP traces. Details on the actual statistics obtained are presented in section 7.1. The traces were also used in the validation process of the proposed TCP performance prediction method, described in details in Chapter 8.

6.4 Random Yahoo Link traces analysis

6.4.1 Network topology and connectivity

It is fair to admit that the RYL analysis was limited in the sense that there was a single connection point, the UoP network. Further, the UoP network is connected to the Internet, via a single link to the UK academic network, JANET. This is why, after the spring 2002 round of connections, an experiment was carried out in order to estimate the diversity of individual paths explored by the RYL connections. The experiment consisted of running traceroute on a random subset of the sites - 350 out of the 2744 unique servers which were contacted during this second round. It is true that the experiment was not carried during the data collection but after it, which may have introduced slight alterations to the results. In spite of the time difference, it may be assumed that the overall distribution did not change considerably (its shape resembles the one from prior studies such as [Hyun *et al* 2003]).

Due to the topology characteristics, all the routes were the same for the first 8 hops (up the Janet backbone). Because of this, the traceroute command was started each time with an initial TTL of 9, to reduce the number of packets generated. Some of the traces failed to reach the destination due to one of the following reasons:

- Long path. The path up to the remote endpoint was longer than 30 hops. Such traces showed no errors and a valid IP address at the last hop, but this last hop was different from the targeted address. This category could have been removed by using the "-m" option of traceroute but, unfortunately, the option was not considered at the time of the experiment
- Administrative prohibitions. Routers that did not respond to the traceroute probes (or dropped them altogether). Such traces included valid IP addresses up to a certain hop, then either no response for any of the following hops (up to hop 30) or an *administratively prohibited* flag. This is one of the reasons why traceroute-based path measurement tests may prove to be less successful in comparison with non-intrusive TCP analysis.
- Host unreachable errors. A remote router would report that the requested host is unreachable.

Out of the 350 sites, 23 probes failed due to administrative prohibitions, 22 paths were longer than 30 hops, and in 91 cases one of the routers along the path dropped the probe. These 136 probes, although not complete, indicated the route up to the router that was prohibited / did not respond / reported a host unreachable. This is why, when evaluating the variety of paths, they were taken into consideration up to the router that stopped the probe.

Two sets of paths were created: one containing all the 350 probes (but with the faulty entries removed), referred to here as *all_probes*, and one containing only the 210 probes that reached their destination, named *good_probes*. The paths from the two sets were then compared hop-by-hop, i.e. hop 9 from all routes, then hop 10 from all routes, and so on, up to the last router before destination, to determine the number of unique IP addresses for each hop. The method is not 100% accurate, as some paths might use two or more alternative sub-paths between the same two nodes for load balancing, but gives a rough estimate of the number of unique paths followed by the packets. Also, in the case of the *all_probes* dataset, some of the routes were incomplete, indicating that even higher variety could be found in the paths. The experiment is somewhat similar to that run by Paxson in [Paxson 1997a] to estimate the characteristics of the Internet but, in this case, the only purpose is to provide information about the topology of the remote sites in relation with the collection point (UoP network).

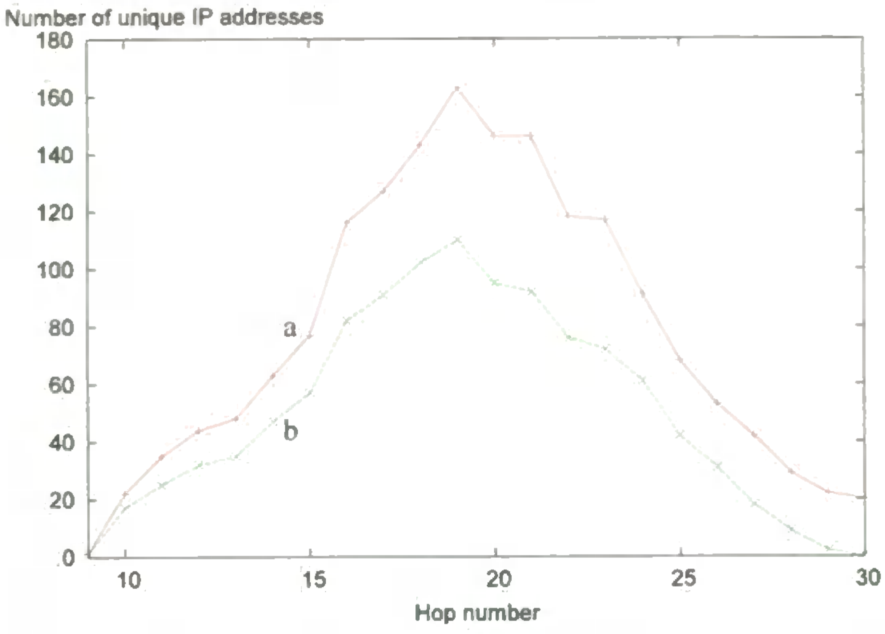


Figure 6.2 - Path distribution for the (a) *all_probes* and (b) good_probes sets, spring 2002 experiments

The results of this analysis are presented in the distribution from Figure 6.2. The graph shows a high number of unique routes, particularly considering the fact that the last hop (the target) was removed from each path. The maximum number of unique IP addresses was, in both datasets, recorded at 19 hops - 163 for *all_probes* and, respectively, 110 for the *good_probes* dataset. These figures are equivalent to a percentage of approximately 50% unique paths in the case of both datasets. Also, the average path length (calculated only for the *good_probes*) was 23.15 hops.

One final issue in relation to the efficiency of this probing technique is the redundancy of Yahoo's randomness. As shown in previous studies [Evans 2001], while RYL provides a very convenient interface for random pages, it has a bias towards certain web sites. This observation was also reflected in this study. Amongst the 21654 requests made (after removing the redirections from the Yahoo website), there were only 11356 unique IP addresses. This redundancy may be seen below in Figure 6.3. The values beyond the value of 100 (connections / unique IP address) were due to only three addresses[1], but it also shows that more than 80% of connections were made to unique hosts. In order to avoid redundancy, the resulting datasets should be run through a filter to keep only one sample from each IP address. However, this approach is inconvenient for the second round of experiments, due to the fact that each query requested all objects from a page and such an IP-only filter would keep only one of the connections[2]. A second type of filter may be applied in order to keep all the unique objects retrieved from an IP address, which works similarly to a cache memory: remove duplicate samples by comparing their source web server and the size of the retrieved object.

None of the two filters represents the perfect solution. The first one, IP-only based, provides accurate path information (particularly delay), but it would not reflect true characteristics of the traffic, particularly with the second round of experiments. The second filter, IP-and-size based, distinguishes between different objects retrieved from the same IP address, which allows a better analysis for typical page transfers, but it may bias the path-related results. This is why, for the analysis, the RTT subset was extracted using the IP filtering, while the loss and congestion window analysis used the IP-and-size filtering.

[1] All three addresses were hosting at the time sites likely to be popular sources of data: one of them was resolved as news.bbc.co.uk website and the other two were resolved as www.cnn.com

[2] The kept connection is also likely to be the shortest one the one - the html text rather than e.g. the images on the page.

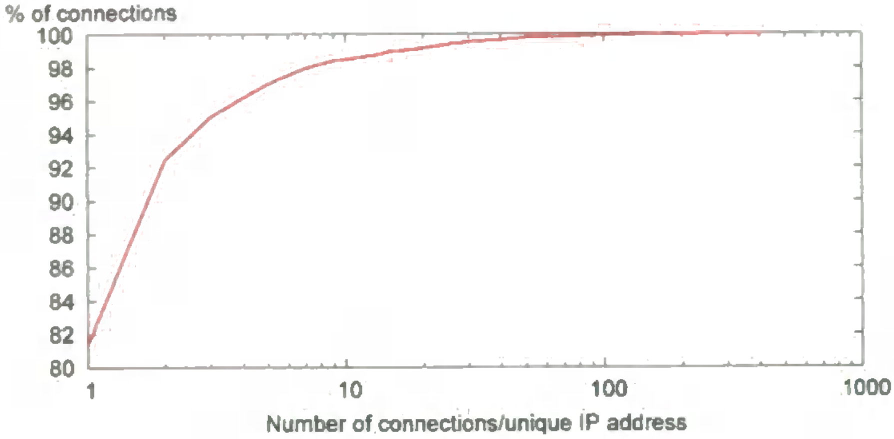


Figure 6.3 - Distribution of connections/unique IP address

Generalising, this experiment indicates that, once redundancy is removed, the objects retrieved during the RYL experiments were diverse, in terms of the number of paths involved. It was preferred to run this trace only on a random subset of addresses in order to avoid triggering any network alarms / lead to external complaints, fact that happened in the recent history of the Network Research Group [Evans 2001]. Unfortunately, this analysis could not be backed up by a geographical analysis due to the (limited) amount of information provided by *whois* queries.

The following sections present the results obtained from the analysis of the two rounds of traces. In each case, an additional filter was applied to the data in order to maintain only the valid results for that type of analysis (e.g. for the round trip time results, the connections that did not yield at least 3 RTT measurements were removed)

6.4.2 Round Trip Time statistics

The first metric analysed was the round trip time. The traces yielded a low number of RTT measurements for each connection, mainly due to the position of the capturing device relative to the endpoints, i.e. right at the receiver. Still, there were three RTT measurements within each connection: first one in the synchronisation sequence, the second when sending the page request, and the last when closing the connection, enough for establishing the average and variance values for round trip delay.

The distribution of the RTT for the two sets of experiments is presented in Figure 6.4. As can be observed, in both cases the average RTT values are very low for most of the connections, with an overall average of 192.4 ms for the first round of experiments and 149.6 ms for the second round. The difference between the figures may be associated with the network upgrade mentioned previously (unfortunately, there was no path information collected during the autumn 2001 experiments), as the shape of the distribution remained the same for the two sets of results.

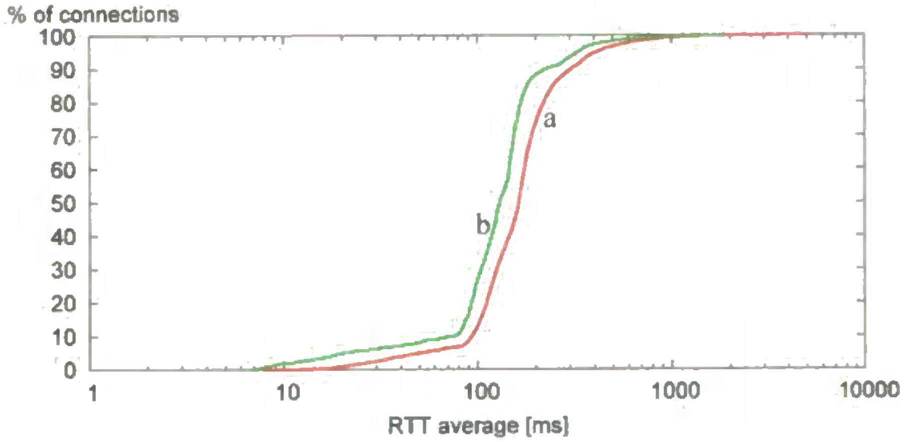


Figure 6.4 –RTT average [ms] cumulative distribution for: (a) autumn 2001, (b) spring 2002

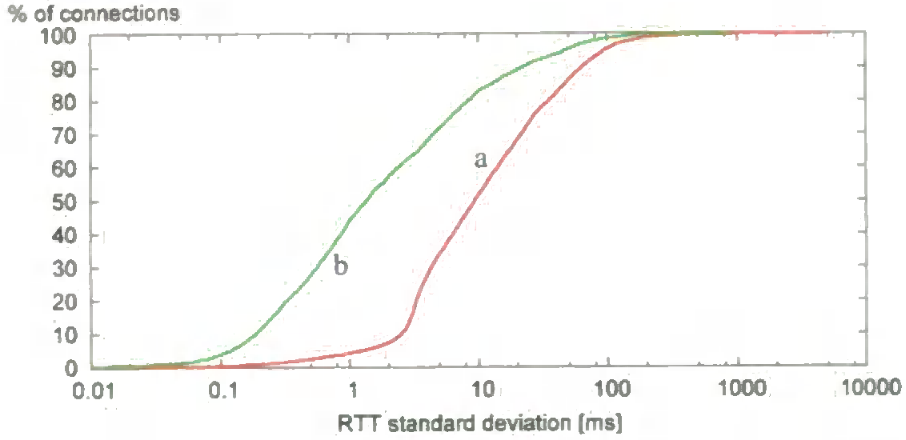


Figure 6.5 - RTT standard deviation [ms] cumulative distribution for: a) autumn 2001, b) spring 2002

Aside from the actual value of the RTT, the standard deviation of the RTT throughout a connection was calculated; the result is shown in Figure 6.5. The average value of the standard deviation was 22.3 ms (10.4 % of the RTT averages) for the autumn 2001 round and 7.8 ms (4.7 % of the RTT averages) for spring 2002. The data results from spring 2002 indicate that, for more than 80% of the flows, the RTT standard deviation was less than 10ms.

Statistical T-testing was employed in order to evaluate the two rounds of measurements. The analysis was slightly different from the one applied in Chapter 5 because this time the two rounds of measurements were independent, with no criteria to group the two sets of results. The hypothesis was that the overall RTT average for autumn 2001 was higher than the average for spring 2002. The hypothesis made was that the average RTT from the autumn 2001 dataset was higher than the average RTT from the spring 2002 round of measurements. The method required calculation of:

- The pooled average s for the standard deviation σ of the sample mean difference for the two measured variables:

$$s^2 = \frac{\overline{X_1} + \overline{X_2}}{n_1 + n_2} \qquad 6.1$$

where:

- o $\overline{X_1}$ and $\overline{X_2}$ are the overall average RTT values for the autumn 2001 and, respectively, spring 2002 measurements round
- o n_1 and n_2 are the number of connections yielding RTT values for the autumn 2001 and, respectively, spring 2002 measurements round

- The variance for the difference of the two measured variables:

$$s_{\overline{X_1}-\overline{X_2}} = \sqrt{s^2 \frac{n_1 + n_2}{n_1 n_2}} \qquad 6.2$$

- The t quantity, $t = \frac{\overline{X_1} - \overline{X_2}}{s_{\overline{X_1}-\overline{X_2}}}$, that follows Student's t distribution

The calculation for RTT average returned the following values for the above variables: $s^2 = 29786$, $s_{\overline{X_1}-\overline{X_2}} = 4.07957$, and $t = 10.9325$. The value for t is higher than the 0.1% probability, $P_{0.1\%} = 3.2905$, for a measurement with ∞ degrees of freedom. This leads to the conclusion that the average RTT for the autumn 2001 dataset was higher than the average RTT for the spring 2002 dataset by 44.6 ms with 99.9% confidence limits 31.17 ms and 58.02 ms

The same method was applied to the RTT standard deviation, yielding $s^2 = 9352.36$, $s_{\overline{X_1}-\overline{X_2}} = 2.28595$, and $t = 8.35606$. Using the 0.1% probability value, $P_{0.1\%} = 3.2905$, the conclusion was that the standard deviation of the RTT for the autumn 2001 dataset was higher than the average RTT for the spring 2002 dataset by 19.1 ms with 99.9% confidence limits 11.57 ms and 26.62 ms.

The results obtained for RTT average may be compared with Allman's findings [2000] and show an improvement in the connectivity, as that study found that 85% of connections had an RTT in the 15-500 ms interval[1]. In this study, the [0;500]ms interval tends to cover nearly all the spectrum (95% for autumn 2001 and 98% for spring 2002), while 85% of connections had an average RTT of less than 350ms in the spring 2002 dataset and less than 170ms for the autumn 2001 dataset.

The analysis method presented in Chapter 3 allows RTT measurements based on the TCP timestamp option. Unlike the case of sequence-number-based RTT, TCP-timestamp-based measurement produces an estimate for virtually each acknowledgment / every-other data packet, based on the pairing of the timestamp values returned in each packet. The main disadvantage of this method is that the remote endpoint may not implement the TCP timestamps mechanism correctly (or at all), in which cases the RTT estimate(s) should be discarded. In the autumn 2001 round of experiments, the sequence-based RTT inference produced estimates for 10394 connections to unique remote IP addresses. On the same trace, the TCP-timestamp-based allowed estimates only for 5343 connections. Still, this situation may be remedied in the future, with an increasing percentage of TCP clients implementing TCP timestamps[2]. A good example of such an evolving client is the Microsoft Windows TCP/IP stack, which did not include support for TCP timestamps before the Windows 2000 version [Microsoft 2000].

6.4.2.1. Ack-based RTT vs. TCP timestamp-based RTT estimations

The expected outcome of the timestamp-based RTT measurement was, as indicated in RFC1323 [Jacobson *et al* 1992], a substantially larger number of RTT inferences for each connection. Figure 6.6 presents the distribution of the sequence-based vs. the timestamp-based RTT

[1] There is no information in the study about average values or the percentage of connections with RTT<15 ms.

[2] According to a series of studies from Netcraft, the estimated percentage of web servers that implemented to TCP timestamps option increased from 38.1% in 2000 to 80.5% in 2003 [Wendland 2003]

estimations ratio per connection. Due to the position of the capturing point in relation with the endpoints, i.e. near the receiver, the distribution illustrates only the figures for client-server-client RTTs[1].

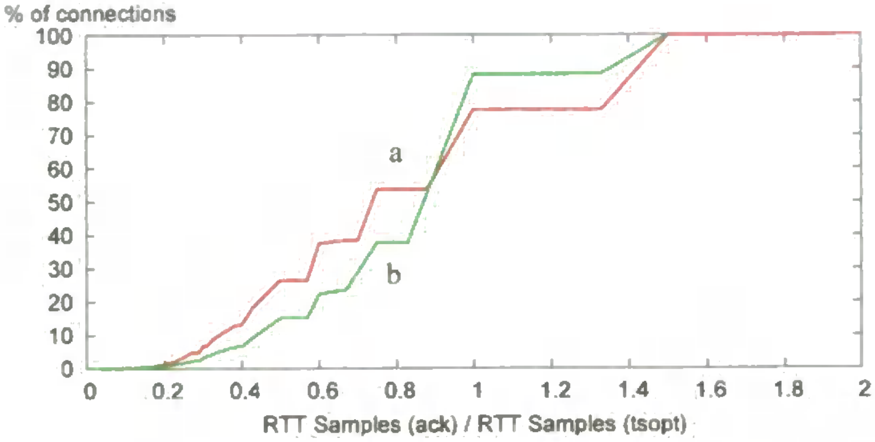


Figure 6.6 – Distribution of RTT samples based on acknowledgments vs. RTT samples based on TCP timestamp options for a) autumn 2001 and b) spring 2002

It may be observed in the above distribution that the vast majority of the connections experienced a comparable number of timestamp-based and sequence-based estimations. The average value of the rapport was 0.865 for autumn 2001 and 0.899 for spring 2002). This result, less encouraging than the RFC1323 expectations, may have been caused by two factors:

- size of the connections (subject expanded within this chapter, in section 6.5.4). If the actual number of transferred data packets is low, the number of RTT inferences will be low, regardless of the method used. A good example is an HTTP request that returns an object which can be transported in two data packets. The associated TCP connection cannot have more than 3 RTT estimate / direction, as the entire data transfer consists of:

[1] These are inferences resulting from the server issuing acknowledgments to the data packets sent by the client (e.g. the acknowledgment to the HTTP client request)

  - 3-way handshake to open the connection – 1 RTT estimate / direction
  - client transmits the request; server acknowledges and transmits 2 back-to-back packets; client replies with 1 acknowledgment – 1 RTT estimate / direction
  - 3-way handshake to close the connection – 1 RTT estimate / direction
- resolution of the TCP timestamp. The timestamp-based inference cannot produce more than 1 estimate for each unique timestamp, i.e. a new RTT estimate can be produced only when the value of the timestamp increases. This impediment affects the number of inferences and it may have an effect on the average RTT value obtained for a connection only if the RTT throughout that connection varies considerably. The sequence-based analysis indicated an average of nearly 100 ms for the spring 2002 dataset. If the resolution of the sender-generated timestamps is lower than the RTT values, several packets (in some cases, for low-delay and high-bandwidth paths, even all packets) may carry all the same TCP timestamp, reducing the number of estimations for that connection. Figure 6.7 presents the distribution of the timestamp resolution for the two rounds of experiments. It may be noticed that the majority of connections (around 68% of the probed websites during autumn 2001 and 82% of the sites from spring 2002) were produced by senders with a 10ms resolution of the TCP timestamp. It is interesting to note, according to the distribution, that the difference between the two figures was made up of senders that had a 100ms resolution for the TCP timestamps. It is difficult to determine whether this difference was made up of websites that upgraded to a newer version of operating system (one which came with more accurate timestamps) or these are only websites which have not been probed during the spring 2002 experiments[1]. The rest of the senders had timestamps of at least 100 ms, with almost 20% of the TCP senders using a resolution of at least 200ms, comparable with the RTT values for autumn 2001. With such values, aside from limitations induced in the RTT measurement, the

senders themselves cannot take advantage of any of the benefits that TCP timestamps introduce (e.g. better RTT estimates).

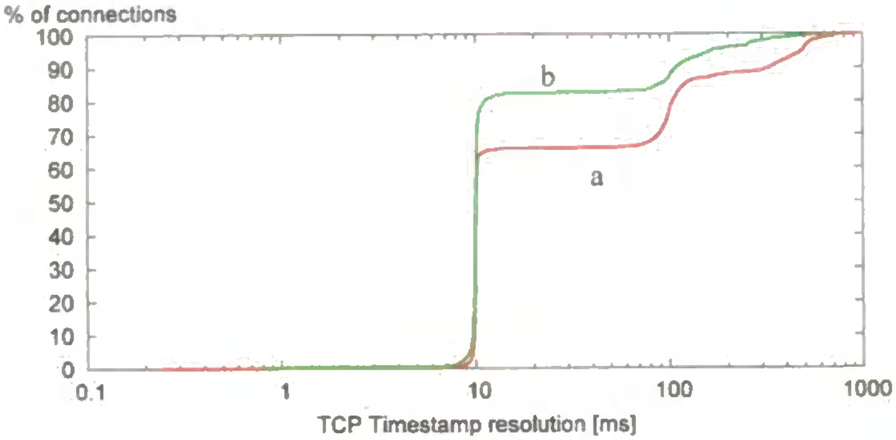


Figure 6.7 - TCP timestamp resolution for a) autumn 2001 b) spring 2002

6.4.2.2. Effect on current TCP implementations

The real network values for RTT should be considered when implementing TCP timers, which is not currently the case. RFC 1323 does not set a specific limit for the granularity of the measurements, but only requires the minimum RTO to be of 1 second and the minimum variation added to the RTT to be equal to the granularity[1]. It is in fact admitted by the authors that the proposed figures are rather conservative and may be changed by results of future research, but no updates were published since. It is obvious from comparing these low figures for RTT with the coarseness of the retransmission timeout algorithm that these limits are *too* conservative for the current Internet, at least for an environment such as the analysed one. Firstly, the update of the algorithm is too coarse in comparison with the variations of the RTT. It is fair to argue that the number of RTT inferences was low in each case, with only 3 samples /

[1] A larger number of TCP timestamp resolution estimates to unique IP addresses (5108) was made for the autumn 2001 connections in comparison with the number of estimates obtained during spring 2002 (1717).

connection, but these samples were placed at the beginning and at the end of the connection, which gives a good indication about how much the delay changed throughout the transfer. Considering the average RTT value of 136ms from April 2002, a TCP sender will have to wait another 6*RTT before retransmitting the lost segment. It is true that the analysed connections were generated in a privileged environment, but, again, at least for such low-delay environments, the minimum RTO should be changed to an lower limit of, at most, 500 ms. This change should not affect the initialisation value for RTO, which may remain at 3 seconds in order to accommodate high-delay environments such as satellite links.

The main issue when looking at RTO is how often the timeout mechanism is likely to be used. The retransmission policy depends, as explained in Chapter 2, on how the loss is observed. Due the way congestion window evolves, for paths experiencing low figures for packet loss, the larger a file is the more likely it is for losses to be recovered through by fast retransmission, because more acknowledgments are in flight at any moment in time. At the other extreme, short-lived connections will experience small congestion windows and the TCP senders are likely to recover losses through timeout instead of fast retransmission. Unfortunately, the connection analysis performed as part of this study (section 6.4.5) showed that most flows are short-lived and the congestion window experiences at low values, which indicate that, on a path experiencing loss, TCP senders are likely to timeout instead of fast retransmit for typical web transfers.

Our observations can be compared with the findings of Allman and Paxson, who modified the proposed values for granularity and RTO in [Allman and Paxson 1999]. The conclusions of this study were that, over the analysed mesh of Internet paths, a coarse timer (e.g. 500 ms) would reduce by half the time spent while waiting for the retransmission timer to expire but would

[1] It only mentions that a fine granularity, defined as less than 100 ms, performs better than a coarse one

double the number of *bad retransmissions* (i.e. retransmissions due to an erroneous low RTO). The increase was even higher when a lower figure was chosen for the minimum RTO but, as Allman observed, one of the main reasons why this happened (and, generalising, why the minimum RTO should not be changed on its own) was the timing of delayed acknowledgments. According to the recommendations [Braden 1989], delayed acknowledgments *should* be delayed for *at most* 500 ms, a rule that allows the receiver to delay the receipt of a data segment for up to 500ms. In fact, [Paxson 1997a] and [Allman and Paxson 1999] are the most quoted studies that critically analysed the effect of the proposed values on Internet TCP transfers.

6.4.3 Bandwidth

The estimation was based on identifying back-to-back packets using the TCP timestamp option, as described in section 3.4.4.2. The position of the monitor was extremely favourable for this type of measurement, as the data packets coming from the server were captured right at the receiver, i.e. at the end of the path, allowing identification of the bottleneck for the download path. Some of the senders had coarse TCP timestamp clocks, therefore the estimation used both of the timestamp values within each data segment in order to determine the back-to-back pairs. The rationale behind using both values is that back-to-back packets should be transmitted due to the same acknowledgment, therefore they should carry the same tsres value. Unfortunately, only approximately 50% of the connections produced a reliable bandwidth estimate in each round of experiments, statistics likely to be due to erroneous implementation of the TCP timestamps at the senders, or due to connections with too few packets. The result of the measurement is illustrated in Figure 6.8.

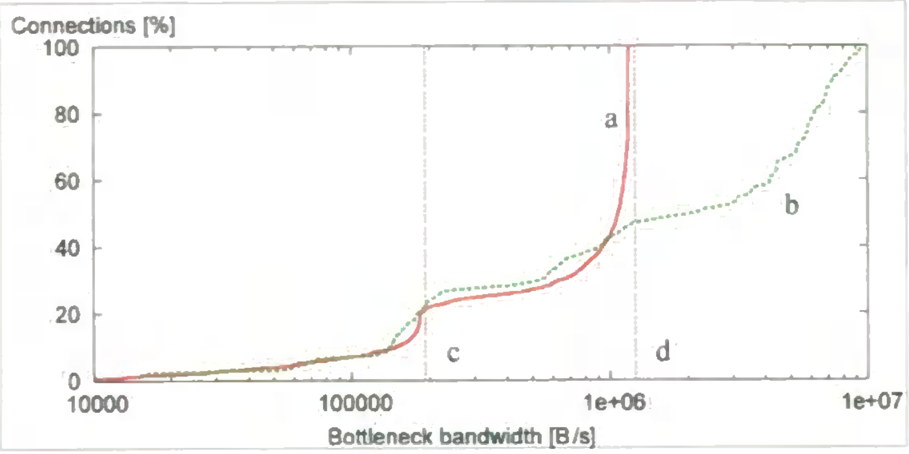


Figure 6.8 – Bottleneck bandwidth cumulative distribution for a) autumn 2001 and b) spring 2002. The two grey markers indicate c) the T1 (1.544Mb/s) boundary and d) the 10Mb/s boundary

From all network characteristics, the network upgrade mentioned earlier affected bandwidth the most. It can be noticed in the distribution from Figure 6.8 that bandwidth reached a maximum of approximately 1.2MB/s for the autumn 2001 round of experiments. This matches, in fact, the configuration of the network: at the time of the experiment, the connectivity of the desktops was 10Mb LAN. For the spring 2002, the maximum figure is 12MB/s, which reflects the tenfold increase in desktop bandwidth. The reason why the 2002 graph is not asymptotic to this second limit is clock accuracy of 1µs which becomes a factor at 100Mb/s as a full Ethernet frame requires only 120µs to be transmitted.

6.4.4 Loss

Due to its self-adjusting behaviour [Jacobson and Karels 1988], TCP performance is critically affected by loss. Nevertheless, previous studies [Paxson 1997a], have shown that packet loss is low, at least for the monitored mesh of Internet paths. A major purpose of this study was to investigate the current *typical* values of network parameters, but based only on traces collected from a single point. From this perspective, the survey may appear similar to [Allman 2000], but, in this case, there is no control over the senders – rather than have "one server and many clients", this project used the "one client and many servers" approach. It may be argued that the survey carried out was somehow limited, as the client used, ***wget***, did not support HTTP1.1. As a result, the objects from a page were downloaded in separate connections, a fact that led to smaller congestion windows. Further, the resulting figures for loss may be less accurate than some obtained for long-lived connections, due mainly to larger congestion windows and better resolution, as will be detailed later in this section.

The approach used when analysing the loss results was similar with the one for RTT evaluation. As explained in section 3.4, losses rely on RTT and information, obtained at least from acknowledgments if not from TCP timestamp options too. Also, the loss analysis does not perform well for very short lived connections (i.e. connections lasting for less than 1 RTT). This is why a filter was applied to the datasets to remove any connections that have these characteristics. The initial datasets were reduced through filtering to 11337 samples (2001) and 8585 samples (2002).

There were 3 categories of losses, as defined in section 3.4.5: visible (*LostBefore* and *LostAfter* events), inferred timeouts (*LostTO* events), and errors (losses due to either erroneous timeout at the sender or due to lost acknowledgments from the receiver). The events from the third category, although not produced by genuine losses, have the same effect on the sender - reduction of the congestion window. The remainder of the section will present, for each category, the number of connections that exhibited the event, as well as the associated distribution of losses, measured in packets/connection.

6.4.4.1. Visible loss

The expected result was decrease of losses due to the network upgrade. However, the first category, visible losses, had a strange evolution for the two rounds of experiments. Throughout the (filtered) datasets, 75 connections (0.4%) experienced visible losses during 2001, a figure that increased to 127 connections (0.6%) for the 2002 dataset. The resulting distributions of loss rate (measured in lost_packets / data_packets) for the two rounds of experiments are displayed in Figure 6.9

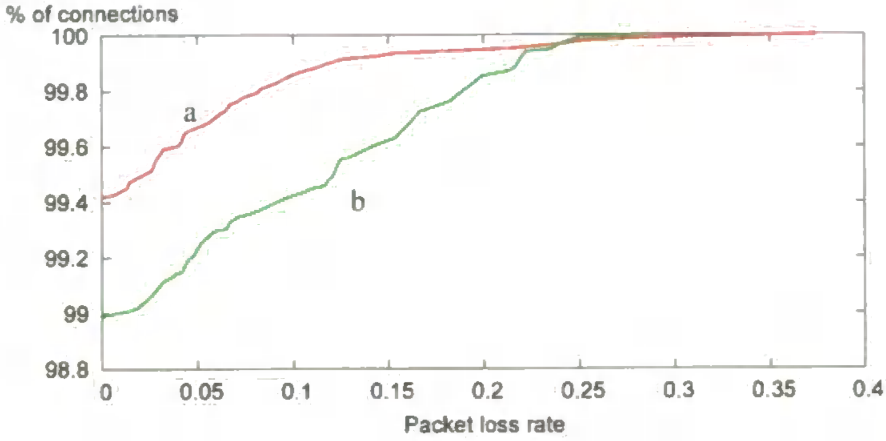


Figure 6.9 – Packet loss rate distribution for visible loss events (a) 2001 and (b) 2002

The anomaly was even higher when looking at the remote endpoints that generated these errors. All but three connections from the 2001 dataset were generated from distinct IP addresses, resulting in a number of 70 unique paths (equivalent with 0.659% of the total number of unique paths) that exhibited loss. Using the same analysis, the 2002 dataset had only 53 unique IP addresses, equivalent to 2.425%. The higher loss rates for the spring 2002 dataset were confirmed also by the statistical t-testing, which produced values of $s^2 = 0.0001445$, $s_{\overline{X_1}-\overline{X_2}} = 0.000167$, and $t = 4.50017$, indicating that the average packet loss for visible loss events for the spring 2002 dataset was higher than the average figure for the autumn 2001 dataset by 0.07% with 99.9% confidence limits 0.02% and 0.1%.

The only explanation consists in the difference between the two rounds of experiments: the 2001 method retrieved only the HTML content from each page, while the 2002 method retrieved the associated objects too. An example of page retrieval from the 2002 experiment is presented in Figure 6.10. There was one lost packet during the connection (bearing the sequence number 11657 and retransmitted at 4.431 seconds)

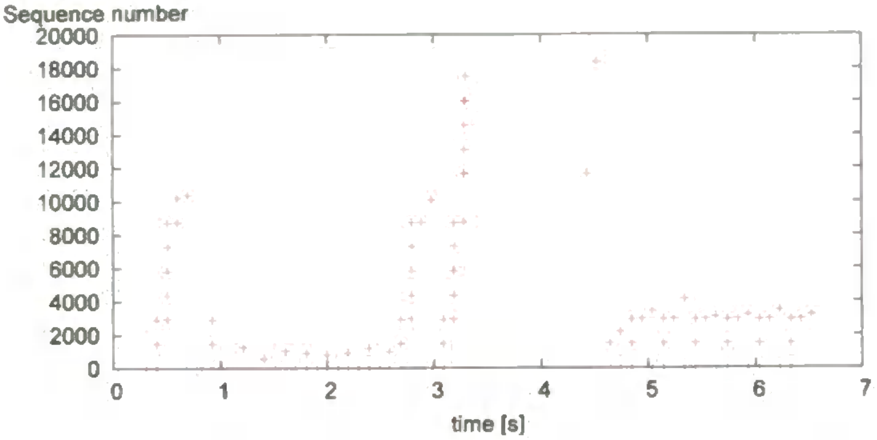


Figure 6.10 – The sender packets from a page retrieval example from the 2002 round of experiments (each packet is represented by the relative sequence number of the first byte)

A total of 19 connections were retrieved during this retrieval, ranging between 968 bytes and 19382 bytes, with an average of 3780 bytes / connection and a total transferred of 71824 bytes in 77 data packets. If a request to this page would have been made during the 2001 experiments, only the first object, measuring 10415 bytes would have been retrieved, probably using 11 packets, as was the case for this retrieval. First, the lost packet happened during the longest connection of the retrieval, measuring 18382 – almost twice the size of the first object. This is an argument in favour of the greater packet loss for 2002, but not a definitive one. A separate analysis showed that the average size of connections with losses was 47534 bytes for 2001 and only 29056 bytes for 2002, which indicates that losses seemed to have happened in shorter connections during the second round of experiments (more details about the distribution of the file sizes is given in section 7.1.5. A second argument comes to justify both apparent anomalies – the higher packet loss during shorter connections for the 2002 dataset. With 1 packet lost out of the 77 data packets transmitted, the overall loss rate for this retrieval was 1.29%. Using the same rationale, the minimum visible packet loss for the first object would have been 1 packet out of the 11 transmitted, equivalent with 9.09%. The above statistics are based on the page size, a variable that may have played an important role in the difference between loss figures. Another variable, not available for study from the perspective of this research, is the actual configuration of the paths beyond Janet that may have changed between the two experiments.

The conclusion is that the 2002 dataset led to better resolution of loss rates for the probed websites/IP addresses. First argument for this is that the 2002 experiments produced longer connections due to some of the objects from a web page being larger than the first object (the one that would have been retrieved using the 2001 method). The second argument is that, by retrieving all objects from a page, the 2002 experiments always retrieved a larger amount of data from each probed page in comparison with the 2001 experiments. These differences indicate that it is difficult to draw a line between path properties and the traffic properties. It is true that the primary target of a holistic Internet study is to look at unique paths rather than packet traces. However, while specific analysis of paths may show relative stability in time, the picture offered by overall traffic analysis may show high variations of the loss figures. This discrepancy may appear due to the higher/lower usage of certain websites[1] or due to the content of the web pages, both of these causes being able to bias the analysis. To expand on this last issue, higher usage of a website will bias the network characteristics towards the path that connects the analysis point with that website. Also, larger web pages will lead to higher number of exchanged packets, therefore are likely to reveal lower loss rates in comparison with the short-lived transfers.

6.4.4.2. Inferred loss

The inferred losses were even fewer than visible losses, which may be due to restrictions posed on the spacing between packets. First, the minimum and maximum timeout periods were limited in order to identify false positives due to small RTT / slow server responses. Second, the identification header field numbers from the two instances of the data segment were compared in order to separate packet duplication events from visible loss events. After applying all these boundaries, only 9 connections from the 2001 dataset and 3 connections from the 2002 dataset were identified with timeout events. A graph of the resulting distribution of losses is presented in Figure 6.11

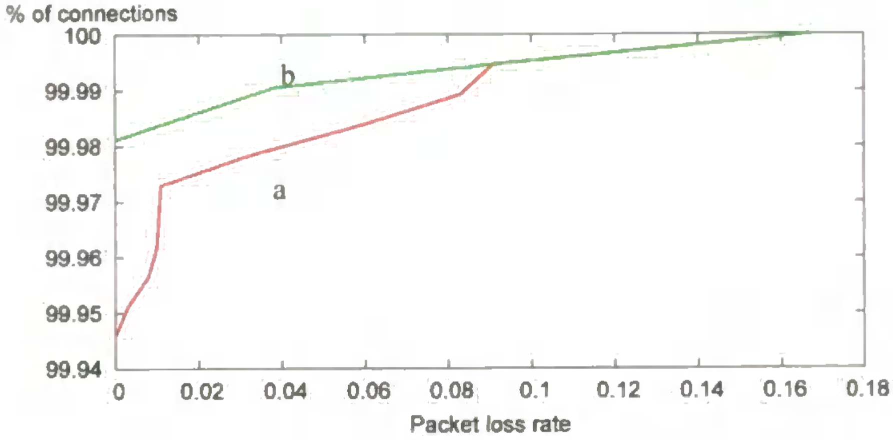


Figure 6.11 - Packet loss distribution for inferred loss events (a) 2001 and (b) 2002

[1] The inversion in figures (i.e. higher loss for the 2002 dataset compared with the 2001 dataset) persisted for the unfiltered set (the one to which none of the filters from 6.4.1 was applied).

6.4.4.3. Overall loss

The First observation to be made is that the image offered by these experiments is virtually loss-free. The main reason for this is, apart from the characteristics of the connectivity point, the actual average size of the objects transferred. The short-lived connections allow only small increase of the congestion window and, implicitly, do not impose stress on the path between server and client.

The short-lived connections have an additional undesired effect: the accuracy of the measurement cannot go beyond the granularity of the download due to the low number of packets exchanged[1]. For example, having a transfer consisting of 10 packets, the minimum detectable loss is 0.1, a situation also described in [ARPA 1981b]. To reduce this granularity error, the total number of losses was compared to the total number of packets captured. The year 2001 connections subset had a total of 166325 packets, with 206 visible and 10 inferred packet retransmissions, producing the overall packet loss figures 0.123 / 0.006 / 0.129% visible / inferred / total). For the 2002 tests, 232 packets were visible retransmissions and 4 packets were inferred retransmissions; comparing this with the total of 125960 packets, results in an overall packet loss of 0.184 / 0.003 / 0.187% (visible / inferred / total).

Due to the low figures obtained, losses were considered to be rather exceptional (inferred losses in particular) throughout the two traces. However, more consistent results were obtained for the backbone traces, analysed in section 6.5

6.4.4.4. Retransmission errors

It became obvious during the analysis that the monitor had an unfortunate position (i.e. at the receiver) for identifying losses due to timeout; the solution offered was to infer these events based on the packet spacing. On the other hand, this position allowed identification of avoidable retransmissions, i.e. retransmissions triggered by the retransmission timer expiring at the sender before the arrival of the required acknowledgment. From the perspective of the monitoring point, a retransmission was avoidable if at least one acknowledgment was captured between the first time and the second time a data segment was transmitted. These losses, appearing to the monitor as visible losses, were not ignored because, regardless of what it actually happened with the packet, the network conditions forced the sender to infer that the data segment was lost. Figure 6.12 presents the distribution of avoidable retransmissions / connection, as observed in the two rounds of experiments.

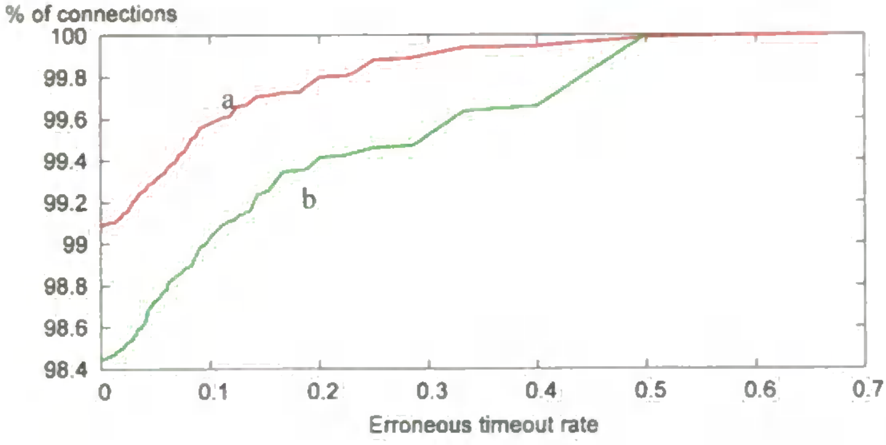


Figure 6.12 - Erroneous packet retransmission distribution (a) 2001 and (b) 2002

It can be observed that 1.6% / 0.9% of the connections experienced such events for the autumn 2001 / spring 2002 experiments. These figures are both higher than the cumulated visible and inferred losses for the two rounds of experiments, which leads to one of the following conclusions / reasons:

[1] This might be compensated at least partially by averaging the 'per connection' results

- Some of the apparent errors may have been unavoidable. The retransmissions were, for this category of losses, due to loss on the reverse path: the acknowledgment sent by the receiver might have been lost on the receiver-sender path.
- Some of the apparent errors may have been not genuine losses, but rather packet arrival sequences that the TCP analysis interpreted as a retransmission, as shown by one of the examples from section 3.4. Further refinements of the TCP analysis may reduce this erroneous interpretation from the analysis method, but would also slow the speed of analysis.

The statistical t-testing also confirmed the graphical results, with $s^2 = 0.000806$, $s_{\overline{X}_1 - \overline{X}_2} = 0.000286$, and $t = 7.28229$, indicating that the spring 2002 yielded a 0.2% higher ratio of erroneous packet retransmissions than the autumn 2001 dataset with 99.9% confidence limits 0.11% and 0.3%.

6.4.5 Connection size

It was considered that connection analysis would be irrelevant for the RYL traces, as the HTTP client used, ***wget***, did not include HTTP v1.1 support. All current web browsers (with Internet Explorer and Netscape Navigator being the typical examples) have HTTP1.1 implemented and enabled by default, therefore they all *should* [Fielding *et al* 1997] use persistent connections to retrieve web pages. However, in order to be able to compare the RYL results, obtained using HTTP 1.0, with any others, resulting from HTTP1.1 browsers, the analysis included the distribution of data object sizes, as retrieved during the two rounds of experiments. The result of the connection size analysis is displayed in Figure 6.13.

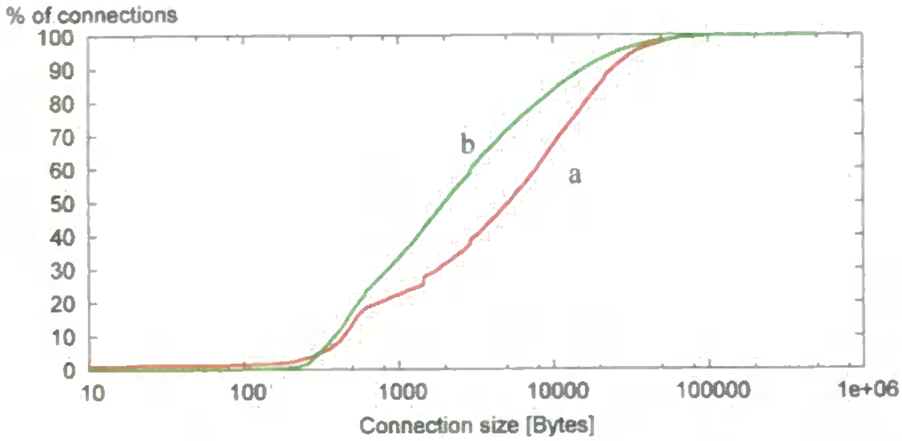


Figure 6.13 - Cumulative distribution of connection size for a) autumn 2001 b) spring 2002

The statistical t-testing was based on the hypothesis that the autumn 2001 dataset produced longer connections than the spring 2002 dataset. The resulting figures, based on this hypothesis, were $s^2 = 2.088 \cdot 10^8$, $s_{\overline{X_1}-\overline{X_2}} = 141.023$, and $t = 25.0261$, confirming the hypothesis by a difference of 2529 bytes, with 99.9% confidence limits 3065 bytes and 3993 bytes.

It is interesting to notice that the autumn 2001 round of experiments had a higher overall average for connection size (9899 bytes/connection) compared to the spring 2002 experiments (6370 bytes/connection). This difference is apparent not only in the average figures, but also in the above distribution. This may be due to the structure of a web page. Aside from the HTML document, the page may contain a number of images, some of them larger than the HTML document (such as pictures) and some of them smaller than the HTML document (such as buttons). The resulting average file size will depend on the proportions of these three types of files when contributing to the page totals.

6.4.6 Congestion window

The congestion window inference includes a high level of assumption in terms of TCP connection analysis. In this case, the task had an increased level of difficulty due to the characteristics of the monitored transfers: unknown senders, receiver-based capture, and no control over the endpoints / transfer. The fact that the TCP implementations of the senders were unknown did not allow any inference with regards to the congestion window evolution profiling. The intention was to produce a rough estimate of the congestion window, not to compete with *tcpanaly* [Paxson 1997b], which includes more complex analysis but also requires traffic capture at / near both endpoints. The receiver-based capture introduces uncertainty with regards to *if*, *when*, and due to *which* acknowledgment has the sender transmitted a data segment.

Due to the variety of window increase policies and the lack of knowledge about which acknowledgments reached the server, the congestion window inference was based exclusively on timing between different trains of packets rather than acknowledgment dialogue (the third method of congestion window estimation from section 3.4.3). The third problem, no control over the endpoints, differentiates the study from Internet measurement efforts such as [NIMI 2003], also expanded in [Paxson 1999]. Within measurement infrastructures, endpoints are running dedicated clients transfer large files between them at regular intervals in order to determine the network characteristics. Within this study, all the senders were remote sites on the Internet and the objects transferred were various web pages residing on the servers; as a result, there was no control over the size / timing of the connections.

As described in Chapter 3, the timing between successive packets may be estimated using several methods. For this trace analysis, the chosen one involved the packet spacing analysis, based on the captured timestamps. The alternative, which was to combine this analysis with TCP options timestamps, was avoided due to reasons relating to the actual TCP implementations. As presented in section6.4.2.1, not all the senders had implemented / enabled TCP timestamp options, and, from the ones that did, the coarse resolution used by some of the senders would have rendered the method unusable. The results for the two rounds of experiments were similar, as reflected below in Figure 6.14 and Figure 6.15, which picture the distribution of the initial and maximum congestion window per connection for the two datasets.

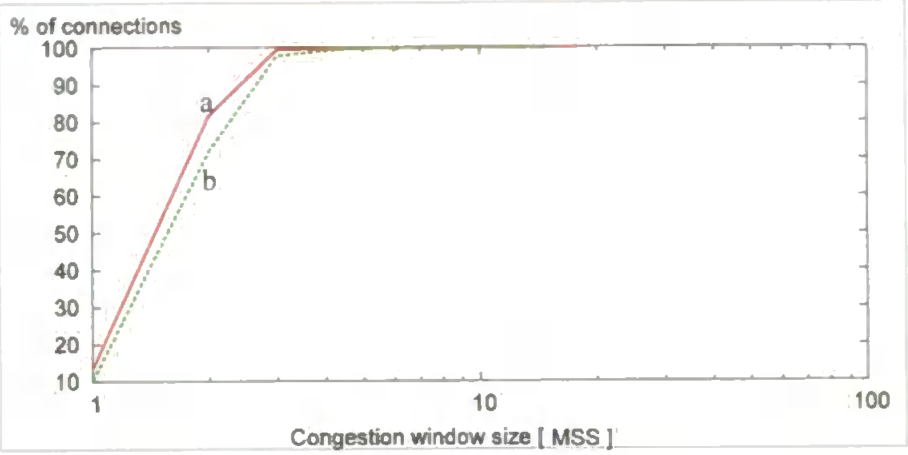


Figure 6.14 - Cumulative distribution of the initial congestion window size for the a) 2001 and b) 2002 datasets

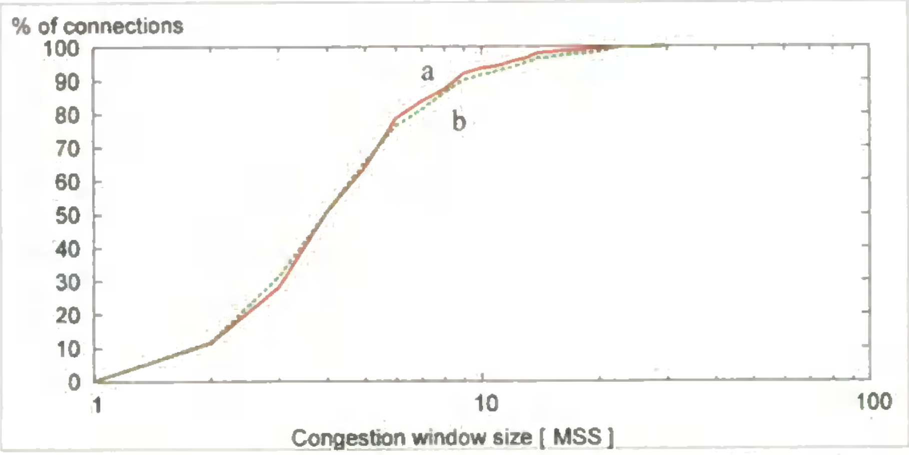


Figure 6.15 - Cumulative distribution of the maximum congestion window size for the a) 2001 and b) 2002 datasets

The average figures for the two variables (initial / maximum congestion window) were 2.07 / 5.32 MSS for 2001 experiments and 2.21 / 5.52 for the 2002 experiments round. The variation is, as shown by the distributions, only marginal, leading to the conclusion that, overall, the characteristics of the data transfers were similar between the two experiment rounds, in spite of the difference in the method used. It is also interesting to observe that the great majority of senders did implement a 2MSS initial window, a fact that is visible in Figure 6.14. This speeds up the start of the TCP connection as it avoids the time-delayed acknowledgment which the receiver will produce after an initial 1MSS window.

Similar statistical t-tests tests were applied for both the initial and the maximum congestion window, using the hypothesis that the spring 2002 dataset produced generated higher values for initial/maximum congestion window in comparison with the autumn 2001 dataset. In the case of the initial congestion window, the analysis produced $s^2 = 0.557532$, $s_{\overline{X}_1 - \overline{X}_2} = 0.011666$, and $t = 11.4324$, which confirmed the hypothesis and indicated a difference of 0.1333 segments between the two datasets, with 99.9% confidence limits 0.095 segments and 0.172 segments. The hypothesis was also confirmed for the maximum congestion window, where $s^2 = 11.778678$, $s_{\overline{X}_1 - \overline{X}_2} = 0.05364$, and $t = 3.60128$; in statistical terms, the t value indicated the same confidence level, 99.9%, an overall difference of 0.193 segments with confidence limits 0.0166 and 0.370.

6.4.7 Throughput

From all the parameters, data throughput should be the easiest one to evaluate: the number of data bytes divided by the elapsed time between the first and the last packet of the connection. The first unclear issue is the elapsed time: which packets should be considered to mark the beginning of the connection? It was considered throughout this study that considering the entire connection time would not present any interest from a throughput perspective. Inclusion of the SYN/FIN packets does not illustrate data efficiency but only combines the data throughput with the time spent to establish / close a connection. This is why the throughput was measured using the interval between the capture of the first and last data segments with non-zero payload.

A second problem is the behaviour of the endpoints throughout the connection. In the case of an HTTP 1.1 persistent connections it is rather difficult to define the connection time, because the client is expected to leave the connection open for further requests / downloads[1]. The time between two such downloads is spent by both endpoints in an idle state, therefore it should not be accounted for when calculating the elapsed time. To support persistent connections, the loss inference mechanism had an upper bound to account for the idle time between retrieving two successive objects and the sum of idle periods was removed from the total elapsed time.

A third problem exists in relation to determining the throughput of a very small data transfers. Connections that carry such objects may have all the data transmitted in the first congestion window, in a single back-to-back train of packets. When this happens, the resulting value for throughput is actually comparable to the value of the bottleneck bandwidth. To exemplify, the traces were split into two-packet connections, three-packet connections and at-least-four-packet connections. The split was based on the results from section 6.4.6, which showed that that the vast majority of the analysed TCP senders did not have an initial congestion window larger than three packets. The results are illustrated in Figure 6.16

[1] Although not an issue for the RYL analysis, it may be a consistent problem for real packet traces

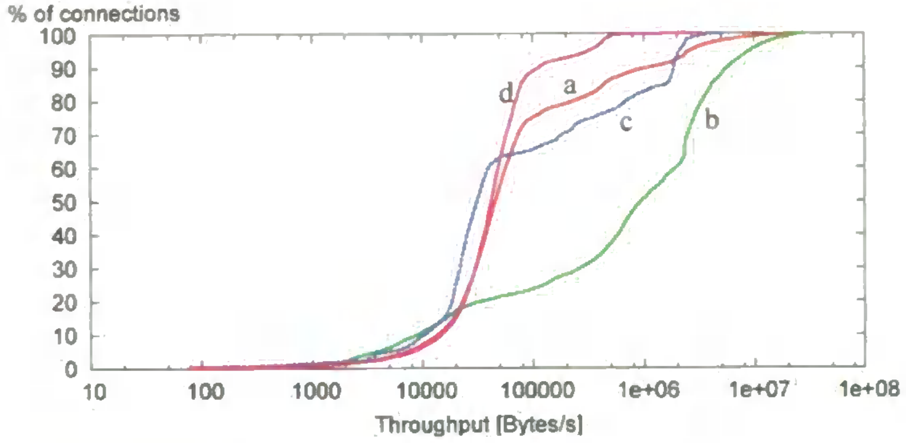


Figure 6.16 - Throughput distribution for the a) autumn 2001 dataset and the subsets of: b) 2-packet c) 3-packet, and d) 4-or-more-packets connections

The distribution reflects the statistics behind the connections: while the throughput has an average value of 501595 B/s for the raw dataset, it reaches an average of $2.32 \cdot 10^6$ B/s for the 2-packet connections, 434972 B/s for the 3-packet connections, and only 69823 B/s for the 4-or-more-packets connections. In fact, the above figure for 2-packet connections is higher than the bandwidth available at the time. This is due to the way timestamps are collected: for each packet, the capturing routines apply a single timestamp; in the case of the *pcap* library, used to collect the traces, the timestamp is applied when the last byte of the packet is captured[1]. As a result, for a 2-packet connection, where the duration of the connection is the time elapsed between the moment when the last byte of the first packet is captured and the moment when the last byte of the second packet is captured; as the two packets are sent back-to-back, this is approximately equal with the time required to receive the second packet by the capturing program. The resulting figure for bandwidth is calculated as the report between the size of the two packets and the time required to receive the second packet, which leads to double the amount of real bandwidth if the two packets are filled with data. To make things worse, in most cases the

[1] This pcap characteristic was considered when implementing the bandwidth estimation algorithm, which considers the size of the newer packet when analysing a pair of data packets inferred to have been transmitted back-to-back.

second packet was not full: the average size of such connections was 1.44 MSS (average connection size - 1728 bytes / average MSS – 1195 bytes). As a result, the elapsed time considered in the calculation is less than half of the actual figure, i.e. for both packets.

After eliminating the 2- and 3-packet connections, Figure 6.17 reveals the throughput distribution for the autumn 2001 and spring 2002 traces.

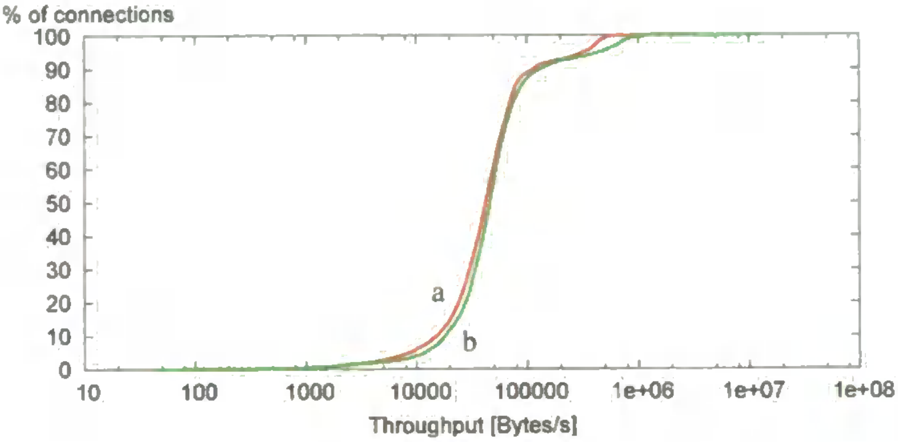


Figure 6.17 - Throughput distribution for the connections with at least 4 data packets for the a) autumn 2001 and b) spring 2002 traces

The two graphs confirm the results from section 6.4.6: the shape of the distribution remained virtually the same between the two experiments. The improvement in download speed, due to the bandwidth increase, was better visible in the average throughput figures: from the above-mentioned average of 69823 B/s, resulting from the 2001 experiments, the 2002 average throughput reached 89986 B/s.

The above average figures were confirmed also by the statistical t-testing. Using the hypothesis that the spring 2002 connections experienced higher throughput than the autumn 2001 connections, the calculations produced $s^2 = 1.81 \cdot 10^{10}$, $s_{\overline{X_1}-\overline{X_2}} = 1997$, and $t = 10.096$. Based on the resulting t value, it was concluded that hypothesis was correct, with a difference between the two datasets of 20163B/s and with 99.9% confidence limits 13591 B/s and 26734 B/s.

It is difficult to say whether the throughput distribution of 4-or-more-packet connections from Figure 6.17 should be considered the *correct* one. The rationale used in this section aimed to identify transfers with genuine throughput, in the sense that the RTT had a role to play during the associated TCP connections. Additional analysis, not included here, on a packet-by-packet basis up to 9 packets showed a similar picture to the one in Figure 6.16, with the 5-, 6-, 7-, 8-, and 9-or-more-packets connections distribution having similar shape and values with the one for 4-or-more-packets. However, a 4-packet connection in the case of a sender with an initial window of 3 packets was only delaying the 4th packet from transmission by one RTT, making this case a particular one as well. Expanding from this, in an ideal world, only connections reaching steady state, e.g. as defined in [Padhye *et al* 1998], should be considered for throughput measurement. But, in the case of the RYL traces, the 2- and 3-packet connections represent a considerable percentage of the connections, accounting for approximately a third of the total number of connections (for autumn 2001, 21% of connections were 2-packet connections and another 13% were 3-packet connections; the spring 2002 set registered even higher figures, with 21% 2-packet connections and 19% 3-packet connections). The only reason for not resolving this dispute here is because it can be argued that the two traces used HTTP 1.0 traffic only and persistent connections introduced by HTTP 1.1 would radically increase the connection size. A similar study was performed in the backbone traces in section 6.5 where, this time, the clients *were* real hosts, running actual web browsers, therefore creating a more credible environment.

6.4.8 Elapsed time

Two time-related variables were extracted from the traces for analysis: the *connection time*, indicating the time elapsed between the moments when the first and the last packet were captured, and the *data transmission time*, which is delimited by the first and the last captured *data* packets. For reasons mentioned in section 6.4.7, only the connections with at least 4 data packets were used for analysis. Figure 6.18 illustrates the distribution of the two variables.

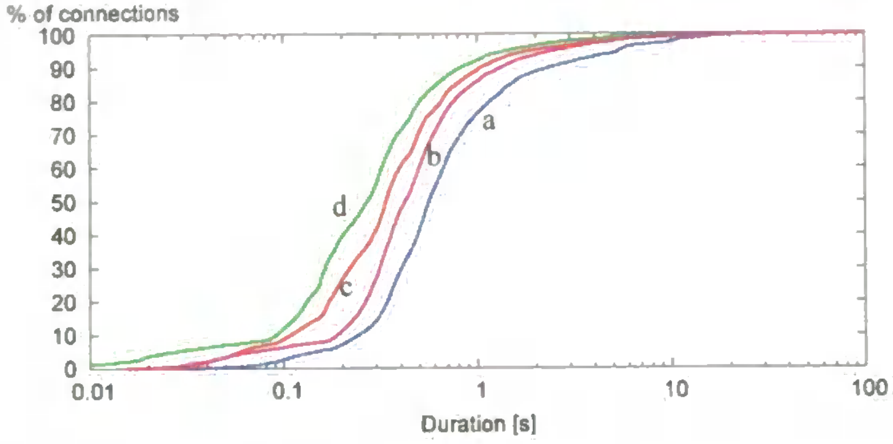


Figure 6.18 – Cumulative distribution of the a)/c) connection time, b)/d) data connection time for the autumn 2001 / spring 2002 traces

Due to the nature of the client (HTTP 1.0), the connections should have been closed as soon as possible after the server finished transmitting the requested object. Still, the difference between the data transmission time and the connection time was rather high for the raw dataset, due mainly to servers that maintained the connection open after transmitting the last data packet.

This behaviour should not be considered illegal, because the HTTP 1.1 specification mentions that a "HTTP/1.1 server MAY assume that a HTTP/1.1 client intends to maintain a persistent connection unless a Connection header including the connection-token *close* was sent in the request" [Fielding *et al* 1997]. ***Wget*** formatted its requests with a *keep-alive* connection token, probably for compatibility reasons with the HTTP 1.0 specification, which also allowed *negotiated persistent connections* but, as admitted in the above-mentioned HTTP 1.1 specification, had only faulty implementations.

6.4.9 Page content

The files retrieved during the 2002 round were saved for offline page analysis. The saving preserved the directory structure in order to identify each web page, but this led to problems when analysing the results because of the redundancy previously mentioned in section 6.4.1. Due to retrieval of multiple pages from the same website, some of the apparently larger web pages were actually several web pages collected from the same website. The saved content required parsing, based on the timestamps of the files, to separate between pages collected in separate retrieval sessions. The resulting distributions are presented below in Figure 6.19

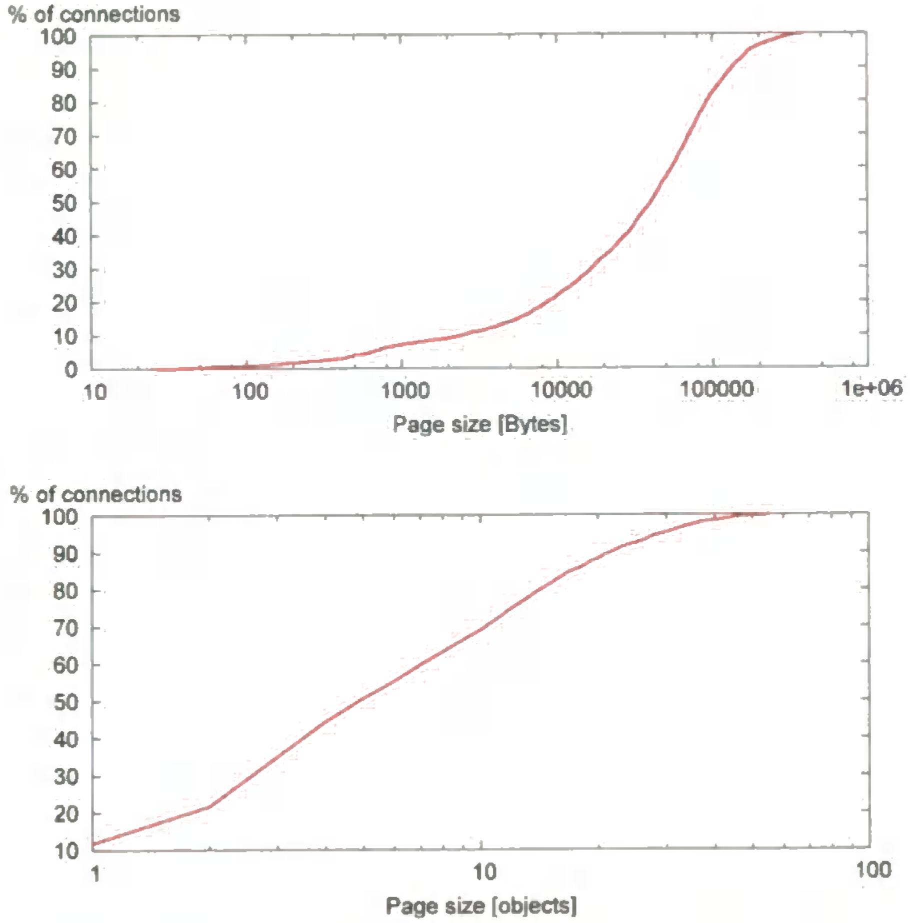


Figure 6.19 - Distribution of page content in (top) bytes / page and (bottom) objects / page

Figure 6.19 shows that a considerable amount of the retrieved web pages had relatively large sizes (previous studies, such as [Paxson 1999], considered 100 KB files to be sufficiently large for demonstrating all aspects of TCP). Also, from the distribution of objects per page, it may be concluded that full usage of HTTP 1.1 request persistent connections would considerably reduce the overall time to retrieve the web page. The average figures for Figure 6.19 are 57506 bytes / page and 9.15 objects / page.

6.5 UoP traces analysis

In the autumn of 2002 an opportunity appeared, thanks to the University of Plymouth (UoP) Computing Services, to collect data from the connectivity point between the UoP network and Janet. This allowed a thorough comparison between the distributions obtained for network paths and TCP connections parameters during the RYL experiments and larger amounts of real traffic data. The results in this section were produced using traces collected from the UoP backbone using port mirroring technique on one of the core switches from the UoP network infrastructure. Five half-hour traces were collected at the end of November 2002, all of them during daytime. The traces were sanitised using tcpurify and analysed offline similarly to the RYL traces, as described in 6.1. The data was filtered also using the same techniques as in the RYL case: IP-only filter to build a subset for path characteristics (RTT and bandwidth) and IP-and-size filter to extract a subset for traffic characteristics (loss, congestion window, and throughput values). The following sections give an overview of the findings, and compare them, where appropriate, with the RYL results.

6.5.1 RTT

The average RTT values led to a distribution similar to the one obtained for the RYL, with a single exception: the number of connections with RTT<100ms (32%, compared with <10% for any of the RYL traces). The RTT average and standard deviation associated graphs may be seen below in Figure 6.20 and Figure 6.21

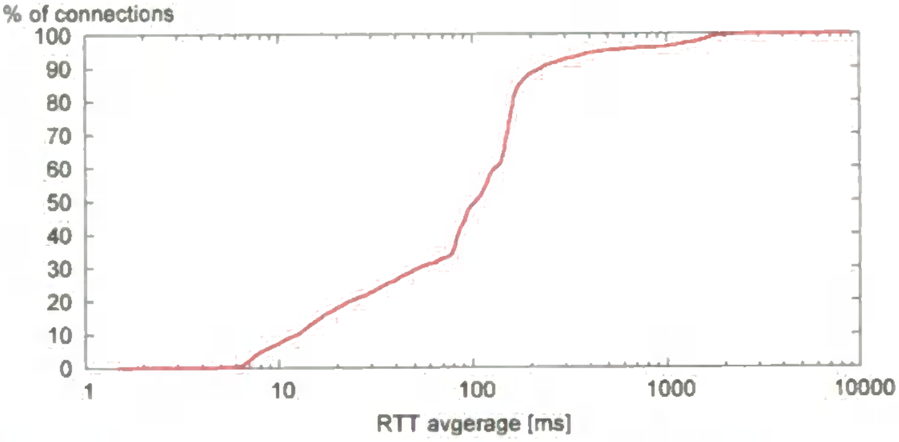


Figure 6.20 - RTT average distribution for the UoP backbone traces

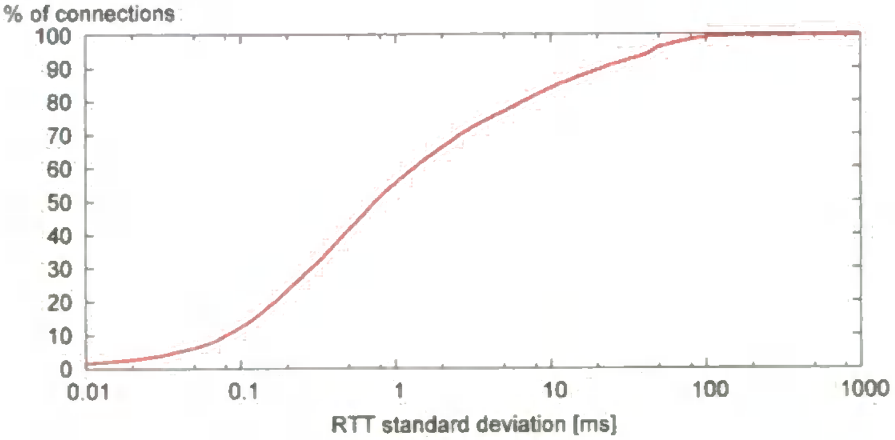


Figure 6.21 - RTT standard deviation distribution for the UoP backbone traces

The average values throughout the two distributions are very close to the ones obtained in the

RYL experiments: 168.85 ms for RTT delay and 22.71 ms for RTT standard deviation. It became of interest to determine what causes this change in the distribution, i.e. what is the 100ms mark that appears in both Figure 6.4 and Figure 6.20 and, further, whether this is likely to appear in other studies. To clarify this issue, a traceroute was run to cnn.com from within the UoP network and from an ADSL host connecting via the ISP Pipex. The results are shown below in Figure 6.22.

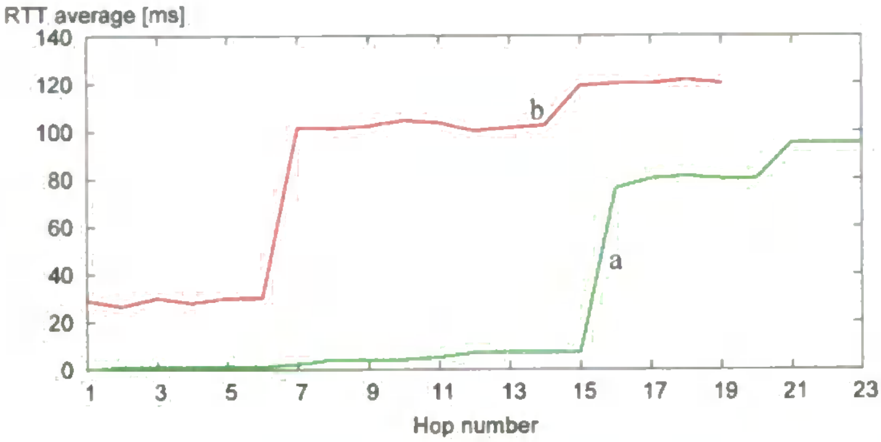


Figure 6.22 - RTT average values for a traceroute to cnn.com using a host from a) UoP network b) Pipex ISP (the initial 30ms difference is due to the ADSL connectivity, compared with the 100Mb/s UoP access)

Both graphs (for UoP and Pipex) exhibit an abrupt increase of approximately 70ms in the RTT. The increase appears in Figure 6.22 between hops 6 and 7 for the Pipex graph and hops 15 and 16 for the UoP path. In both cases, the two nodes are the routers of the corresponding carriers placed in Europe and US (AlterNet for Pipex[1] and Sprintlink for UoP).

The graph for the UoP trace may be therefore split into three regions:

- Hops 1-15, corresponding to the Janet backbone and connection to Sprintlink, which

[1] The names of two AlterNet hosts provided an indication about their location: so-0-1-0.TR2.**LND9**.ALTER.NET (likely to be placed in London) and so-6-0-0.IR1.**NYC12**.ALTER.NET (likely to be placed in New York)

introduce a ~10ms RTT delay.

- Hops 15-16, corresponding to the Sprintlink transatlantic segment, with an associated ~70ms delay
- Hops 17-23, corresponding to US carriers, introducing an additional ~20ms delay

The three figures lead to a total approximate RTT delay of 100ms. It is difficult to generalise from this experiment to the overall results due to other factors that may have influenced the RTT, such as the time of day – the data was collected during day time over several days. However, it is believed that US sites are likely to be located in the >100ms RTT interval (the reverse – all the >100ms sites are located in US - does not apply, because there are likely to be poorly connected sites in Europe as well).

If the assumption presented in this section is true, then it may be concluded that the RYL experiments were biased towards US-based websites. However, as seen in Figure 6.20, even real traces exhibit a large number of connections to sites bearing the same delay characteristics.

The ratio between the number of ack-based and TCP timestamp-based RTT inferences is shown in Figure 6.23. The shape of the distribution is similar to the ones obtained during the RYL experiments, as was the average ratio of the distribution - 0.92.

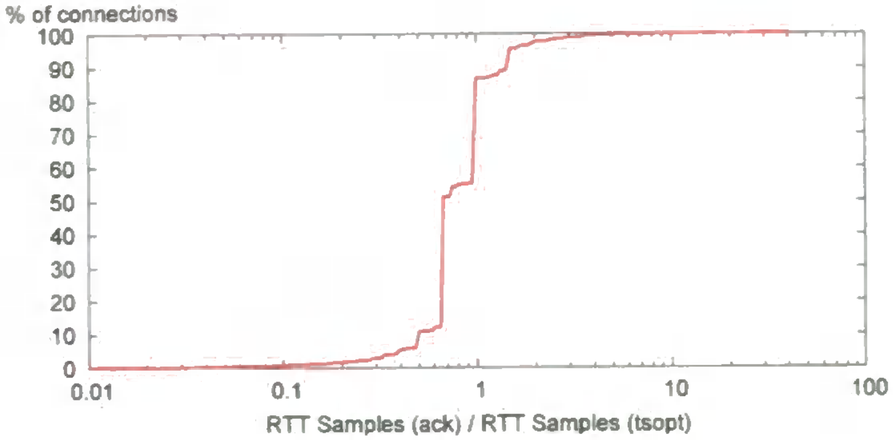


Figure 6.23 - Distribution of RTT samples based on acknowledgments vs. RTT samples based on TCP timestamp for the UoP dataset

A statistic of the TCP timestamp resolution was produced as part of the analysis. Because in the case of the traces both the senders and the receivers varied, the results included two sets of results: one for the hosts acting as web clients (the UoP hosts) and the ones representing the web servers (the websites accessed). The resulting distribution is presented in Figure 6.24

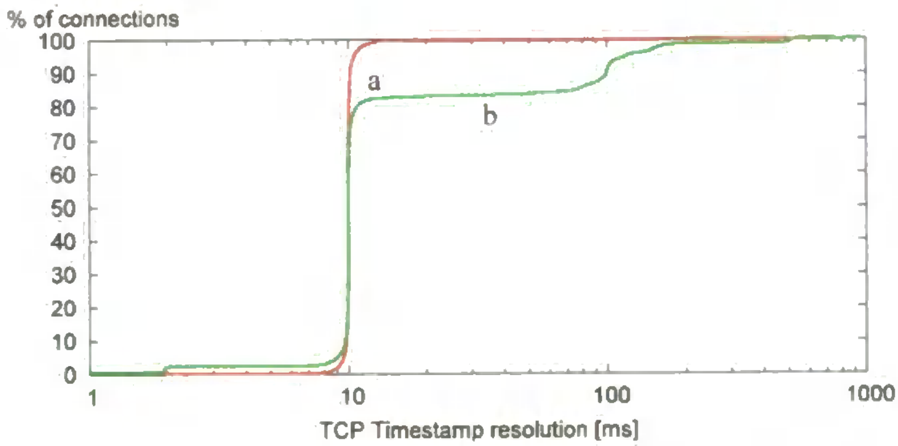


Figure 6.24 - TCP timestamp resolution of the a) TCP clients and b) TCP servers for the UoP dataset

It may be observed that the distribution obtained for TCP timestamp resolution of web servers is similar to the one obtained during the RYL tests, with approximately 80% of hosts using a 10ms resolution, while the rest of the TCP implementation in the rest of the servers uses a >100ms timestamp. The 'default' desktop PCs at UoP were running, at the time, Windows 2000, which does implement TCP timestamps but it does not have them enabled by default [Microsoft 2000]. Customised hosts, typically machines used by researchers, were running, in addition to / instead of Windows, a flavour of Linux (e.g. the Network Research Group connected to the UoP through a NAT host, running Smoothwall - a dedicated Linux-derived firewall).Working by elimination, the web clients which produced the diagram are most likely various flavours of Linux, all using a BSD-derived TCP implementation

6.5.2 Loss

The rate of data from the backbone capture was, as expected, much higher than during the RYL experiments. Due to the hardware used (PentiumII 450MHz machine with off-shelf PCI network interface card and default Linux installation), the logs had multiple reports of network card malfunction during the capture ("too much work at interrupt"). This affected the results of the analysis (such as the number of data packets captured during each connection), but it did not impact strongly on loss figures. This is because lost packets are identified when the retransmission occurs, not when there is a gap in the continuity of data transmitted (this also allows separation between misordering and retransmission); as a result, such gaps are ignored by the analysis and the loss figures are not affected.

The loss figures were higher in comparison with the RYL results. Cumulative distributions of the visible, inferred, and avoidable losses are presented below in Figure 6.25

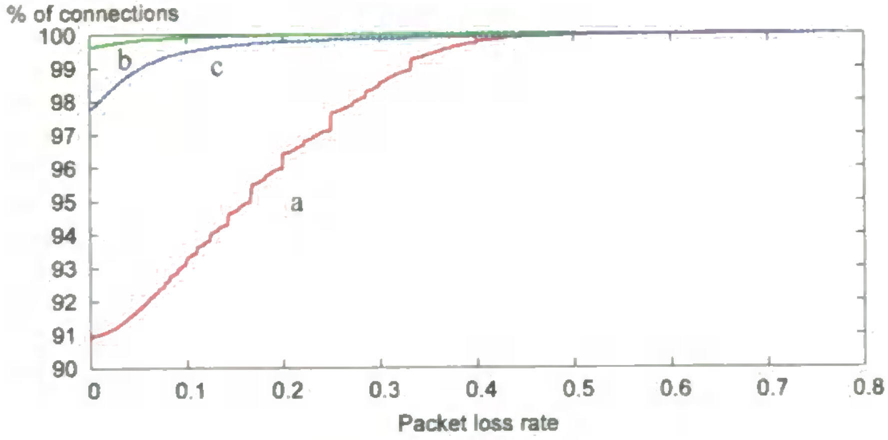


Figure 6.25 - Packet loss rate cumulative distribution for (a) visible, (b) inferred, and (c) avoidable loss events within the UoP dataset

The visible loss rate events appeared more often in the UoP dataset in comparison with the RYL experiments. There were 16305 connections (9.09% of the total) that contained visible loss events. This might be due to the larger transfers that took place – more details about the connection sizes will be presented in section 6.5.4. The inferred loss events were very rare throughout this collection of traces as well; only 636 connections (equivalent to 0.35% of the connections usable for loss analysis) exhibited inferred loss events. A change was also noticed in the distribution of the avoidable retransmissions. Compared to RYL, fewer connections exhibited avoidable retransmissions in the UoP dataset: 4001 connections, equivalent to 2.23% of the total number of connections. This was due probably to the position of the capturing point, i.e. not right at the receiver. Being away from the receiver, the capture might not have seen some of the acknowledgments for outstanding data segments being sent before the arrival of those segments.

6.5.3 Bandwidth

The bandwidth analysis of the UoP traces revealed a very similar picture with the one produced from the RYL traces. A cumulative distribution of the bottleneck bandwidth is presented below in Figure 6.26; the average value obtained was 6.8MB/s

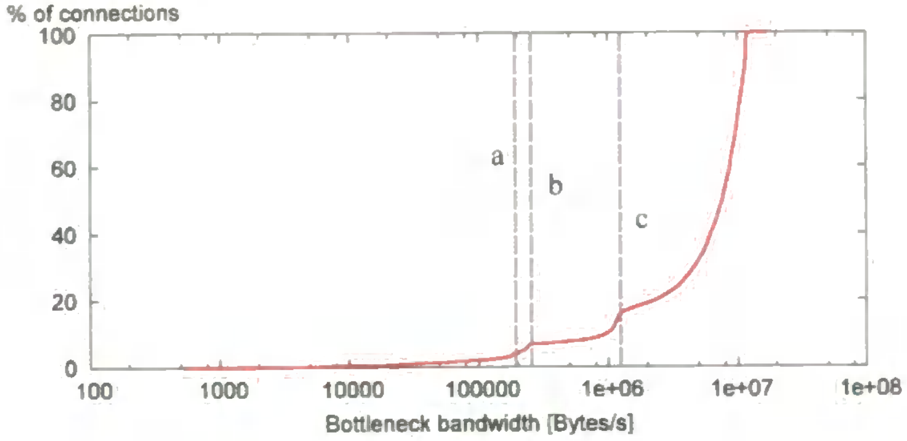


Figure 6.26 - Bottleneck bandwidth cumulative distribution for the UoP dataset. The three grey markers indicate a) the T1 (1.544Mb/s) boundary, b) the E1 (2.048Mb/s), and c) the 10Mb/s boundary

It may be observed that the shape of the graph strongly resembles the distribution from Figure 6.8. The only difference between the two distributions is that the UoP dataset produced an additional change in the slope around the E1 value (marker (b) in Figure 6.26). This is most likely because the UoP dataset included, as explained in section 6.5.1, a larger percentage of connections to sites likely to be geographically positioned in Europe. With this combination of geographical sources, the web sites could have been connected via one of the two middle-speed equivalent choices – E1 (2.048Mb/s) in Europe or T1 (1.544Mb/s) in US.

6.5.4 Connection size

An important part of the UoP traces analysis was the evaluation of connection size. Based on the findings from 6.4.9, it was expected that the distribution and average values of connections size will be higher then the one obtained from the RYL traces. The purpose of the connection length analysis was to determine the average length of HTTP retrievals for the real traffic case. The results of the analysis are presented in Figure 6.27.

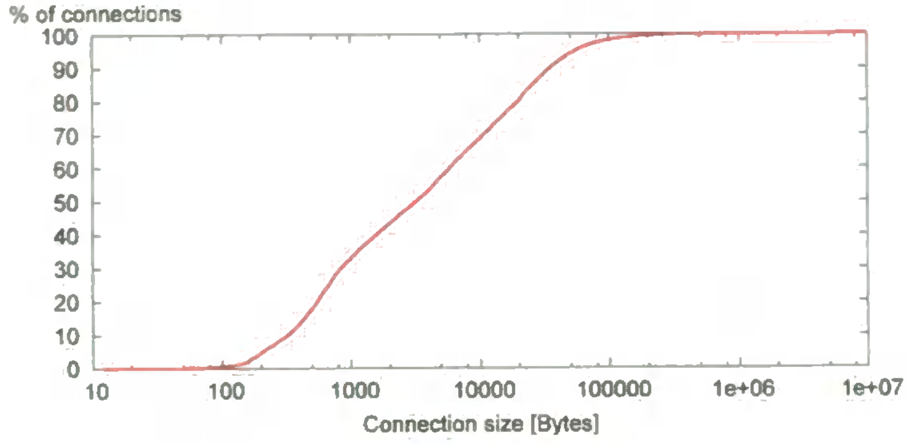


Figure 6.27 - Connection size analysis

The resulting distribution, pictured above, does not match the expected outcome. As described in sections 6.4.9, the average page size is around 50KB, which also indicates the middle of the distribution from Figure 6.19 (top). It was also explained in section 6.4.5 that average connection size within the RYL datasets was under 10KB due to the HTTP 1.0 limitations and it was expected for the UoP traces to come close to the 50KB average of the page size. However, the average behind Figure 6.27 is only 17532 bytes/connection; this is, indeed, higher than the RYL experiments, but still much lower than the page statistics.

This observation is very important from the perspective of connection life span. It does indicate that, even up-to-date web clients, using latest protocols, designed to increase the average connection size, still do not reach steady state, which is the condition for which TCP was created. If web browsing continues to remain the main network application used over the Internet, certain improvements are worth to be added to the TCP specification, such as faster slow-start, in order to increase its performance. Such changes would have to face the current recommendations [Allman *et al* 1999] that suggest strict boundaries for initial congestion window size and congestion window policy. It must be stressed that both sides make justified claims, but also may lead to undesired results. Changes in current TCP algorithms, while improving the evolution of TCP transfers, will alter the self-tuning character of TCP and may lead to higher overall congestion. On the other side, strict recommendations will keep the current balance of Internet transfers, but, as shown in this section and the next one, which discusses congestion window average size, do impact on the performance obtained for current HTTP traffic.

It was not possible to determine whether the clients connecting to the websites disabled the HTTP1.1 option[1] but this is rather unlikely. The most likely conclusions are that either the HTTP persistent connections functionality is not efficiently / correctly used or that the web servers have disabled the feature in order to have better connection management. The first category of issues relates to incorrect implementations and is likely to be eliminated in the future. The second one, however, relates to configuration of web servers: depending on the loading of the server, administrators may choose to disable the persistent connection feature in order to reduce the associated overheads [IBM 2000]

6.5.5 Congestion window

The congestion window analysis produced estimates for 107176 connections within the IP-and-size filtered UoP dataset. The cumulative distribution is presented in Figure 6.28.

[1] This would have required saving the HTTP headers, which may have been seen as breaching the users' privacy

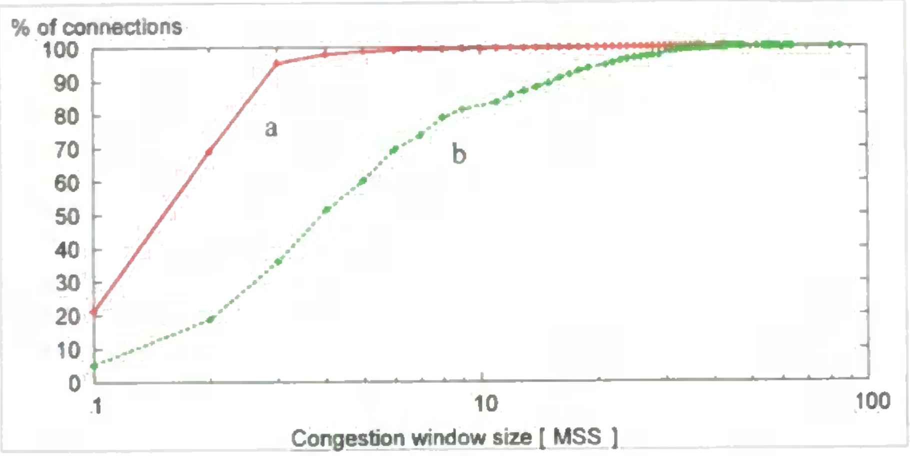


Figure 6.28 – Cumulative distribution of the (a) initial and (b) maximum congestion window size for the UoP traces

The average values were 2.24MSS for the initial congestion window and 6.94MSS for the maximum reached congestion window. It may be observed that almost all connections started with either 1 (21.2%), 2 (47.9%), or 3 segments (26.3%) – the next size, 4 segments, accounted for only 2.7% of the total. The rest of the estimations may be misconfigured web servers but, more likely, they are estimation errors. The problem lies in the criteria used to separate between two trains of packets which is distance between packets vs. average RTT: if the acknowledgment to the first packet is lost, the timeout is 3 seconds which, depending on the arrival sequence, might lead to an erroneous RTT estimate close to this timeout value. As a result, at least for the first few windows, the inter-arrival will be compared with this erroneous RTT rather than the correct value, leading to a large figure for the initial congestion window.

The congestion window analysis also created problems when estimating the maximum window acheived. As described in section 3.4, the method only works as long as the bandwidth*delay product is higher than the size of the current congestion window. For example, using a T1 bottleneck and with the approximate average of 150ms RTT:

$$W_{visible} \leq \frac{RTT}{2} \cdot BW = \frac{1.544 \cdot 10^6}{8} \cdot \frac{0.15}{2} = 14475 Bytes \cong 10MSS \Big|_{MSS=1460Bytes} \qquad 6.3$$

The formula is based on the assumption that paths are symmetric; an assumption which does not always hold, but it is difficult to asses the asymmetry of the two path directions. This is why the RTT is divided by 2, to account for the fact that half of the RTT is the forward path, used for data transmission, while the other half is the reverse one, used for acknowledgments. As a result, windows above 10MSS are more likely to be the result of reaching steady state rather than genuine congestion window growth in case of a T1 connection. However, as shown in the diagram, this was a rare event judging by the fact that 81% of the inferred maximum congestion windows were lower than the above 10MSS and that the estimate provided used a throughput much lower than the average one observed in 6.5.3.

6.5.6 Throughput

The throughput filtering used the same criteria as described in 6.4.7. The 2- and 3-Packet connections were separated from the overall set; the resulting distributions (with and without these connections) are presented in Figure 6.29.

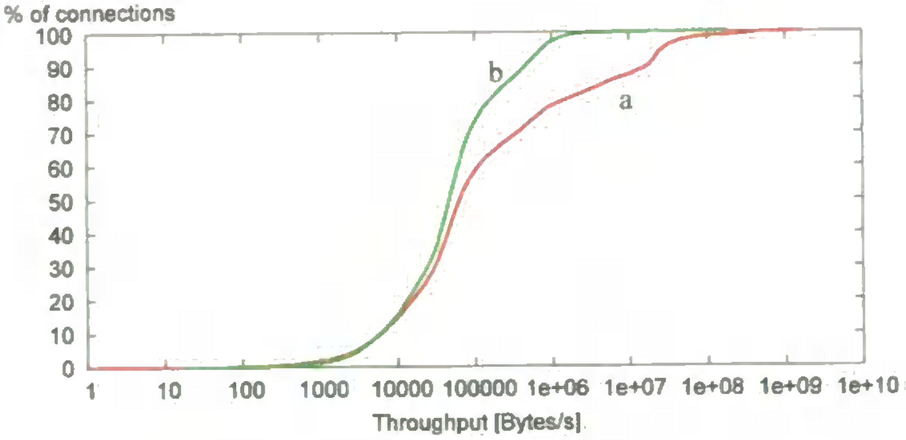


Figure 6.29 - Thput distribution for a) entire UoP dataset and b) only 4+ packets connections from the UoP dataset

The distribution shows the difference between the all-connections dataset and the 4+ Packet connections, a difference reflected in the average values: 7.5 MBytes/s average value for the entire dataset and only 177240 Bytes/s for the 4+ Packets subset. A strange feature of the distribution is visible in the upper part of the graph: several connections from the complete UoP dataset were recorded with a throughput higher than 1GBytes/s. Some of these values resulted due to the effect explained in section 6.4.7: all packets from the connection were sent back-to-back in a single train, with erroneous calculation of the timestamps. However, an additional problem was observed when studying the individual connections, exemplified in Figure 6.30

```
58.171805 client.1600 > server.http: tcp 0 (DF)
58.178275 server.http > client.1600: tcp 0 (DF)
58.178477 client.1600 > server.http: tcp 0 (DF)
58.186164 client.1600 > server.http: tcp 293 (DF)
58.192973 server.http > client.1600: tcp 0 (DF)
58.196612 server.http > client.1600: tcp 1460 (DF)
58.196613 server.http > client.1600: tcp 146 (DF)
58.197066 client.1600 > server.http: tcp 0 (DF)
74.993911 server.http > client.1600: tcp 0 (DF)
74.994104 client.1600 > server.http: tcp 0 (DF)
76.482712 client.1600 > server.http: tcp 0 (DF)
76.489479 server.http > client.1600: tcp 0 (DF)
```

Figure 6.30 - Connection exhibiting a possible timestamp error

It may be seen in the connection that the two data packets sent by the server are only 1µs apart. As the timestamp is applied after the packet was entirely captured [Donnelly 2001], the conclusion would be that the 146 bytes packet arrived at the capturing point in only 1µs. Adding the 54 bytes of headers, the resulting correct bandwidth figure is 200MBytes/s, equivalent to 1.6Gb/s. This is impossible, as the network interface card speed was only 100Mb/s. The error is likely to come from the way timestamp is applied to the packet, a subject that relates closely to the mechanisms between the Linux kernel and ***tcpdump***.

6.5.7 Elapsed time

The last part of the UoP trace analysis was the duration of the connections. Only the 4+ Packet connections were kept, and the analysis looked at the total duration of the connection and the duration of the data transfer. The results are displayed in Figure 6.31

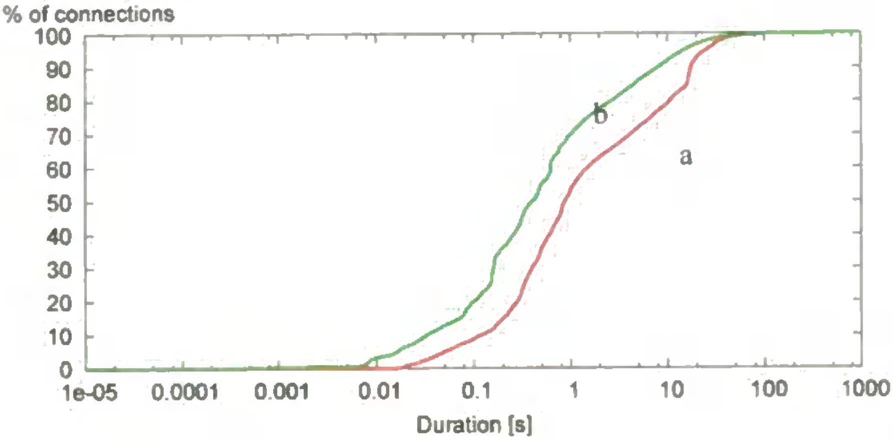


Figure 6.31 – Cumulative distribution of the a) total and b) data transfer only duration of connections from the UoP dataset

The average figures throughout the dataset for the two variables were 3.19s for data transfers and 6.07s for full duration of the connection. This does not imply that half of the time of transfers is spent in establishing and closing the connection because clients may be keeping the connection open to avoid establishing a new one in case other requests are coming.

6.5.8 Issues and limitations

One disadvantage of the study was probably the difference in the sources that produced the three datasets. Although the two rounds of RYL experiments were grouped, the retrieval process was different between them. Due to the limitations of ***wget*** at the time, the first round allowed retrieval of only HTML objects, without the additional objects (e.g. images), while the program evolved until the second round and allowed full retrieval of pages. Still, none of the ***wget*** versions included a HTTP v1.1 tool, which would allow persistent connections and, implicitly, longer TCP transfers. In comparison, the UoP dataset offered the image of the real traffic, but, due to privacy implications, did not allow an in-depth study of the HTTP retrievals.

A limitation of the study was also the lack of diversity in the studied environment. All traces were collected from within the University of Plymouth and, besides a network upgrade that happened between the two rounds of RYL experiments, the connectivity remained the same throughout the datasets. This limited the network paths results in the sense that, rather than analysing a mesh of independent paths, the connections happened between one fixed point, connected via a single technology, and a variety of remote endpoints. However, this may also be seen as an advantage because, with the exception of the first round of RYL experiments, the connectivity characteristics did not appear to affect the results of the measurement (e.g. a low access bandwidth would have limited the visible spectrum of bottleneck bandwidth).

6.6 Summary

This chapter presented the findings resulting from the analysis of three sets of network traces, collected from the same end network, the University of Plymouth. The traces were captured using *tcpdump* and analysed with an implementation of the method described in Chapter 3. The traffic for the first two sets of traces, grouped under the term "RYL experiments" was produced in autumn 2001 and spring 2002 by connecting to random web pages. The third set, referred to as "UoP dataset", was traffic produced by real clients and was collected during autumn 2002 from a UoP backbone node.

The analysis attempted to profile three main issues: the Internet paths characteristics, the evolution of TCP connections, and the web page transfer features. The paths characteristics were defined by the round trip delay (RTT) and bottleneck bandwidth encountered by the analysed flows. The analysis also looked at packet loss; it was shown that, although related to the path characteristics, this variable had strong links to the transfer features. The evolution of TCP connection focused on its two boundaries, the sizes of the initial congestion window and of the maximum congestion window. Finally, the analysed transfer features were the size of web pages and the size / efficiency of TCP connections / HTTP retrievals. Throughout the analysis of the UoP dataset, the results between the two collection methods were compared to determine whether the findings from the artificial RYL experiments were similar with the results from the real traffic. The RYL experiments were also analysed in terms of redundancy and variability.

In terms of Internet paths characteristics, the two datasets have shown a very similar image: fairly low delay, proportionally low standard deviation, and high bandwidth paths. The RTT average distribution was dominated by a strong increase around the 100ms delay value. It was shown that one of the main possible causes for this increase could have been the large number of US-based websites, all incurring a 70-100ms transatlantic and US-continental delay. The analysis also compared the resolution of ack-inferred vs. TCP options timestamp inferred RTT averages. It was shown that for most flows (and on average) the TCP options timestamp offers a higher number of estimates, in spite of the unfavourable position of the capturing device in all the cases. However, the TCP options timestamp analysis showed that a considerable number of websites still used a clock with a 100ms resolution. With RTT averages being of comparable values, this diminishes the benefits provided by TCP timestamp usage.

The bandwidth analysis appeared to be accurate in the vast majority of cases, being able to reveal several features of the analysed environment. The analysis of the RYL experiments was able to identify the tenfold network bandwidth upgrade, an event that happened between the two rounds, revealing a bottleneck distribution asymptotic to the local access speed. Also, the analysis of the UoP dataset revealed three peaks corresponding to typical connectivity solutions (E1, T1, and 10Mb LAN).

The loss analysis focused on three types of loss-related events: visible losses, inferred losses, and avoidable losses. Overall, the image offered by the traces was a loss-free environment for the vast majority of the connections, with a considerable number of avoidable losses and very few genuine timeout events.

The results for web page size, TCP transfer size, and congestion analysis were closely related. The RYL spring 2002 analysis showed that web pages were fairly large in terms of total size, but they also included on average a large number of objects to account for that size resulting in typically short connection sizes. The congestion window analysis has shown that almost all senders were using 2 MSS or 3 MSS initial congestion windows, allowing faster transfers by eliminating the first delayed acknowledgment.

The analysis of the transfer speeds focused on throughput, with a brief discussion about connection duration for the UoP dataset. The results were similar for all traces, with low average throughput figures, due to the low average values of maximum congestion window transfers which did not allow for proper usage of bandwidth.

Overall, the results obtained from the RYL experiments scaled well to the backbone study, in spite of the various limitations. This indicates that such short studies may be used to provide detailed information about Internet paths and typical web pages. The results of both types of traces have shown, through the UoP Internet perspective, good path connectivity in terms of bandwidth, delay, and loss.

In conclusion, the proposed analysis method was successfully used to produce an image of Internet paths parameters, as observed from a single collection point. The study was performed on packet traces collected from a single connectivity point and focused exclusively on HTTP object retrievals. Based on the resulting Internet characteristics, fast and virtually loss-free paths, coupled with transferred objects features (short-lived TCP connections) several discussions and recommendations were produced.

In the context of this study, this chapter represented the final stage of monitoring – applying proposed TCP analysis on real traffic in order to observe current network conditions. The next step is to try and use the information obtained in order to build the relationship between network and endpoint conditions on one side and the resulting performance on the other side. The study of this relationship will be attempted next, with Chapter 7 proposing a novel approach for tackling the TCP performance modelling, then, throughout Chapter 8, benchmarking the proposed approach using various sources of traffic.

Chapter 7. TCP performance prediction model based on IDA

7.1 Introduction

This chapter presents the top level of the research undertaken as part of this programme. The first stage aimed to identify the state of the art and the associated problems within the performance analysis area, with a particular focus on monitoring and modelling. The second stage proposed a novel approach to monitoring, based on elements of existing techniques. The proposed method used non-intrusive analysis on single-point captured traffic to provide timely results using online monitoring. As part of this traffic analysis stage, the method was applied to semi-controlled and real traces in order to build a holistic image of the Internet.

This last stage is concerned with modelling the performance of TCP transfers. The proposed model aims to overcome the lack of robustness from current mathematical approaches but also to produce a higher accuracy when predicting performance. This first section of this chapter will highlight the limitations of current methods in terms of both robustness and accuracy. The second section will then introduce a knowledge-based approach that aims to provide better results to performance modelling. The introduction of the tool will link the limitations of a mathematical approach with the capability of knowledge-based techniques to learn from past examples and will present previous successes of such methods in the area of networking.

7.2 Limitations of existing mathematical models

Section 2.5 presented the attempts made to date to produce a comprehensive and robust model that describes the evolution and, implicitly, the performance of a TCP connection. In order to fit their proposed models to pseudo-real clients / environments, the theories described are based on several simplifications:

1. The clients have all the same behaviour / use the same implementation (e.g. TCPReno), either of them being fully known

This assumption provides a precise evolution of the connection, due to the predefined characteristics of the clients. Unfortunately, this assumption does not apply to real-world cases, where the TCP senders belong to different implementations, running under various operating systems. Because of this, such models are bound to fail when confronted with a different implementation. The task of determining the implementation itself is cumbersome, as shown in [Paxson 1997b], but necessary, if an accurate model is desired. In regards to the differences between various implementations, in [Popescu and Shankar, 1999] the authors, after profiling 3 TCP implementations, all allegedly based on TCPReno (NetBSD 1.2, Windows 4.0 SP3, and SunOS 5.5), remarked "the differences between these [the studied] profiles are so large that we wonder whether the overall performance in the Internet can be improved merely by just implementing TCP more carefully"

2. The test environments are either controlled or semi-controlled.

This assumption relates somewhat to the first one. The validation data for the currently proposed models comes mainly from two sources: synthetic connections and controlled connections. The first category encompasses connections produced typically using *NS* (Network Simulator) [NS 2003], the network simulator described and used as part of the validation in Chapter 5. The second category is represented by connections going through real networks, but under controlled conditions: both of the ends are known and the amount of data transferred is predefined and typically large. A good example for this type of experiments is series run in [Padhye *et al* 1998] to validate the proposed model.

Both of the above mentioned solutions highlight different aspects of the real network, but are they sufficient? A good start of the discussion regarding the appropriateness of *NS*, or, in general, simulation environments, as a reliable TCP investigation tool is made in [Allman and Falk, 1999]. As the scope of the study is to propose a methodology for analysing the performance of real TCP, it highlights all the pluses and minuses of simulation. It is true, also according to the above-mentioned study, that *NS* proved to be the favourite environment for researchers to test their theories as it offers a wide range of scenarios. Due to its synthetic character, it may reproduce or generate any network conditions or generate environments which would be very expensive, money- and time-wise, to obtain in reality. Nevertheless, aside from all these advantages, it is *just* a simulator. It offers flexibility, as the parameters of the endpoints can be modified as desired, but its behaviour differs from the real case in several ways:

- The one-way TCP clients available under NS are based on the Tahoe, Reno, NewReno, or TCP Vegas specification; a better choice, especially when trying to reproduce real transfers, is the two-way (FullTCP) TCP client from NS, which is based on NewReno [NS 2003]. All these clients are fully compliant with the standards; there is nothing wrong with this, from the point of view of testing TCP implementations but, as observed in [Popescu and Shankar, 1999] or [Floyd and Padhye 2001] the *real* clients vary their behaviour from operating system to operating system, and even from version to version, aiming towards pure specifications (like TCPReno), but never converging. In fact, [Paxson *et al* 1999] even standardised the errors likely to appear in implementations, while [Floyd and Fall, 1999] warned about the negative effects that such implementations may have;
- Endpoints without processing delays. The endpoints lack processing delays due to e.g. high load, which eliminates some of the timing variations;
- Limited environment. The topologies generated with ns are basically limited only by the

software constraints. Within this project, topologies with up to hundreds of hosts were created, in order to provide a near-reality effect. This is far better than configurations or topologies that can be built using real testbeds. Nevertheless, as argued in [Paxson 1997d], the variety of conditions on the Internet and their variations can rise above any type of simulation. On top of this, the limited topologies reflect also in the traffic load: what is the *correct* size of the routers buffers for a *good* replication of the reality: small buffers to produce high losses, ruining the overall performance; large buffers that never fill, therefore there is no loss in the simulated network, but the delay is unrealistically high.

Having discussed the above, it is very convenient to validate the TCP model, built on the TCP Reno specification on a simulation environment that runs exactly the same implementation. The remark probably sounds a bit harsh; the aim is not to suggest that the authors biased the validation conditions to be in agreement with their proposed models, but to highlight their limitations.

To overcome the limitations due to simulation environments, recent models, such as [Padhye *et al* 1998], included a certain degree of validation in semi-controlled environments: a number of computers, running various operating systems, exchanged large files between them. In this case, while the degree of realism is high in regards to the network, the experiments were rather far from reality in regards of the dimension of the transferred files. As was expanded in Chapter 6, the average dimension of the files transferred within HTTP transactions is around 10-20KB, which is several degrees smaller than the transferred data within these validation experiments (each TCP connection lasting for 100 seconds). The reason behind this is the actual purpose of the models. As explained in Chapter 2, the declared aim of the above-mentioned study, as well as [Ott *et al* 1996] was not to determine the overall performance of the download, but to model the stationary behaviour of the congestion avoidance algorithm. Nevertheless, they are currently the landmark papers for defining TCP performance based on the network parameters. Later on, with the extensions brought by his study [Cardwell *et al* 2000], Cardwell attempted the closest experiment to uncontrolled environments: connections to a mixture of popular websites and random web pages. Unfortunately, it is precisely the results of these experiments that he did not detail in his paper, for reasons which are unknown.

Summarising the above observations, there is a clear tendency of the mathematical models to limit to the theoretical / simulated situations. The reasons behind this are multiple; they relate to the limitations of the models and they were discussed in some detail in this section, as well as in the previous one:

1. The models are less aimed at performance, but at congestion window evolution in time, in particular in the stationary stage.
2. The behaviour modelled is not likely to appear in real traffic conditions.
3. The models are heavily relying on knowledge of the TCP behaviour of the endpoints; in fact they use an idealised TCP implementation (the TCPReno specification) as a starting point for the models.

7.3 Why use IDA?

This project aims to overcome the TCP modelling limitations listed in previous section by proposing a novel prediction model, based on intelligent data analysis (IDA) techniques. The approach used aims to bridge the relationship between network and transfer conditions on one side and the resulting performance on the other. After judging the features that range beyond mathematical models, such as unknown client behaviour and uncertain network events, this approach was built on Intelligent Data Analysis (IDA). IDA, also known as Knowledge Discovery in Databases (KDD) [Fayyad *et al*, 1996], is a process of extracting features from raw data, encompassing techniques for preparing, transforming the data, extracting the features then analysing them.

Current models all use a *theory-to-reality* approach to define the TCP behaviour. This way of thinking proved beneficial in terms of understanding and following the TCP evolution in time. One of the successful outputs resulted from the comparison between real clients behaviour and the idealised/modelled behaviour, which allowed possible errors / problems in the implementation details to be identified. They also led to proposing improvements, such as emerging TCP improvements or new TCP algorithms and implementations, all aiming to be *friendlier* to the other traffic and, implicitly, networking environment (i.e. less aggressive, more lenient, *optimum* for the traffic conditions). But, behind all these, remained the reality: a wide range of unknown TCP clients, short transfers, and Internet users interested in the throughput values resulting from these conditions. All these were less of a concern for the conclusions of the existing models. This is why, within this project, the proposed approach takes a different view of performance when compared to the mathematical based models:

Analyse the performance of a comprehensive range of current traffic from real networks and, based on this knowledge, predict the performance of TCP traffic for any combination of network conditions within the known scope.

7.3.1 The structure of IDA

This section describes the typical stages of an IDA procedure. The purpose of this description is to have a generic view of the process, a view that will be customised for the requirements of this research programme in the next section. The model used is the one proposed in [Fayyad *et al*, 1996], displayed below in Figure 7.1.

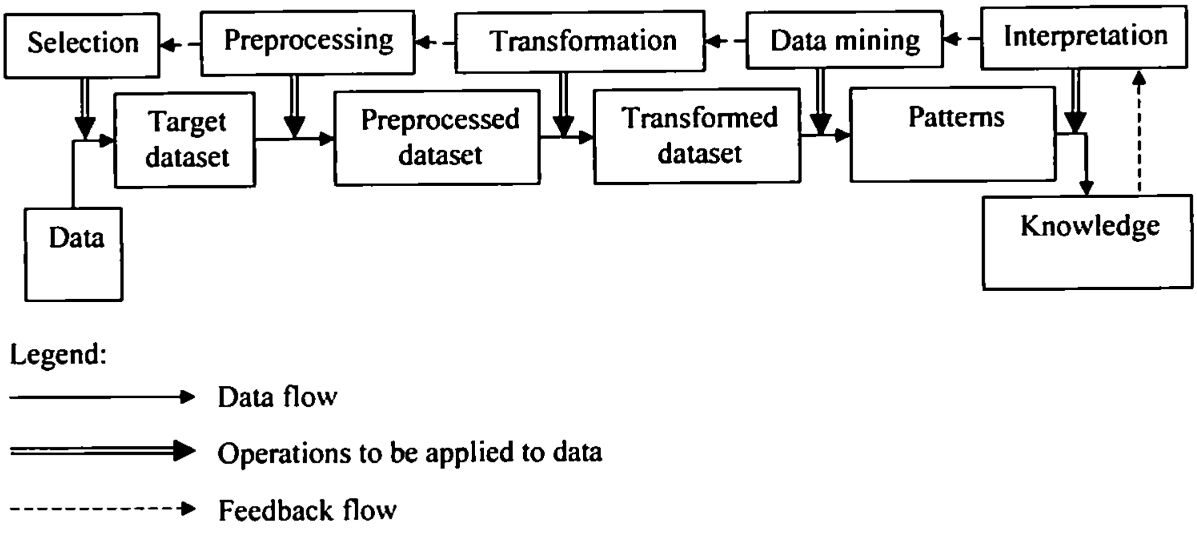


Figure 7.1 - IDA processing diagram – basic representation (adapted from [Fayyad et al, 1996])

The IDA process, as described in [Fayyad *et al*, 1996], includes 9 stages:

1. Build prior knowledge of the domain to study
2. Produce a target (raw) dataset that will be used as the basis for analysis
3. Pre-process raw data: remove noise
4. Project data onto the problem – identify the relevant variables
5. Identify the actual process, such as classification, regression, clustering
6. Identify the optimum analysis to use – determine the best combinations of analysis methods and input/output parameters
7. Apply the data mining itself – perform the pattern/knowledge discovery
8. Interpret the knowledge.

The process follows the successive transformations applied in order to extract useful knowledge from raw information. Typically, from all 8 steps, the focus falls on the middle stages (5-7), with less attention to the pre-processing and post-processing stages. However, as will be shown in section 7.5.1, preparing data for the analysis represents a considerable part of the process, as applied to this study.

7.3.2 IDA for this project

The IDA concept uses a very generic approach to data analysis. It aims to analyse raw data and to extract patterns and knowledge with a degree of novelty and/or *interestingness* [Silberschatz and Tuzhilin 1995]. The applications of the process vary from classification of data into known categories to clustering data into previously unknown categories. In this case the IDA model was adapted for the needs of this study, resulting in the schematic diagram from Figure 7.2. The meaning of the processing blocks from the diagram below will be detailed in the reminder of this section.

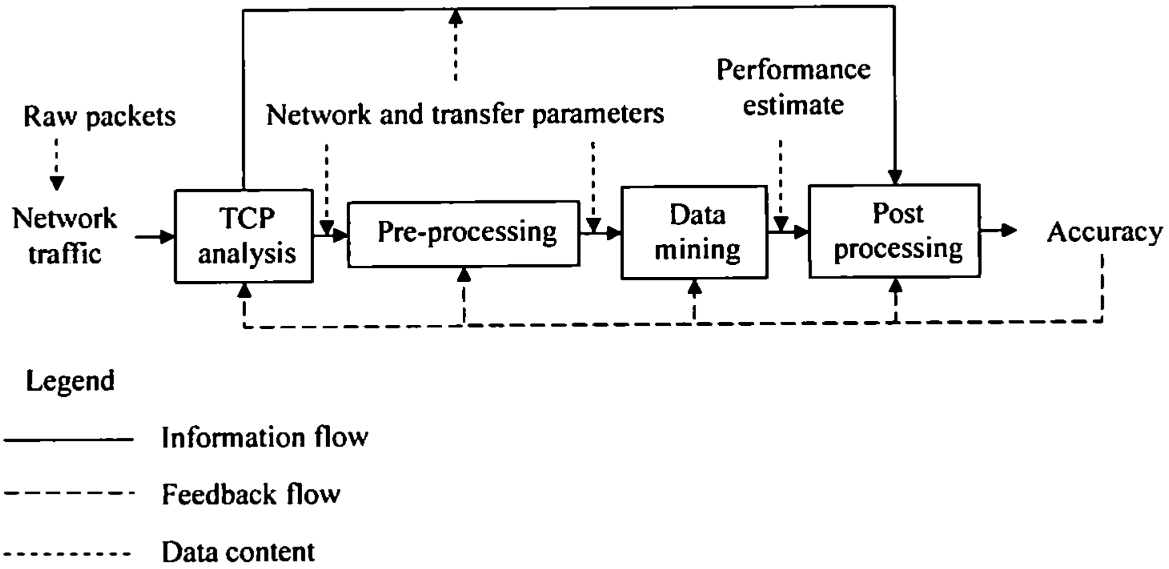


Figure 7.2 - IDA processing diagram – basic representation

The input to the process is the *Network traffic*, representing the raw data captured from the headers of the packets. The capturing can be done for either live processing (where data is fed to the IDA right away) or offline processing – the content of the packets which is stored in a file and analysed at a later time by the IDA.

The raw header content is unusable for data mining. The next step is to acquire knowledge and transform data in a meaningful format, using *TCP analysis.* This was achieved through the research associated with network monitoring, Chapter 3, which provided the mechanisms to understand and interpret the data exchange within TCP connections (section 3.4). The process also requires targeting the relevant data and variables: having determined the domain knowledge (TCP behaviour), it must be decided what data would be useful to use in the analysis. At this step, described in section 2.5.2, the actual variables that will constitute the input of the analysis have to be chosen.

The next step, *Pre-processing*, is essential for the performance of the entire algorithm: the parameters obtained from the raw input during the previous step must be put in a form suitable for the data-mining algorithm. It is at this stage where data may be transformed, filtered, scaled in order to fit the chosen algorithm. Due to the various steps involved in this task, details about the actual processing are provided separately in section 7.5.1.

Data mining is, aside from highlighting the importance of data pre- and post processing, the core of the IDA process. The aim of this stage is to determine a relationship between different instances of the input parameters and the variable that is predicted and classified. The methods used within this project are detailed in section 7.5.2. The process includes two sub-stages: training and testing. During the training phase, the IDA does not generate any output, but only produces a set of rules / a function to map the TCP behaviour onto resulting performance based on the (filtered) *Network and transfer parameters* samples that it is presented with. Following training, in the test phase the data mining engine is presented with unseen samples. The set of rules and functions established in the training phase is then used to provide at the output an estimated value, the *Performance estimate. Network and transfer parameters* represents an instance of the set of variables that define a connection (network conditions, endpoint types, and file size). They are extracted using the proposed monitoring method from a network trace, in the same way as it happens during the training phase.

The output of the data mining algorithm may require further processing in order to interpret its significance. It is the combined task of the data mining output and the *Post processing* block to perform this further analysis in order to evaluate the success of the method.

As shown in Figure 7.2, the process has, aside from the forward flow of data, a feedback flow, where the resulting accuracy is used as an indicator for the efficiency of each step involved. This reverse flow allows identification of sources of error and, if possible, remedies them. It is worth noting that this is a logical flow and requires external interaction.

7.4 Why use a neural networks approach?

As explained in the previous section, the prediction model proposed as part of this study aims to approach the relationship between the network and connection characteristics and the resulting connection duration from a different perspective: predict the performance based on prior knowledge, rather than attempt to model the evolution of the connection. The rationale behind this change is supported by the variety of uncertainty sources when studying the TCP data transfer: the variations between the mathematical model and the existing implementations, the differences between actual network parameters and their values, as inferred by the TCP endpoints, or the preferred steady-state behaviour of the model versus the typical short-lived real connections.

Due to the nature of this approach and the uncertainty sources, the model is required to **learn** the relationship between TCP performance and its influencing factors. The closest concept to match this task is the artificial neural network, which is designed specifically to acquire knowledge from the studied environment via a learning process and to store this knowledge in the interneuron connection strengths [Aleksander and Morton 1990].

The neural network consists of a number of *neurons*, connected via synaptic weights in a structured manner. The model of a neuron, as presented in [Haykin 1998], is shown in Figure 7.3:

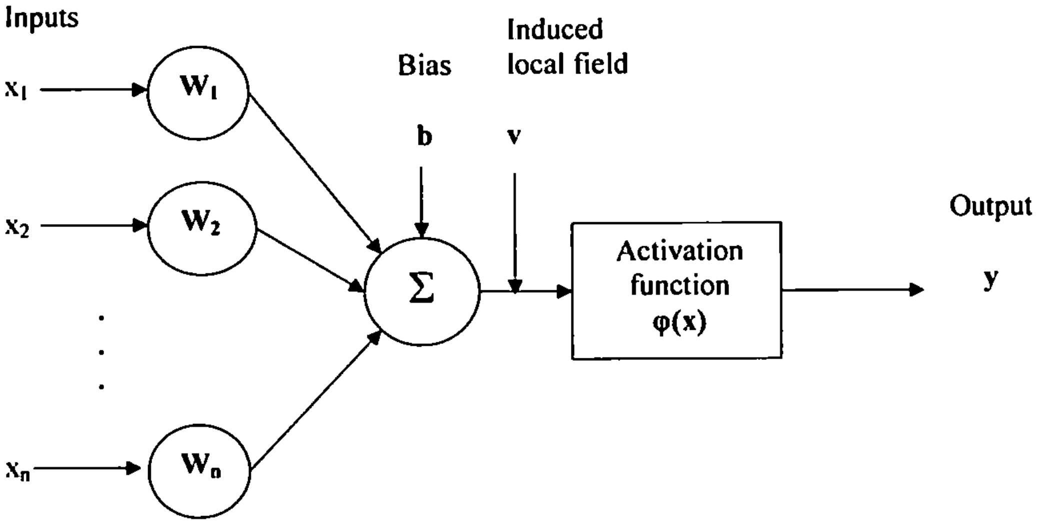


Figure 7.3 The model of a neuron (based on [Haykin 1998])

The neuron has n of inputs, $x_i, i = \overline{1,n}$, added into a weighted sum - v, the *induced local field* of the neuron:

$$v = \sum_{i=1}^{n} w_i x_i + b \qquad 7.1$$

The role of the bias is to adjust the input of the activation function. It may be modelled through an input v_0 that is added to the sum via the weight w_0=b.

The induced local field is then applied to an activation function, which varies from the threshold (Heaviside) function, employed in neural networks used for binary decisions, to the sigmoid function:

$$\varphi(v) = \frac{1}{1+e^{-av}} \qquad 7.2$$

where a is the *slope* of the function. The sigmoid function is preferred as activation function particularly when the neural network is required to map onto a continuous domain.

Neural networks approaches have been extensively used in recent years in the networking area. A few examples of successful applications were obtained particularly regarding network errors classification and decision making. Since early 1990s, [Hiramatsu 1990], [Cheng and Chang 1996], [Catania *et al* 1996], until recently [Ramaswamy and Gburzynski 1999], neural networks were proposed as an alternative to improve QoS control in ATM networks. The studies, although performed in slightly different area of networking, have shown that a knowledge-based approach may lead to better results when faced with the variable nature of traffic, in comparison with statistical approaches.

Based on the above-mentioned studies, neural network-based solutions are likely to overcome typical limitations of the mathematical approaches when dealing with complex relationships between the influencing variables. Within this project, it is expected that the neural network based TCP model will provide better accuracy than the existing mathematical approach, and also, due to its learning capability, will perform better in terms of robustness and flexibility. Robustness is probably the weakest point of all the previous models: they are only able to cope with certain types of TCP endpoints / file sizes / traffic conditions. An ideal model should be able to cope with *any* combination of these characteristics. The flexibility will prove to be advantageous when the model will be applied on *future* implementations of TCP.

It is virtually impossible, due to the diversity of possible combinations, to address these two issues within a model by using a mathematical approach. To build a TCP model using mathematical reasoning would require some sort of consistency throughout the range of TCP endpoints and network conditions, consistency which at least currently does not exist. This is, in fact, the fundamental distinction between mathematical models and the performance prediction model proposed in this project: use prior knowledge to predict overall performance instead of trying to determine the evolution of the TCP connection.

It must be stressed that the primary aim of the proposed model is not to compete with the mathematical models, but to cover a different area of TCP performance analysis by using a different approach. The proposed solution does not provide any information about the evolution of a connection in time, but only about the overall performance (in terms of connection duration) that is likely to be achieved, based on the network and endpoint conditions.

The next section will detail the Pre-processing, Data Mining, and Post processing blocks of the IDA process, as applied within this research programme.

7.5 Applying IDA to TCP performance modelling

7.5.1 Data collection and pre-processing

The first two stages of the IDA process are essential for the success of the IDA approach. Previous sections already described the various sources and methodologies for obtaining the raw data, ranging from synthetic to real network traces, to use for training / testing the proposed model. This section will focus on how this data is pre-processed before entering in the prediction model.

The first issue is what data should be fed into the data mining algorithm. From the studies that developed mathematical TCP models (see section 2.5) and the overall behaviour of TCP clients, it is clear that network parameters, as seen by the endpoint, have a vital impact on the performance of the transfer. In the case of this study, the data collection provides a network trace, which is only a list of the captured packets with their headers; it does not provide any information about the network parameters or status. This is where the TCP analyser, described in Chapter 3, comes into place to interpret the connections within the raw trace and infer the network conditions that were in place during the time of the transfer. The output of the TCP analysis includes the average values for delay (round trip time) and loss (probability, measured in lost data packets per total number of packets transmitted).

A second category of parameters that affect the throughput are connection parameters: the initial and maximum values for congestion window and the file size. Their usage as inputs may be seen as a limiting factor for the robustness of the model (aimed to predict performance only on network parameters). However, due to the major impact they have on the connection evolution, it is impossible to obtain an accurate prediction without including them in the analysis.

Finally, a third set of parameters completes the picture: aside from the network status, which models the behaviour of TCP, and the amount of information transmitted, which allows TCP to adjust properly to the network conditions, the performance depends on the behaviour of the endpoints. This project did not include in its scope to profile the behaviour of the TCP senders or receivers. However, it provided an easier solution to retrieve the sender information for HTTP transfers by extracting relevant fields from the HTTP headers, as presented in section 2.5.2. The main problem encountered while extracting the sender information was how to map it onto a numerical variable. The issues come from the differences between different implementations: there is no better/worse relationship between them (e.g. a Linux-based TCP client, inferred from an Apache Linux server tag, is not better or worse than a Windows-based TCP client, inferred from an IIS tag, they are simply different). Further, different versions of the same implementation have different characteristics (for example, IIS 4 is likely to run on a Windows NT4 server, while an IIS 5 server will be running most likely on a Windows 2000 / XP server, resulting in differences between the TCP implementations).

Considering the above-listed factors, but also the inputs used by the mathematical models described in section 2.5, two types of models were proposed: one for loss-free connections and one for connections that encountered losses.

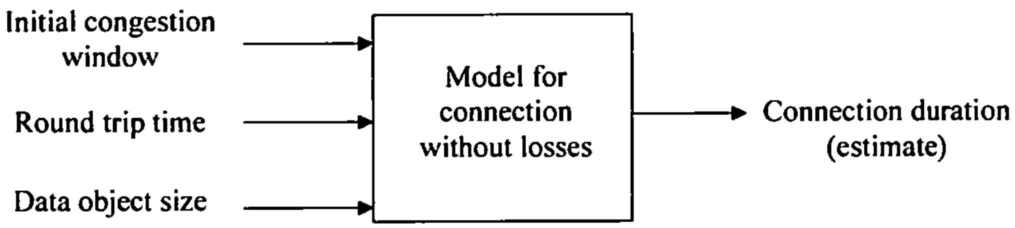

Initial congestion window
Round trip time
Data object size
Model for connection without losses
Connection duration (estimate)

Figure 7.4 - Block diagram of the TCP model for connections without losses

As shown in Figure 7.4, the TCP model used for connections without losses takes three inputs for each connection: the initial congestion window, the average round trip time, and the size of the object to be transferred. Using these inputs, the model produces a duration estimate for that particular connection.

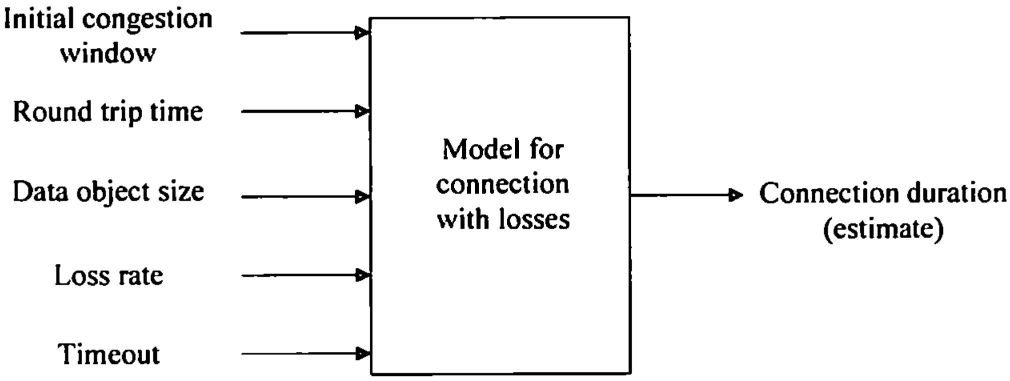

Initial congestion window
Round trip time
Data object size
Loss rate
Timeout
Model for connection with losses
Connection duration (estimate)

Figure 7.5 - Block diagram of the TCP model for connections with losses.

The TCP model for connections that encountered losses, as presented in Figure 7.5, uses the three inputs from the no-loss model and two additional ones (loss rate and timeout), to account for the behaviour of the TCP client when packet loss occurs.

The second part of the data pre-processing relates to filtering. Ideally, both the input variables and the output predicted performance should be uniformly distributed in the sample space; in the real case, for the environments studied, at least for the delay and throughput, the distribution is concentrated in a narrow spectrum but is long-tailed, as was illustrated by the analysis performed in Chapter 6. The problems with such distributions are two-fold:

- Normalisation. The neural network requires the input and output values to be normalised to the interval [0.0, 1.0]. As a result, the larger the interval, the lower the accuracy relative to the overall average value of the throughput. In order to reduce the errors introduced by this issue, four different methods of scaling were tested to reach the optimum accuracy. The results of the tests are presented in section 8.5.3.

- Training. The neural network requires sufficient values to be trained throughout the definition domain of the values; if there is insufficient data in a specific sub-interval of this domain, the neural network will perform poorly or, at least, will be biased by the rest of the domain.

To limit the effect of these two issues, the output data from the TCP analysis was filtered to remove the extremities of the distribution. Two types of filtering were applied:

- Simple – remove only the extreme values for the output variable (throughput)
- Comprehensive – remove the extreme values for throughput *and* the input variables (delay, loss, congestion window)

The second filter was obviously more aggressive, as it removed the extreme values from all domains, not only the output one. In the end, a biased version of this filter was used: eliminate the extreme 1% from the input variables domains and the extreme 5% of the output variables domains. It is worth mentioning that the filters were not additive, but simultaneous – the filter for each variable was applied to the original dataset, not on the remaining data. All these measures aimed to balance between excessive removal of data and reduction of the prediction errors.

7.5.2 Data analysis – procedure and algorithms

The crucial problem within the data mining step is deciding what method to use. In informal discussions regarding IDA, there are three main issues arising: 'use good data', 'understand your problem', and 'explain your results'. From the three, the second step is the one that is reflected in the data mining choice: the algorithm itself is nothing but number crunching; what is important is to 'understand the problem', to clarify what results are expected at the output of the algorithm.

As shown in Figure 7.4 and Figure 7.5, two separate sets of IDA analysis engines were used, depending on whether the connections encountered any packet loss or not. In spite of the differences between the number of inputs and, implicitly, the structure of the chosen data analysis method, the main characteristics of the two associated data sets remained largely the same:

- Small number of inputs. The number of inputs was either 3 or 5, depending whether loss was modelled or not
- Large number of samples. Aside from a notable exception, lack of connections with loss, the number of collections analysed was fairly large – at least thousands of samples. As will be shown in Chapter 8, this also had an impact on the way data was split into training and a testing subset.

The starting point was to compare the learning capability of a neural network with the relationship between the performance of TCP and its influencing parameters. Neural networks may be employed to reproduce this relationship by modelling a function that bridges between the variations of the output (TCP throughput) and the inputs that caused it (network and transfer parameters). This is due to the fact that function approximation is one of the six typical learning tasks for Artificial Neural Networks (ANNs) (the others being pattern association, pattern recognition, control, filtering, and *beamforming*) [Haykin 1999]. Some of these tasks have already proven the ANN's superiority in the networking area, as indicated in section 7.4.

One main IDA toolset was identified when approaching the task of deciding for a processing technique: SNNS, the Stuttgart Neural Network Simulator (SNNS) [SNNS 2003], a generic neural network engine that includes a comprehensive list of neural network algorithms to use for specific prediction / classification tasks. SNNS was preferred as the environment to test the efficiency of the neural network due to its complexity and maturity. The product began its development in the early 90s [Zell *et al* 1994]; at the time when it was used within this project, it included all major types of neural networks and came with a powerful environment that allows loading/training/testing/accuracy evaluation of a neural network on a dataset and with a full GUI to perform all these tasks. The project had a limited need for the GUI support, as it used *batchman*, the SNNS scripting environment to automate the testing/training of neural networks; in fact, the entire data processing, from raw form to final post-processing output, was built within a script, included in Appendix C.

The accuracy of various neural networks was tested intensively for the available datasets. The evaluation included analysis to determine the optimum for: the split for training/testing subsets, the algorithm and type of network to use, and, finally, the training parameters for that specific algorithm.

The aim of this section is only to describe the steps involved in the IDA processing model. This is due to the fact that, although the actual processing is performed using known techniques and available software packages, the approach itself, i.e. building a TCP performance model using IDA, is novel and so is the entire process involved (extraction and preparation of input parameters and output variable, the analysis, and post-processing of the results). A detailed description of the experiments performed and their results will be given later on in Chapter 8 as part of the validation tests, together with an accuracy comparison between the proposed IDA-based model and the existing mathematical models.

7.5.3 Interpretation of the results

In both of the cases, JGDM and SNNS, the output of the analysis method was improper or incomplete for the proposed method. This is why it was preferred to save the evaluation results in a rather raw form in each case and in the post-processing phase to apply a different analysis on the results.

First, for both of the methods used, the visual (graphical) comparison was used to determine how well the predicted values follow the real values. This involved plotting each of the *n* real values s_i versus the predicted values p_i: $f(s_i) = p_i, i = \overline{1,n}$. In the ideal case, 100% accurate prediction, the result is the identity function, f(x)=x. Although convenient, the graphical method cannot be used for analytical purposes and the accuracy has to be evaluated using mathematical tests, relating to the relative errors between the above-mentioned s_i and p_i.

In the JGDM case, due to the integration of the toolset, the overall classification result had the same format for all the algorithms: indicating the percentage of correct classification. This type of output is exactly what is required by a classification method: how many samples were correctly classified and how many were erroneously classified. What this overall result does not tell is the error, the 'distance' between the real value and the predicted one. This is because the classification algorithms used were designed for unrelated output variables (e.g. classifying fruits in apples, pears, plums, etc) and there is no metric to uniformly describe their domain. This is why, during the post processing phase, the comprehensive format was preferred, which provided information about how the variables were classified and it consists of a square table that has on columns the real values of the output variable and on rows the classes as resulting from the classification method[1].

The SNNS toolset included a much more convenient facility: output the predicted value for each data sample tested. This allows a simpler comparison between the real values and the predicted ones to obtain the average relative error. However, the relative error is not relevant for the accuracy of the prediction if the domain of the variable is very narrow (which is the case for some of the throughput values in the validation datasets from Chapter 5), because the prediction domain will be similar with the real domain. This is why, aside from the relative error, the accuracy of the prediction was evaluated using the correlation *r* between the predicted values and the real ones.

Summarising, three indicators were used to determine the accuracy of the SNNS prediction:

- graphical, very good but human-observation based, therefore unusable for analysis;
- average relative error, to determine whether or not the predicted values are similar with the real values;
- correlation factor, to eliminate the inaccurate prediction cases undetectable through relative error due to narrow domains.

[1] The actual table representation was used also for graphical estimation of the resulting error of the algorithm. The narrower and closer to the top-left-bottom-right diagonal is the area of non-zero values, the more accurate the algorithm is.

7.6 Implementation

An actual implementation of the IDA process was produced only for the SNNS-based analysis. As was mentioned before, the classification approach did not lead to satisfactory outcomes, therefore the JGDM toolset was used while experimenting rather than during consolidation of the results. The aim of this section is to overview the components of the SNNS-based processing.

The input used for the processing was the raw network traces, captured with ***tcpdump***. The actual process of collecting the traces and the various sources of traffic used were detailed in Chapter 5 and Chapter 6.

The implementation followed the three stages of IDA processing:

1. Pre-processing.

   This stage included scripts and programs to perform the required tasks to produce a database of TCP connection samples. Each sample included network performance and transfer parameters, as well as the resulting duration of the data transfer, recorded for a specific connection. The analysis of the network trace and the parameter extraction were performed using the method described in section 3.4. After obtaining a raw database of samples, further processing was required to provide an optimum dataset for training and testing:

   - Connection filtering - remove reset/unfinished connections or very short lived connections.
   - Parameter filtering – remove samples with parameters that take values considered outliers.
   - Randomising – in order to avoid estimation errors due to different domains of definition for parameters that varied in time.

- Scaling – normalise the dataset in order to make it appropriate for the data mining engine
- Splitting and formatting – separate the dataset into a testing subset and a training subset; produce a format compatible with the method used for analysis.

2. Processing – included scripts to communicate with *batchman*, the SNNS programming environment

   - Train the neural network with the training dataset, using parameters defined by the user.
   - Test the neural network with the testing dataset after each *n* training epochs, as defined by the user.
   - Provide mechanisms for early stopping. The scripts allowed various training process scenarios:

     - Exhaustive training – do not apply any early stopping mechanisms
     - Early stopping using a single Mean Square Error (MSE) / correlation factor test – stop the training if the resulting MSE / correlation factor is below a certain value.
     - Early stopping using multiple MSE /correlation factor tests – stop the training if the resulting MSE /correlation factor is below a certain value for *x* successive tests

3. Post-processing – included scripts to evaluate the efficiency of the prediction

   - Extract MSE and calculate the correlation factor
   - Plot the prediction graph

All the programming was produced under Linux, using *shell*, *tcl*, and *awk* scripting language. The content of the scripts and further functionality details are given in Appendix C. The results of applying the neural networks on various types of data will be presented in Chapter 8.

7.7 Summary

This chapter began by listing the limitations of current TCP modelling efforts. It was observed that existing mathematical models, while accurately describing the behaviour of TCP transfers, may not map correctly in the case of real network conditions and traffic. As a result, Intelligent Data Analysis has been developed as a valid alternative for estimating TCP performance based on network and transfer parameters.

The chapter described the content of a generic IDA process and applied that description onto the case of TCP performance prediction, as used within this project. Four main stages were identified within the IDA process, as applied to this project: data collection, connection analysis and pre-processing, data mining, and post processing.

Chapter 8 will describe the results of the validation tests of the IDA approach, using all available sources of data, processed using the implementation described in this chapter.

Chapter 8. Validation of TCP performance prediction method

8.1 Obstacles

The development of an accurate prediction method has to consider several possible obstacles: the lack of useful / correct network data/traces, the filtering criteria that should be in place when processing the connection data, the accuracy of the TCP analysis method itself, and the evaluation of its inherent accuracy.

The amount of (relevant) data is critical to ensure the success of an intelligent-based prediction method as the function that approximates the output is based on the available data. In the case of this project, the data consisted of raw network packets, captured from an aggregation point. The best source of such data in terms of generalisation is represented by publicly available raw network trace archives, with the best example being the traces maintained by NLANR. Unfortunately, since privacy laws such as the Data Protection Act were established, capturing raw network traffic has been regarded as a breach of the user's privacy, unless anonymised and sanitised. In order to obey the privacy requirements, research projects that dealt with data gathering had to revise their policy of distributing data. Three solutions were found: stop making publicly available network traces that captured uncontrolled traffic[1], as happened in the case of LBNL; switch to infrastructure-based measurements [AMP 2003]; reduce the amount of information contained in the raw traces [PMA 2003]. The third alternative, while being a theoretically partially viable option, (still unsuitable at its best for the full proposed TCP analysis[2]) was only very recently (e.g. 2001-2002 for NLANR) extended towards formats suitable for TCP analysis. Older traces, due to collection characteristics such as separate

[1] This category does not include measurement infrastructures, as they generate synthetic connections (e.g. fixed sized objects). Traces from such experiments reflect accurately the network characteristics, but do not provide an accurate image of the typical end-user traffic.

[2] The typical capture includes a fixed header, with the largest available size including only the 40 bytes on top of the link layer (enough for minimum IP and TCP headers). This eliminates any scope for TCP timestamp-based analysis.

collection and anonymisation for inbound and outbound traffic, were appropriate only for workload analysis.

The remaining option was to analyse traces collected from the NRG/UoP backbone. The advantage in this case was having full control over the amount of information that has to be removed, without breaching the privacy laws, via the use of software tools such as ***tcpurify*** or ***tcpdpriv*** that allow trace sanitisation without removing the options carried by the TCP header, such as the TCP timestamp options) with the downside of analysing only a smaller network environment (but, still, a large amount of traffic at it became apparent).

The dataset was passed through pre-processing in order to remove outliers, in two phases: first was the removing of the connections containing outliers of the variable to be predicted, then the connections including outliers of any of the inputs, as explained in Chapter 7. The boundaries for two sets of filters were set at 5% (i.e. keep the 5%-95% interval) for the predicted variable and 1% (i.e. keep the 1%-99% interval) for each of the attributes.

8.2 Data analysis

One of the main objectives of the TCP analysis was to produce a tool with online capabilities, which would be able to study in real-time the TCP connections and output the results. This may seem to contradict the prediction part of the project, which used throughout the experiments offline analysis. The decision was taken due to two factors. First, the volume of data generated/collected in real time was not large enough to stress the monitoring tool; only towards the end, during capture of the backbone traces, would the volume of data have been sufficiently high to test the analysis implementation. Second, both the TCP analysis method and the prediction method required repeatable analysis during both development and finalising stages.

The TCP method required numerous adjustments, most of them based on visual analysis of the traces (e.g. to observe the evolution of the congestion window), while the neural network required time for training using the connection samples. In spite of this approach, offline analysis fulfilled also the stress requirements, as the traces, which were collected over longer or shorter periods of time, were streamed to the analysis program.

The training/testing data came from three main sources: *NS* simulations, the RYL experiments and, as an opportunity appeared towards the end of the project, the UoP backbone traces. Each dataset was split into two subsets, based on the evolution of the data transfer, with one dataset including only connections without losses, while the other consisted of connections with losses. The two subsets differed not only in complexity of their evolution in time (connections without losses do not vary the policy of the congestion window increase) but also in the number of inputs.

The transfer during connections without losses is affected only by the congestion window characteristics (initial value), the amount of data to transfer, and the round trip delay between the two endpoints. As a result, the model for these connections used only three inputs: the amount of data to transfer, the initial value of the congestion window (both in bytes), and the estimated round trip delay (in milliseconds). Initially, the maximum value of congestion window was added to the list, to consider the case when the congestion window increase is limited by the receiver advertised window. However, the connection analysis revealed that this did not happen throughout any of the datasets, due to the value of the advertised window itself (32120 bytes, equivalent to 22 full 1460-byte packets)[1] and the small amount of data transmitted during each connection (which did not allow the congestion window to increase too much). Some of the receivers from the UoP dataset had implemented, however, a window scale mechanism [Jacobson *et al* 1992] to avoid such limitations.

On the other hand, connections with losses had several loss characteristics that may have been added to the list of parameters: the fast retransmit loss rate, the timeout loss rate, and the first occurrence of loss. However, as shown throughout section 2.5, mathematical models require only the loss rate and the (estimated) timeout in order to predict the loss events that happen during a connection. In addition, it would be impossible to predict, from a total loss rate, which proportion of it will have secondary effects such as timeout events. To make the neural network model comparable with the mathematical one and, more importantly, to allow its usage on generic loss rates, the list of loss-related parameters included only two variables: the total loss rate and the estimated timeout. The total loss rate was a sum of the visible and inferred fast retransmissions and inferred timeouts. The estimated timeout period was almost impossible to extract due to reasons explained in section 3.5.2. It was instead replaced with the total duration of timeouts; this had a small impact on the mathematical model too, as the $E[Z^{TO}]$ term from equation 2.9 had to be replaced with the measured total timeout, as produced by the TCP analysis. The mathematical model of connections with losses was different in the sense that it used segments rather than bytes to describe the amount of data transferred. These two issues led to the following list of inputs for the connections with losses: the amount of data to transfer d and the size of the initial congestion window w_I (both measured in segments), the round trip delay RTT and the estimated timeout T_0 (measured in seconds), and the loss rate (ratio between the number of retransmitted bytes and the number of useful data bytes transmitted)

[1] This may appear contradictory to Balakrishan's findings from [Balakrishan *et al* 1997], where he observed that in 14% of observed connections the congestion window reached the level of the receiver advertised window. However, as observed by the author, the data collection was server-based, resulting in a variety of remote receivers with some of them having maximum advertised windows as low as 4KB.

These differences imposed the development of two individual neural network-based models, one suitable for loss-free transfers and one designed for connections that encountered losses[1].

The Internet traces were subjected to an additional filter in order to consider the connections that might have experienced delayed acknowledgments. As identified in [Cardwell *et al* 2000], connections with a start window of 1 packet will experience delayed acknowledgments and the actual delay of the acknowledgment depends on the implementation used. To eliminate heuristics, such connections were removed from the traces.

A problem identified from the TCP connection analysis stages was the availability of data. Due to the characteristics of (at least) the environment where the traces were collected, most of the connections were loss free, a phenomenon described during Chapter 6. Because of this, while the neural network training had enough of samples even after removing some of the data, training of the neural network model for connections with losses required all traces available in order to produce a sizeable dataset.

Each subset (i.e. with or without losses) was first filtered and then fed to the corresponding neural network. The following sections start by describing identification of the optimum model. This first part of the process required analysing the impact of several variables, ranging from the structure of the neural network to the training parameters used in each case. Certain assumptions were made during these preliminary stages in order to simplify the analysis, such as using values recommended by literature for certain training parameters. Also, the analysis limited the scope of randomness in order to reduce the time required to study all alternatives.

[1] In the preliminary stages it was attempted to produce a single network that would be able to deal with both types of connections, but its output had low accuracy for both kinds of connections.

The estimation results from the neural network models were compared with the current existing mathematical models in order to assess whether they are superior in terms of accuracy and robustness. The comparison was made in terms of relative error, both in the average value and actual distribution, as well as using the correlation factor from the two methods.

8.3 Preliminary tests – Connections without losses

The purpose of the preliminary analysis was to evaluate which neural network (in terms of structure, algorithms, and parameters) is likely to lead to a greater accuracy for a set of connections that do not encounter any loss events. All tests from this section used the no-loss connections subset of the RYL dataset, as a compromise solution between the synthetic nature of the *NS* simulations and the variety of the UoP traces. To increase the number of samples in the dataset, the two rounds of experiments were combined in a single dataset which included 16865 samples. This also added an element of robustness to the problem as the network environment changed between the two sets of experiments.

There were three main parameters to vary when applying a neural network to a dataset: the structure of the network, the method applied, and the amount of data to be fed to the network. The structure of the network included variables relating to the number of hidden layers, the connectivity between neurons, and the number of neurons for each hidden layer. The method applied was the most complex part, with several sub-divisions: establishment of the training algorithm together with the initialisation and update functions, identification of the optimum parameters for the chosen network, followed by detection of the early stopping decision that would lead to highest accuracy.

8.3.1 Neural network structure

The design of the networks started by considering the number of input and output neurons, and then produced several alternatives of hidden layer(s). For the loss-free case, the dataset included samples with three inputs: the connection size and initial congestion window (both measured in bytes), and the round trip delay (measured in seconds); the network had a single output – the duration of the data transfer (measured in seconds). Based on these inputs/outputs, three neural networks were generated, all fully connected: 3-2-1 (a single hidden layer with two neurons), 3-6-3-1 (two hidden layers, first layer with six neurons, second layer with three neurons, shown in Figure 8.1), and 3-12-6-3-1 (three hidden layers, first layer with twelve neurons, second layer with six neurons, and third layer with three neurons). The reason for using neural networks with two hidden layers was that some previous studies, such as [Funahashi 1989] and [Chester 1990], have shown that neural networks with two hidden layers may perform better than classical 1-hidden layer topologies.

With regards to the first set of parameters, i.e. the structure of the network, there was no unique set of guidelines with regards to the number of hidden layers or the complexity of each layer. This is why the preliminary tests were performed using the 3-6-3-1 network, with the structure shown in Figure 8.1

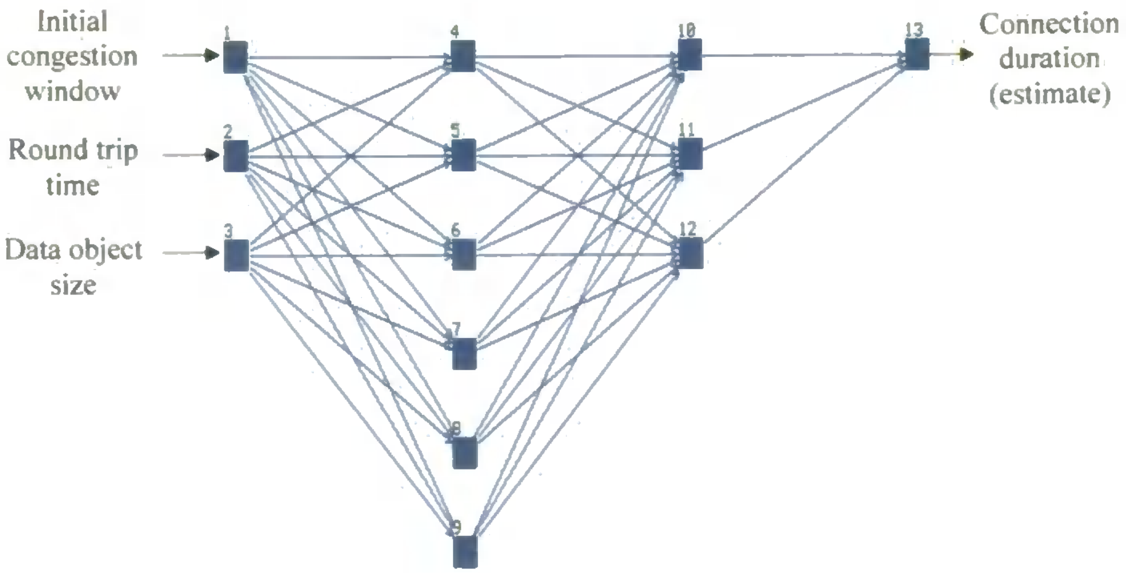


Figure 8.1 - The inputs, output, and the structure of the 3-6-3-1 neural network used during the preliminary tests for connections without losses

The chosen network has three inputs and one output because the preliminary tests used only the no-losses subset of connections from the RYL dataset. The other two networks were also used during the validation tests in order to determine whether they improve the accuracy of the predicted variable.

8.3.2 The stopping criteria

First set of correlated decisions was to find the optimal values of three factors: splitting the dataset into a training subset and a testing subset, the early stopping criteria, and the frequency of tests. All these would impact on the ability of the network to stop before overfitting[1]. These decisions made extensive use of the prior research in the area of neural network training. A good start was provided by [Morgan and Bourlard 1990], [Weigend *et al* 1990] and [Amari *et al* 1997] which describe the phenomenon of overfitting while training a neural network, advocating the split of the dataset into a training subset and a testing subset, in order to perform early stopping. In addition, the mathematical modelling from [Amari *et al* 1996] provided the optimum testing ratio t for networks with a number of m network parameters to be

$$t = \frac{1}{\sqrt{2\,m}}, \text{ where } m = \sum_{i=1}^{n-1}(n_i + 1)n_{i+1} \qquad 8.1$$

for a network with (n-2) hidden layers and n_i neurons in the ith layer. In the case of the network used to train the dataset (3-6-3-1),

$$m = (3+1)\cdot 6 + (6+1)\cdot 3 + (3+1)\cdot 1 = 46 \Rightarrow t = \frac{1}{\sqrt{2\cdot 46}} = 0.10425 \qquad 8.2$$

The figure was rounded to 0.1, resulting in a ratio of 10% testing and 90% training.

However, [Amari *et al* 1996] also describes the other extreme: if the size of the datasets is large compared with the number of free network parameters, early stopping will provide only a marginal improvement in the generalisation error. This case, named *asymptotic* in the study, happens when number of samples W is much larger than the number of free network parameters m that satisfied the 8.3 formula

[1] The process of training a network using a dataset results in a decrease of the relative error when applying the neural network on that dataset. However, after a certain number of training epochs, the network will start to *overfit* the dataset by learning the noise contained in the data rather than the determining features. This will lead to lower

$$W > 30m \tag{8.3}$$

If a W=10000 samples dataset is considered, in conjunction with the above mentioned 3-6-3-1 network, equation 8.3 is satisfied (using data from equation 8.2, 46·30=1830<10000). In fact, for the datasets and neural networks used, the equation is satisfied, at least at the limit, in all the cases. A test was run on the RYL dataset[1], using the 3-6-3-1 network with η=0.01, τ=0.0, μ=0.1, c=0.1, to verify whether MSE starts to increase, as training progresses, for the test subset while it continues to decrease for the training subset. The split of the dataset was the optimal one, i.e. 10% testing and 90% training; the results are displayed in Figure 8.2.

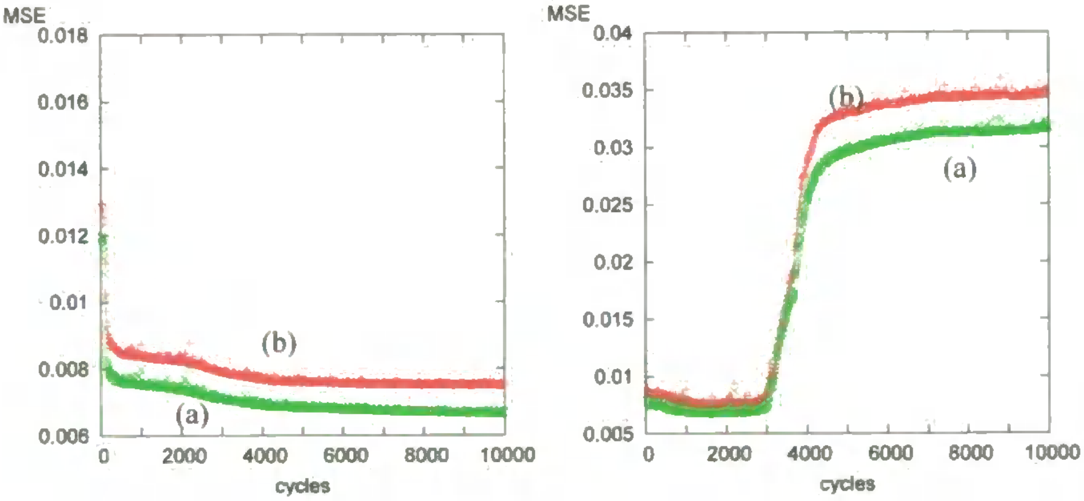


Figure 8.2 MSE=f(cycles) for exhaustive training of the RYL dataset, with a 10% testing (a) and 90% training (b) split, using η=0.01, τ=0.0, μ=0.1, c=0.1 (left) and η=0.1, τ=0.0, μ=0.1, c=0.1 (right)

errors when applying the training dataset but to higher errors when generalising (applying the neural network to previously unseen samples and datasets).

[1] The experiments performed during this subsection relied also on conclusions drawn from section 8.3.3, as the two sets of parameters are independent and one of them has to be preset in order to observe any variations of the other.

The experiment indicated that, as expected, there is no divergence between the training and testing subsets. As a result, the entire set may be used for training and the early stopping condition may be applied also to the entire set.

For this case, Amari recommended [Amari *et al* 1997] that exhaustive training should be used, without using cross validation. Ideally, the training method should be stopped just before the network model becomes overfitting for the dataset. This would be equivalent with a zero or positive variation of the MSE between two successive training cycles. The aim is theoretically attractive, but it cannot be achieved in practice due to computational complexity (the network would require testing after every training cycle) and due to the evolution of the network accuracy (e.g. the average error values might fluctuate slightly before stabilising). To illustrate the problem, Figure 8.3 presents the variation of MSE as a function of the training time (in cycles) for three values of the training rate (η=0.01, η=0.1, η=0.9).

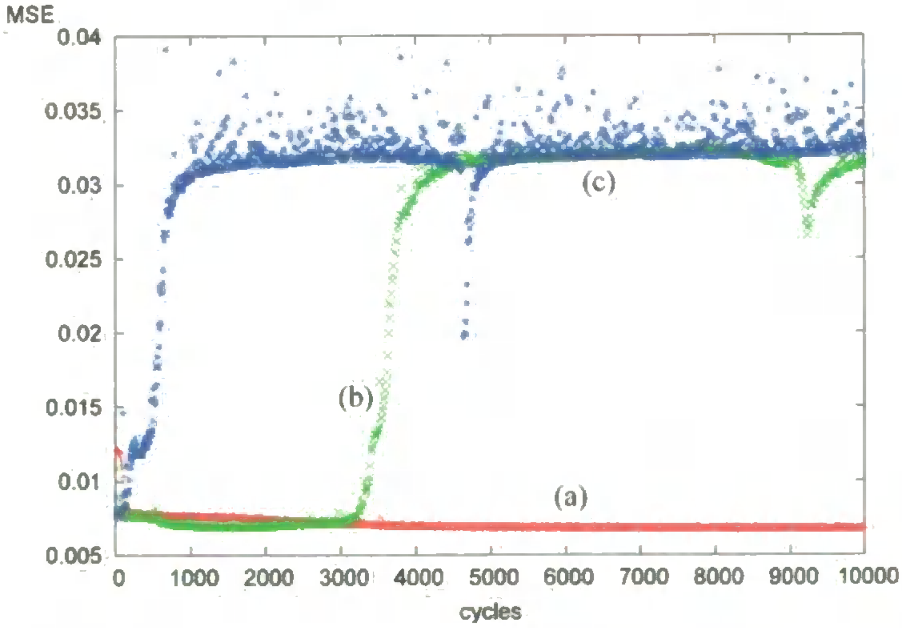


Figure 8.3 - MSE=f(cycles) for exhaustive training of the RYL dataset, using (a) η=0.01, τ=0.0, μ=0.1, c=0.1, (b) η=0.1, τ=0.0, μ=0.1, c=0.1, and (c) η=0.9, τ=0.0, μ=0.1, c=0.1

It can be noticed that the MSE decreases and then becomes stable after 3000 cycles for η=0.01, after approximately 1000 cycles for η=0.1, and it takes only approximately 100 cycles for the network to reach the minimum MSE when η=0.9. This proves that it is difficult to choose a single value for exhaustive training in all cases and, therefore, early stopping will be applied in all cases to find the point of convergence. To identify the point of convergence, the preferred alternative was to use a very low limit of the variation (0.00001) with 10 cycles as testing period and a short-term average to eliminate the possible oscillations [Prechelt 1998]. The average was calculated in batches of 5 consecutive tests, which means that the network was stopped only when the MSE decreased by less than the variation limit five times in a row. Further, in order to avoid missing the minimum due to oscillations with an increasing trend, the minimum MSE value was memorised throughout the test. An example of such behaviour where the MSE increased gradually over time, but not 5 times consecutively, is given in Figure 8.4

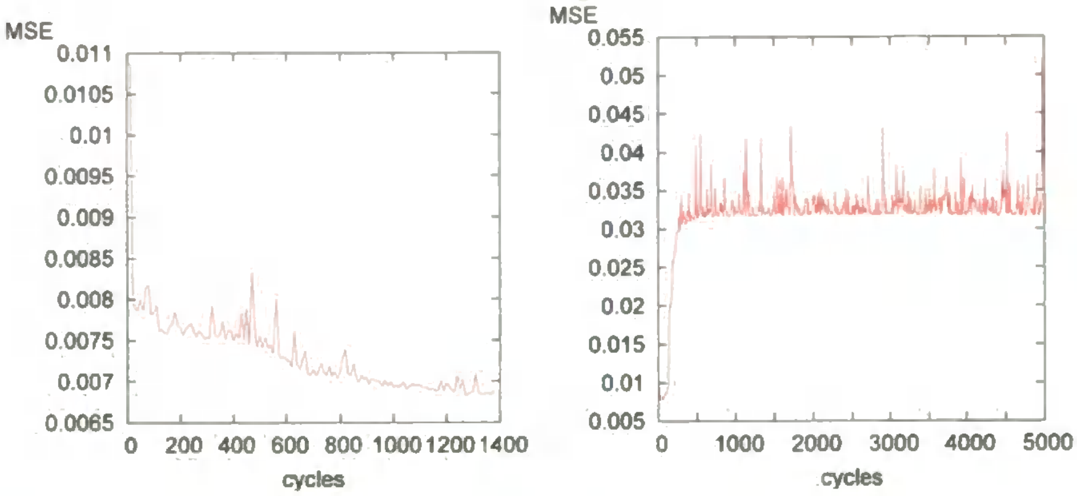


Figure 8.4 MSE=f(cycles) for two different learning processes: (left) slow convergence, with η=0.1, τ=0.0, μ=0.1, c=0.1, and (right) fast convergence, with η=0.5, τ=0.0, μ=0.7, c=0.1

The two experiments were stopped using the same criteria: the resulting MSE decreased by less than 0.00001 in five consecutive tests. The advantage of evaluating the errors over several tests becomes clear for the slow-converging network shown on the left in Figure 8.4. Although the error oscillated in the short term, it had an obvious decreasing trend, which stabilised over 1000 cycles and started to oscillate around a constant level, then increased. On the downside, for the fast-converging network, pictured in the right diagram, the training was stopped long after the convergence (which appeared at 60 cycles), because the error did not increase monotonically from 0.007 to 0.030. In this case it may be admitted that exhaustive training was performed but the minimum value was recorded. In these cases, the early stopping proposed rule is not efficient and the training mode reverts to exhaustive training, which was limited to 5000 cycles.

8.3.3 The optimum parameters for a set network

The method applied to the network involved a wider range of issues. Firstly, the training algorithm had to be chosen and, after testing various algorithms, it was decided to use back propagation with momentum and flat spot elimination. The decision was motivated by three factors: the nature of the problem (i.e. time-independent mapping function), which eliminated e.g. the memory-based algorithms, generic recommendations for back-propagation algorithms [Haykin 1999], and recent developments into such algorithms. Indeed, the chosen variant of back-propagation has two improvements when compared with its predecessors: the momentum term μ that absorbs any eventual oscillations of the resulting error and the flat spot elimination term c, which allows the network to surpass possible flat spots (local minima) on the error surface. Aside from these two terms, the algorithm also includes the typical learning rate η and tolerance threshold τ, which define back propagation. The weights of the neural network were initialised with random values in all experiments; the only particular feature was maintaining the same seed for network initialising throughout a single batch of tests. This allowed proper comparison of the results of each set of training values throughout a batch and also offered repeatability of a batch of experiments. The process of identifying the optimum values for the four parameters was automated using the functionality of the script from Appendix B1. The establishment of the intervals for the parameters was based on the recommendations from [SNNS 2003], which suggested $\eta\in[0.1;1)$, $\tau\in[0;0.2]$, $\mu\in[0.1;1)$, $c\in[0.0;0.25]$.

The first step in determining the optimum training parameters for the 3-6-3-1 neural network was to set the minimum threshold, τ=0.0. This is because the purpose of the neural network was to follow as accurately as possible the real values, rather than provide a confidence interval for them. This left the learning rate and momentum terms available to vary.

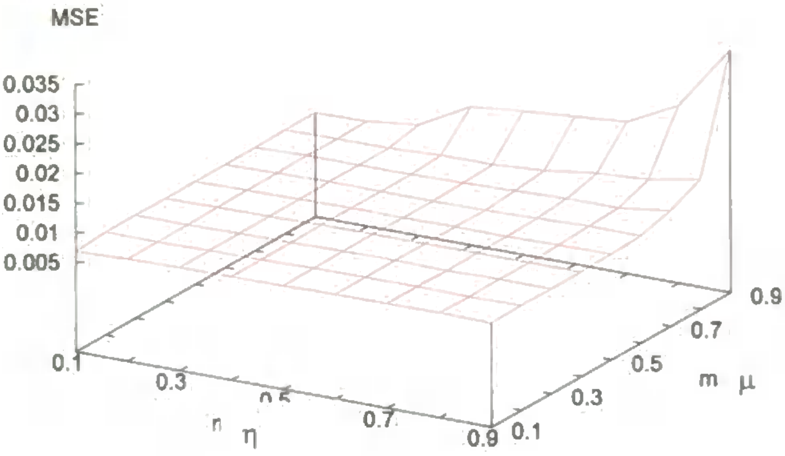


Figure 8.5 MSE=f(η,μ) for η∈[0.1;0.9] and μ∈[0.1;0.9] (τ=0.0, c=0.1)

Figure 8.5 presents how MSE varies when η and μ vary in the [0.1; 0.9] interval; both parameters were incremented in 0.1 steps. It can be observed that the surface is relatively flat, except the extremities: the value obtained for (η, μ) = (0.1, 0.1) is slightly smaller than the majority of the tests, while the errors resulting for high values of μ, particularly when coupled with high values of η, are higher than the rest of the values. It is interesting to note that there are certain studies (e.g. [Hertz and Krogh 1991]) that encourage the use of combinations such as high η and high μ in order to achieve fast convergence, but, at least in this case, this higher convergence speed comes with dramatically reduced accuracy. This indicates that the figures chosen for η and μ should be as low as possible (the accuracy is almost constant when they are set to values in the [0.2; 0.7] interval).

A second batch of training tests was run in order to explore the impact that low values of η and μ have on the accuracy of the network. The focused intervals this time were η∈[0.01;0.1] and μ∈[0.1;0.9], with 0.01 increments for η and 0.1 steps for μ. This batch was trained exhaustively up to 10000 cycles, but the lowest MSE value was selected from the training process. This was done to ensure that the early stopping criteria does not stop the training process before it has finished, which was likely to happen with the low values chosen for training rate. The results are shown in Figure 8.6.

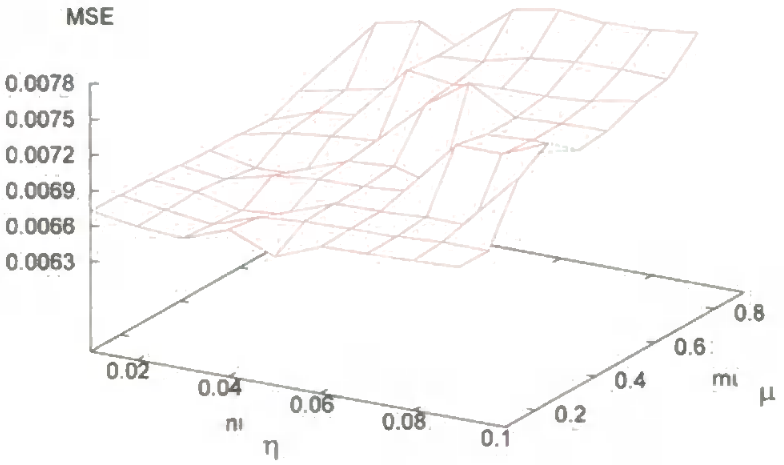


Figure 8.6 - MSE=f(η,μ) for η∈[0.01;0.1] and μ∈[0.1;0.9] (τ=0.0, c=0.1)

It may be observed that the graph exhibits a similar trend with the one in Figure 8.5, i.e. MSE decreases for lower values of η and μ. However, the decrease was not monotonic, as the minimum MSE = 0.00658841, obtained for η=0.03, τ=0.0, μ=0.3, c=0.1, after 9660 training cycles.

Based on the batches of training session described, it was concluded that the (η,τ,μ,c)=(0.03,0.0,0.3,0.1) parameter set was the combination that would lead to best accuracy.

8.4 Validation tests – Connections without losses

The first part of this section is based on the RYL traces, while the second part provides an indication of whether generalisation of the problem, based on the UoP backbone traces, affected the results.

8.4.1 The NS dataset

The *NS* dataset was obtained by generating network infrastructures using random values for the network characteristics. As explained in section 5.5.1, the simulations produced a variety of environments, with up to three levels of connectivity. These environments were used to transport TCP connections starting and finishing at random moments, therefore exchanging data objects of different sizes.

The variety of the clients was the only problem that occurred when processing the *NS* traces. *NS* had a generic type of TCP implementation, called *TCPfull*, which had set parameters for all TCP client variables. Although the documentation [Fall and Varadhan 2003] indicated methods to vary some of these variables, the traces did not exhibit any variations when varying the initial congestion window. Because of this, the data generated had the same value (one full packet) for the initial congestion window for all connections. As the initial congestion window was one of the inputs of the neural net and there was no reason to apply a constant input to the network, a variant of the neural network was designed only for this set of connections. The network had 2 neurons in the input layer, 2 hidden layers (6 neurons in the first hidden layer, 3 neurons in the second hidden layer), and 1 neuron in the output layer.

The dataset was applied to the neural network and trained exhaustively using 10000 cycles, employing the set of parameters obtained in the preliminary tests - (η,τ,μ,c)=(0.03,0.0,0.3,0.1). At the end of the training session, the minimum registered MSE was 0.00222113, resulting after 4720 cycles. After the training, the dataset was then applied to the neural network that led to the minimum MSE in order to obtain the most accurate estimators for the duration of the transfers. Figure 8.7 displays a plot of the estimated values vs. the actual values.

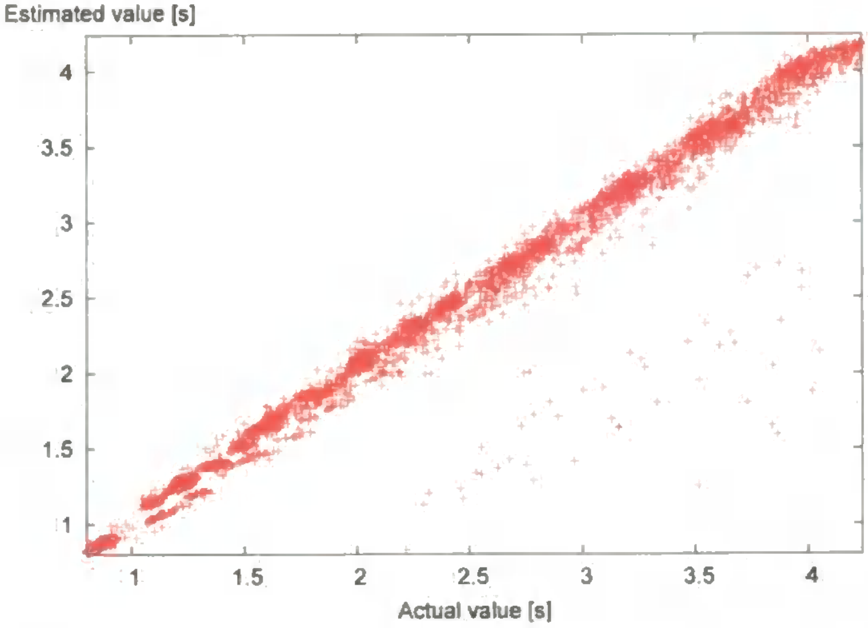


Figure 8.7 – Plot of the real values vs. estimated values, as resulting from the *NS* dataset

The neural network followed the real values accurately, as exhibited by the graph in Figure 8.7, with only a few significant errors appearing towards larger values of the spectrum.

The next step was to apply the *NS* dataset to Cardwell's model in order to determine how accurate the mathematical model is when compared with the neural network model. The results produced by this second model were then compared with the results from the neural network model. The statistical results of the comparison are presented in Table 8.1. It may be observed that, on average, the neural network model outperforms the mathematical model in terms of accuracy. The table also includes a column for the correlation between the predicted values and the real values in order to illustrate that the accuracy of the neural network was not due to the narrow spectrum of the output variable, but to learning the variations in transfer duration produced by changes in the amount of data transferred and the RTT values.

One final test was run in order to determine whether a simpler network configuration would lead to better results. A neural network with only 2 hidden neurons, placed in 1 hidden layer, was trained under the same conditions (same parameters, dataset, and initialisation seed). The results from the model were introduced in Table 8.1, the *neural network (3-2-1)* entry, for comparison purposes.

Model	Average relative error	Stdev. of relative error	Correlation
Mathematical	0.292689	0.184215	0.969077
Neural network	0.0305206	0.0478511	0.987012
Neural network (3-2-1)	0.0454152	0.0527412	0.985258

Table 8.1 Comparison of the resulting average figures for the *NS* dataset, using the mathematical and the neural network models

Table 8.1 shows that the neural network models lead to lower average error values. It may be observed also that the 3-2-1 neural network, although still superior to the mathematical model, it shows a 50% higher figure for the average relative error when compared with the results from the 3-6-3-1 neural network. This led to the conclusion that simpler structures may be used for the neural network with similar qualitative results, but they would impact on the quantitative results obtained. The following and conclusions were exclusively based on the 3-6-3-1 neural network structure, unless otherwise stated.

A second graph was produced to illustrate how the neural network and mathematical models perform throughout the dataset. The graph, shown in Figure 8.8, compared the relative error produced by the two models.

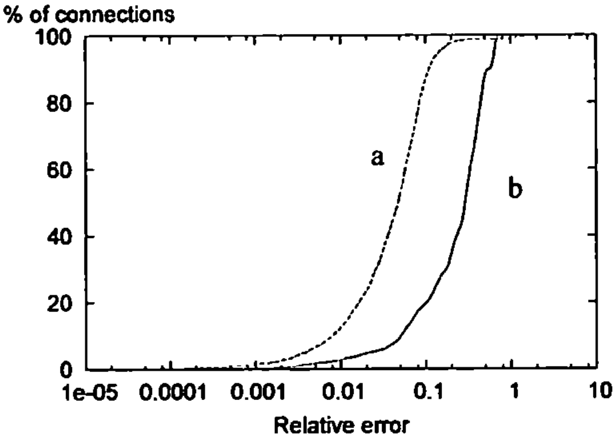


Figure 8.8 Cumulative distribution of the relative error for the RYL dataset using the (a) neural network model and (b) mathematical model

The figure above confirms the average figures, indicating that the neural network model outperformed the mathematical model throughout the vast majority of the dataset. The figure also reveals the outliers from Figure 8.7: the neural network error values increase sharply for a small number of connections towards the top of the distribution. The high errors produced by the neural network appear also in Table 8.1, where the standard deviation of the relative error for the neural network model is relatively high compared to its average.

The statistical t-test was applied to the two datasets to evaluate the differences between the resulting relative errors. The two models aimed to reproduce the same actual values, fact that allowed for the resulting samples to be paired. As a result, the t-test used was similar to the one used in Chapter 5. The hypothesis made was that the neural network model produced lower relative errors in comparison to the mathematical model when estimating the actual values for the duration of TCP connections. Using equations 5.1 and 5.2 and the data from Figure 8.8, the resulting values were: $\overline{D} = 0.223293$, $s_{\overline{D}} = 0.00461539$, and $t = \frac{\overline{D}}{s_D} = 60.414$. The obtained t value is situated beyond the 0.1% probability, $P_{0.1\%} = 3.2905$, for a measurement with ∞ degrees of freedom. The conclusion drawn was that, overall, the neural network model produced estimates of the connection duration with relative errors lower by 22.3% in comparison to the mathematical model, with 99.9% confidence limits of 21.3% and 23.2%.

The results obtained from the *NS* dataset indicated that neural networks may provide better performance estimate, compared to mathematical models, of TCP transfers that encountered no losses. However, the other main objective of the method is to provide a robust estimator. This is why the next step was to apply and test the neural network on a more realistic environment, i.e. the dataset produced from the RYL connections.

8.4.2 The RYL dataset

The training process was similar and used the same dataset with the preliminary tests run in section 8.3. Unlike the *NS* traces, the senders from this dataset had different values for the initial congestion window and, as a result, the 3-6-3-1 network, displayed in Figure 8.1, was used in the process. The dataset used was the same with the one from section 8.3, produced by joining the two rounds of experiments (from autumn 2001 and spring 2002).

The network was trained for 10000 cycles and it reached a minimum of 0.00658795 for MSE after 9230 cycles.

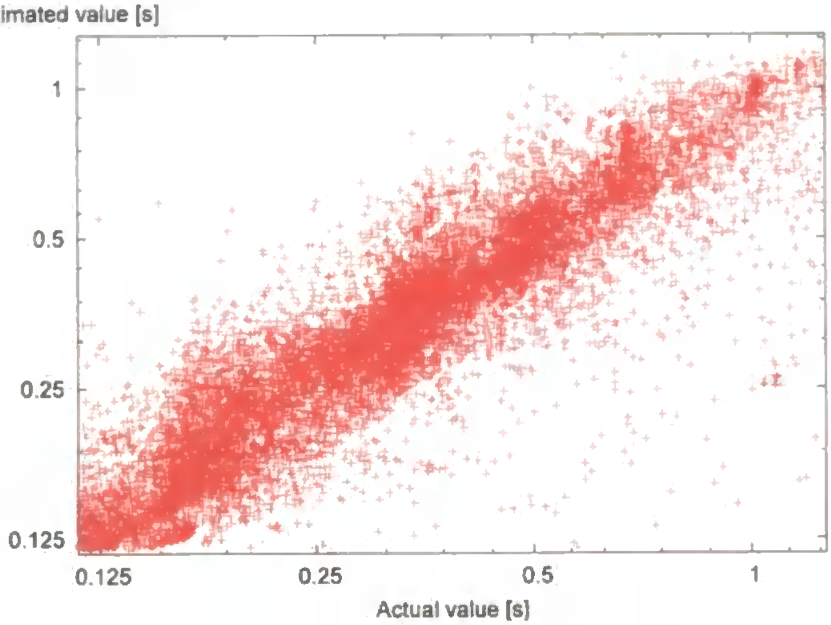


Figure 8.9 Plot of the real values vs. estimated values, as resulting from the RYL dataset

It is visible from scatter shown in Figure 8.9 that the accuracy of the neural network decreased. However, the neural network still performed better than the mathematical model. The average figures for the two models are presented below in Table 8.2

Model	Average relative error	Stdev. of relative error	Correlation
Mathematical	0.58665	0.458046	0.834741
Neural network	0.179934	0.214648	0.899387

Table 8.2 Comparison of the resulting average figures for the RYL dataset, using the mathematical and the neural network model

Finally, a plot of the relative error distributions, as produced by the two models, is shown in Figure 8.10. Although this is less visible in the graph below, the neural model has higher error values than the mathematical model at the top of the distribution. This is again due to the lack of examples at the edges of the transfer duration domain.

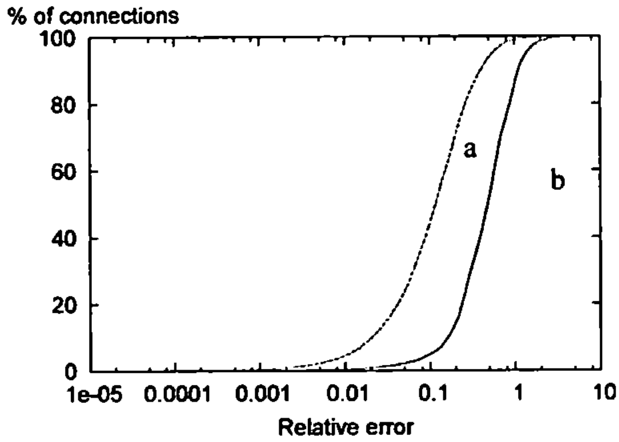


Figure 8.10 Error distribution for the RYL dataset using the (a) neural network model and (b) mathematical model

The average figures and the graphical representation of the resulting errors were also confirmed by the statistical t-test. The hypothesis and the test used were the same with the ones from section 8.4.1. In this case, the resulting values were: $\overline{D} = 0.406715$, $s_{\overline{D}} = 0.00349869$, and $t = \frac{\overline{D}}{s_D} = 116.248$. The obtained t value is situated beyond the 0.1% probability, $P_{0.1\%} = 3.2905$, for a measurement with ∞ degrees of freedom. The conclusion drawn was that, overall, the neural network model produced estimates of the connection duration with relative errors lower by 40.6% in comparison to the mathematical model, with 99.9% confidence limits of 39.7% and 41.5%.

It may be noticed from Table 8.2 that the resulting figure for MSE does not match the figure obtained during the preliminary tests. This is due to the fact that the two training sessions used different seeds, generated at the beginning of the batch of experiments for the preliminary tests and at the beginning of the training session from this section. These generated seeds were used, in each case, to initialise the network and to randomise the examples. A separate batch of 100 training sessions was run to determine what results will be obtained when using other values as seeds while keeping the same training parameters. The batch was run using random values for the seeds but the same values for training parameters: $(\eta,\tau,\mu,c)=(0.03,0.0,0.3,0.1)$. The random seeds were obtained using the same function as previous experiments, *setseed(x),* which, when invoked with an empty argument, provides a random value using the clock of the system. The result of these sessions may be seen in Figure 8.11. The figure indicates that the accuracy of the model depends rather heavily on the initialisation values for the links in the neural networks, at the beginning of the experiment, and on the order that the samples are fed to the neural network, throughout the training. However, the graph still indicates that these values tend to lead to low values in the learning process, with an average value for MSE of 0.00681869 for the entire batch.

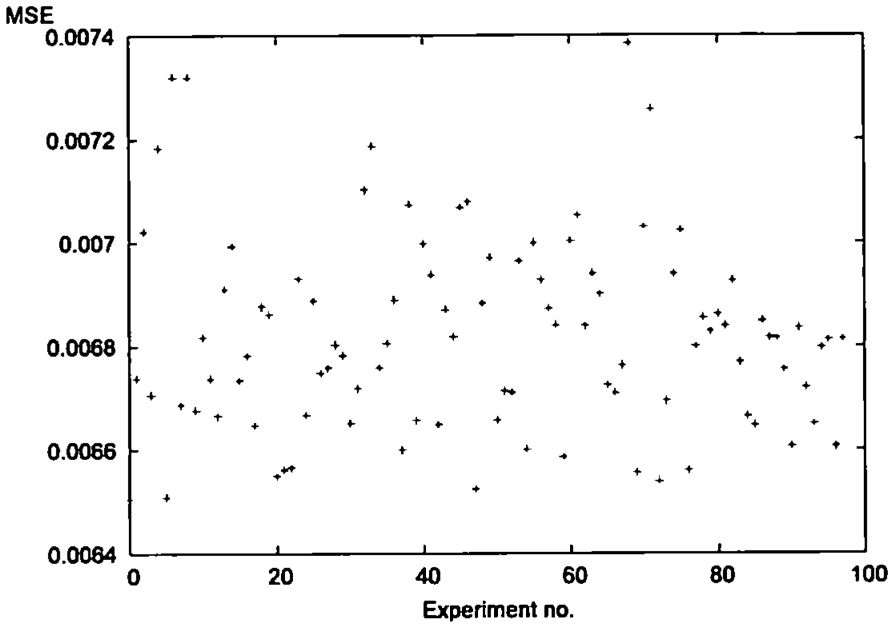


Figure 8.11 Results from 100 training sessions

This batch indicates that accuracy depends strongly on the initialisation values used. This would impact on the resulting figures from training, but it was decided to limit the scope of this study to the qualitative results of the analysis. One of the reasons behind this decision was the duration of the tests: a single exhaustive training[1] (10000 cycles) experiment lasted approximately 20-30 minutes. Each of the batch tests from section 8.3.3 required 100 experiments (based on modifying two parameters, each of them taking 10 values); running each combination of values for 100 times, as in the experiment above, would have required 150-200 days. It was, therefore, decided to determine mainly whether a neural network solution would provide better results compared to the mathematical model, using only a first degree of tuning. However, this area appeared to be an interesting direction to follow and, therefore, it is highlighted as one of the potential directions of research in the further work section.

[1] Applying early stopping would only halved the resulting value, while could have impacted on slow-converging combinations.

The analysis of the RYL traces showed that the trained neural network performed better than the mathematical model. The final test was to check whether the neural model may be generalised for backbone traces.

8.4.3 Generalisation – the UoP dataset

This dataset was expected to provide the full generalisation for the efficiency of the method. With the *NS* dataset, the senders and receivers all had the same behaviour (not influenced by the attempts to vary it); with the RYL dataset, the senders varied and used unknown implementations, but the receiver used the same TCP implementation throughout the experiments (the TCP/IP implementation from SuSE Linux). In comparison with these two datasets, the UoP data resulted from connections with unknown entities at both ends, using most likely a variety of implementations. The only indication to narrow the variety of receivers was that the typical host at University of Plymouth was running the Microsoft Windows 2000 (professional edition) operating system at the time when the captures were made. However, there were several people, particularly researchers, who had different Linux distributions installed on their computers.

The dataset used was generated from a 1-hour trace and had 21629 samples, reduced to 18545 after filtering. The dataset was applied to a 3-6-3-1 neural network (such as the one from Figure 8.1) for 10000 cycles, reaching a minimum MSE of 0.0133164 after 1770 cycles. The decrease in accuracy may be observed in Figure 8.12

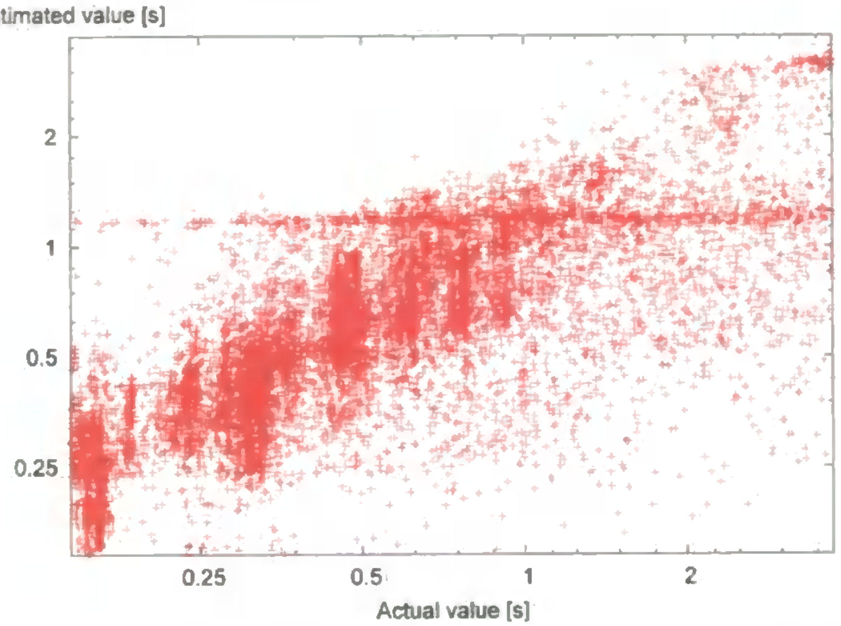


Figure 8.12 Plot of the real values vs. estimated values, as resulting from the UoP dataset

Still, even for this generalisation case, in spite of the slight overtraining that may be noticed (the visible horizontal line at the estimated value of 1.3 seconds from Figure 8.12), the neural network appears to perform better than the mathematical model. A comparison of the average figures may be seen in Table 8.3. The table also contains the results from a second training round, which was run using a more complex neural network, with 12 neurons in the first hidden layer and 6 neurons in the second hidden layer. The purpose of this additional test was to determine whether the accuracy may have been improved considerably by increasing the complexity of the neural network.

Model	Average relative error	Stdev. of relative error	Correlation
Mathematical	0.608076	0.402349	0.62782
Neural network	0.372684	0.410532	0.700272
Neural network (3-12-6-1)	0.341687	0.41052	0.722875

Table 8.3 Comparison of the resulting average figures for the UoP dataset, using the mathematical and the neural network model

It may be observed from the above table that the difference between the mathematical and neural models is smaller in comparison with the previous datasets. Nevertheless, the average error produced by the mathematical model is approximately 60% higher than the one resulting from the neural model. The results also indicate that the improvement provided by a more complex neural network, i.e. the 3-12-6-1 network, is only marginal. Finally, the comparison of the relative errors, as obtained from the neural network model and the mathematical model, throughout the dataset is presented in Figure 8.13.

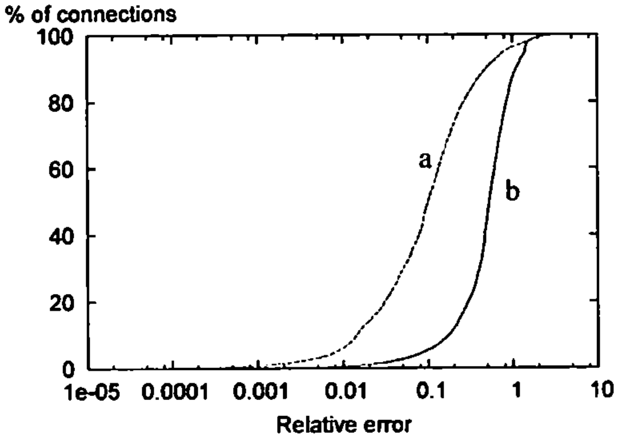


Figure 8.13 Error distribution for the UoP dataset using the (a) neural network model and (b) mathematical model

The figure confirms the average results, showing that the accuracy of the neural model is higher than the one resulting from the mathematical model. In addition, the statistical t-test produced the following values: $\overline{D} = 0.393518$, $s_{\overline{D}} = 0.00363978$, and $t = \frac{\overline{D}}{s_D} = 108.116$. The figures indicated that the errors from the neural network model were 39.3% lower in comparison to the mathematical model, with 99.9% confidence limits of 38.4% and 40.2%.

8.5 Preliminary tests - Connections with losses

The preliminary tests for connections with losses differed from the procedure from connections without losses due to data availability. After applying all filters to the three traces collected, NS, RYL, and UoP, it appeared that none of them had enough data to train the neural network. This problem could be resolved for the *NS* traces by simulating more network structures and for the UoP traces by collecting more data. Unlike the two other cases, generating data for the RYL connections was more tedious, involving slow generation of a limited number of queries (slow – to allow traces time to finish and limited - preferably non-threaded - to avoid raising network alarms). In addition, the downloaded objects during the RYL experiments were small-sized, a fact that further reduced the possibility to lead to loss events for low levels of loss rates. The two combined RYL traces had only 142 usable samples of connections with losses after filtering. This is why, rather than using the RYL traces, the preliminary tests for connections with losses were based on larger UoP traces. A set of 5 traces, each lasting for approximately 30 minutes, were combined to obtain a larger dataset, encompassing almost 10000 samples after filtering.

8.5.1 Neural network structure and stopping criteria

The design of the neural network structure used some of the conclusions drawn from section 8.3. The main problem faced was to balance the complexity of the network with the amount of data available. The overview made in section 2.5 showed that the mathematical model for connections with losses is much more complex than that for slow start. Similarly, a neural model for connections with losses might require a more complex network structure, e.g. more hidden nodes, compared with the one produced in section 8.3.1. On the other hand, the availability of data led to a smaller network, which could be within the asymptotic convergence region (as defined in 8.3.2) and which would not require splitting the data into testing and training subsets. The preferred solution was to use a structure with two hidden layers, but to limit its size in order to avoid the danger of overtraining it. The (theoretical) minimum number of samples/dataset was set to 5000, leading to ~160 free parameters in the network.

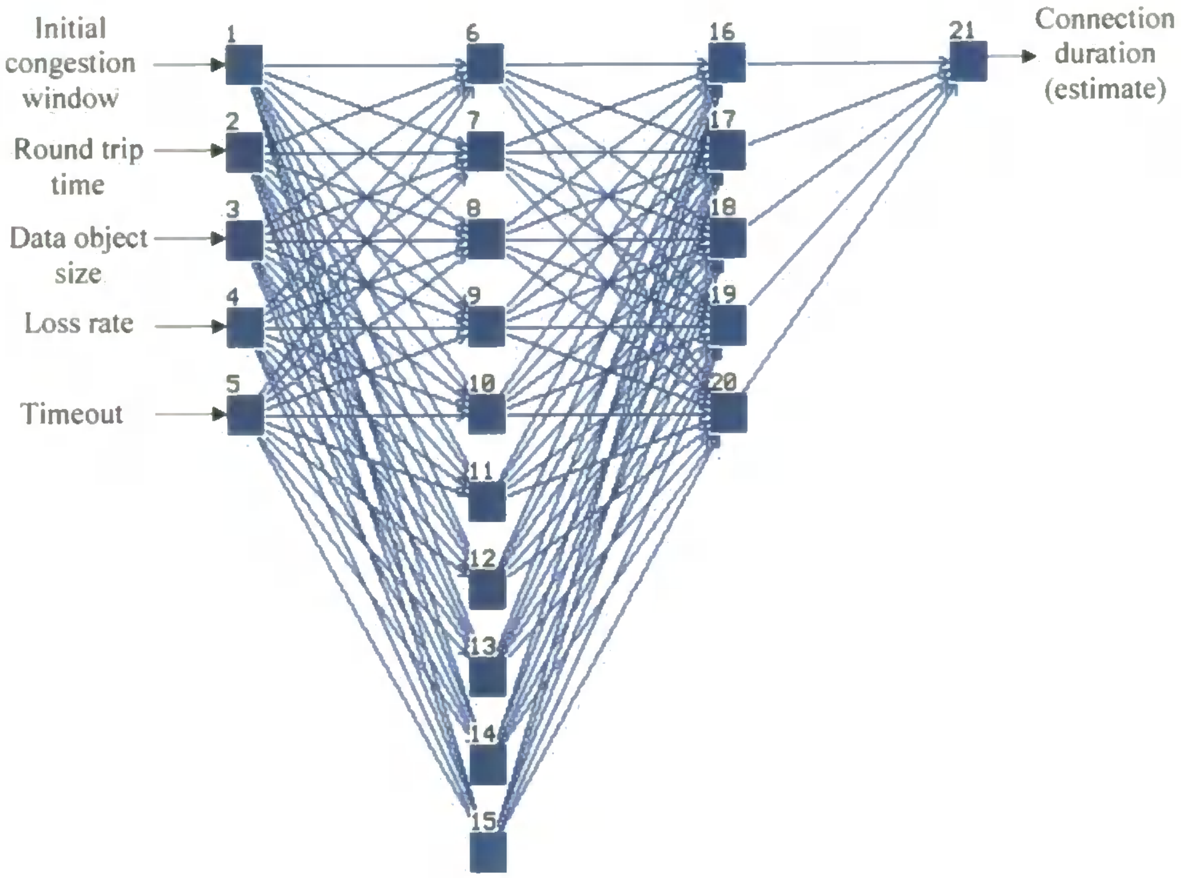


Figure 8.14 - The inputs, output, and the structure of the 5-10-5-1 neural network used during the preliminary tests for connections with losses

The neural network chosen to run the preliminary tests was a scaled-up version of that used to train no-loss connections, as shown in Figure 8.14. The 5 inputs are the ones presented in Figure 7.5 – connection and network parameters (initial congestion window, round trip time, data object size, loss rate and timeout duration) – same applying for the output – the estimated duration of the connection. The chosen network was fully connected and included two hidden layers, with 10 neurons on the first layer and 5 neurons on the second one, to increase its learning capabilities. The network pictured above has 121 free parameters, which, based on [Amari *et al* 1996], would require only 3630 samples in the dataset to avoid overtraining.

A short test was run to observe the evolution of the chosen neural network when applied to the combined UoP dataset. The test, similar to the one from 8.3.2, compared the evolution of accuracy for the test and train datasets. The aim was to determine whether the network is simple enough and whether the combination of the two (dataset neural network) obeys the conclusions from [Amari *et al* 1996].

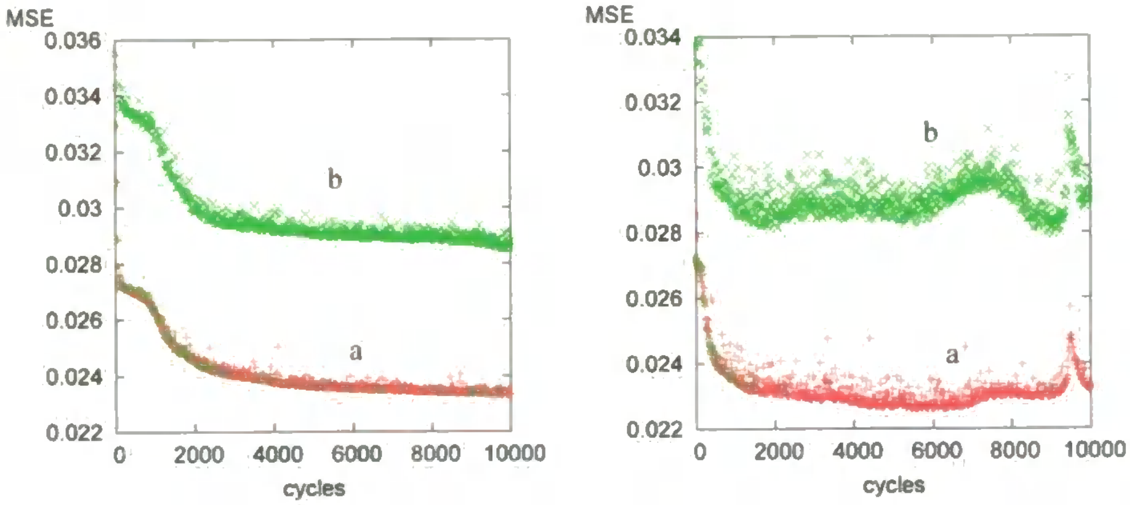


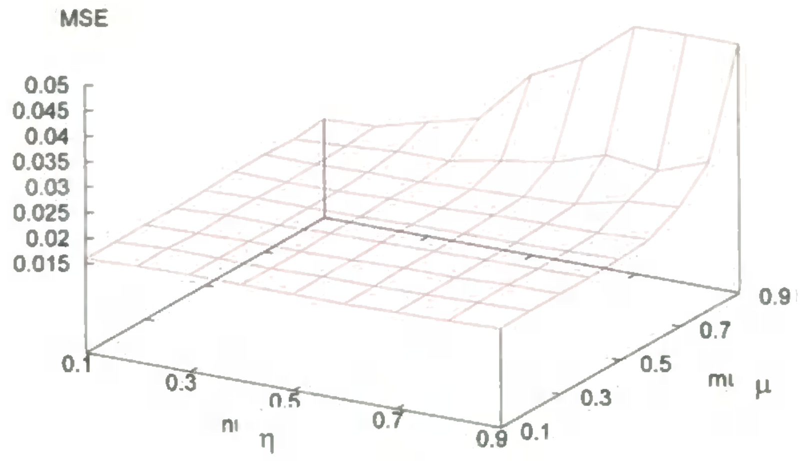
Figure 8.15 - MSE=f(cycles) for exhaustive training of the UoPx dataset, with a 6% testing (a) and 94% training (b) split, using η=0.01, τ=0.0, μ=0.1, c=0.1 (left) and η=0.1, τ=0.0, μ=0.1, c=0.1 (right)

The test used a different test-train ratio, due to the different structure of the network used. The neural network from Figure 8.14 imposed a value of 0.642 for the ι ratio (equation 8.2). The results of the test are shown in Figure 8.15 for two sets of parameters. It can be observed that, as expected, the accuracy of the test subset has the same evolution as the one from the train subset. This leads to the conclusion that the given dataset may be used for exhaustive training of the neural network from Figure 8.14 without overtraining it.

8.5.2 The optimum parameters for a set network

The next phase of preliminary tests followed the procedure from 8.3. After establishing a network structure and the stopping criteria, the network was trained using the extended UoP subset that contained connections with losses. The training ran in two batches, by varying the values for the training rate η and the momentum μ.

In the first batch, both η and μ were varied in the [0.1;0.9] interval with a 0.1 step, while the other two parameters were kept constant (τ=0.0, c=0.1). The second batch covered the [0.01;0.09] interval for η, at a 0.01 step, with μ varied in the same interval [0.1;0.9] and τ and c maintained constant to the same values as before. The results from these batches are presented in Figure 8.16



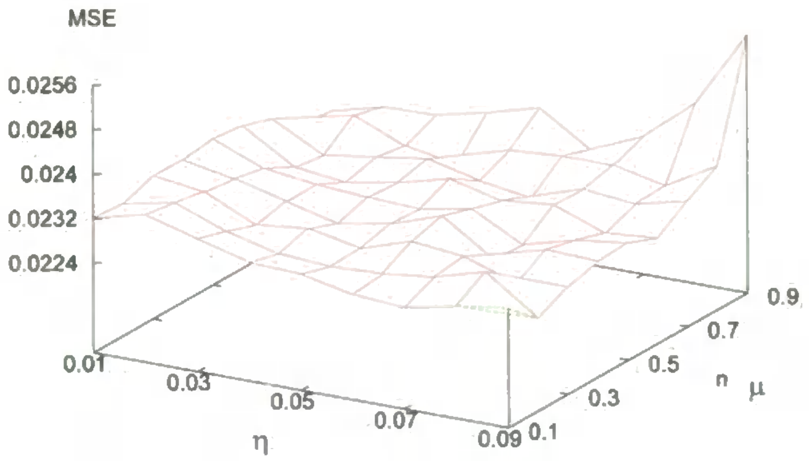


Figure 8.16 - MSE=f(η,μ) for (top) η∈[0.1;0.9] and μ∈[0.1;0.9] and (bottom) η∈[0.1;0.9] and μ∈[0.01;0.09] (τ=0.0, c=0.1)

The graphs led to similar conclusions to the ones from section 8.3. First, it may be noticed that the MSE decreases when η and μ decrease for the [0.1;0.9] interval. Secondly, the decrease is not monotonic for η∈[0.01;0.09]; these variations may have been caused by a combination of the following three factors: the network reached its limits, the low convergence speed did not allow the network to train fully, or the value of the initial seed influenced the results. The first factor, MSE having only small variations, can be one of the causes if considering the small variance in results. The standard deviation of the MSE values from Figure 8.16 (bottom) was only 0.00037, and the [MSE_{min};MSE_{max}] interval was only 0.002893. The interval size is, however, comparable with the one obtained in the batch associated with Figure 8.6 which, although not monotonous, exhibits a trend. The second factor, dependence on seed, will be considered later on using a similar experiment with the one that produced Figure 8.11. Finally, the third factor is also likely to influence the results: the minimum MSE was obtained throughout the $\eta\in[0.01;0.09]$ batch after an average 4673 cycles, compared with only 2988 cycles for the batch behind Figure 8.6. It was also observed while analysing the convergence speed that the training sessions with low learning rates achieved the minimum MSE near the exhaustive training limit (10000 cycles), which suggests that the network may have been insufficiently trained in these cases. This explains the relatively high values for MSE for low learning rates, visible in Figure 8.16 (bottom).

After running the two batches, the minimum MSE was 0.0225312, obtained for $(\eta,\tau,\mu,c)=(0.07,0.0,0.5,0.1)$, obtained after 9100 cycles. This combination of variables was used for the remainder of the loss subset validation section.

8.5.3 Data scaling

The MSE results obtained during the preliminary tests using the loss subset appeared to be less encouraging than the ones from the no-loss subset. In order to provide better results, additional effort was put into improving the other steps of the analysis, particularly the ones from the pre-processing stage. Data scaling was considered to be one of the steps that may be improved, considering the fact that the duration of connections ranged between [0.1s;12.05s] even after filtering. With an approximate 1/100 ratio between the extremities of the output variable, it is apparent that linear scaling will lead to high relative errors for lower values. In [Nagendra 1998] it was suggested that other scaling algorithms may perform better than linear scaling: logarithmic scaling may help to compact larger intervals or values, while exponential is suitable for small intervals or values. Further, [Swingler 1996] proposes *softmax* scaling as a possible alternative, for a statistically spread variable. This is why all four types of scaling (linear, logarithmic, exponential, and softmax) were applied to the data in order to provide a better fit it in the neural network, using the (η,τ,μ,c) combination from section 8.5.2. The distributions obtained are presented in Figure 8.17.

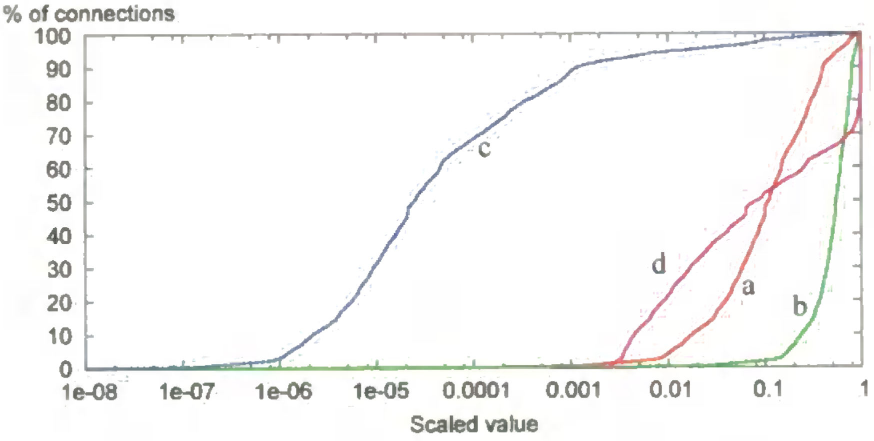


Figure 8.17 - Distributions of values after using (a) linear, (b) logarithmic, (c) exponential, and (d) softmax scaling to the UoP trace loss subset

The above figure indicates an improvement in the distribution of scaled values when using logarithmic scaling, as it compacts the interval of the variable to predict. At the other extreme is the exponential mapping, which cannot be used due to the fact that it expands the range of the scaled variable. A 10000 cycles exhaustive training session was run on each of the four obtained (scaled) datasets to determine which of them performs better. The cumulative distributions for relative errors obtained after the four training sessions is presented in Figure 8.18

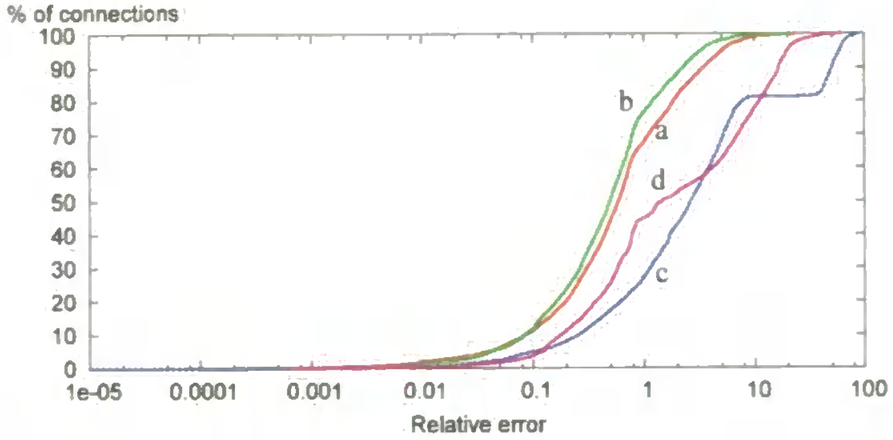


Figure 8.18 Cumulative distribution of relative error as resulting after applying (a) linear, (b) logarithmic, (c) exponential, and (d) softmax scaling to the UoP trace loss subset

It may be seen that the logarithmic scaling provides slightly better results than the linear scaling: the average error decreased from 1.307 for linear to 0.913 when using the logarithmic conversion. The other two scaling methods produced worse MSE figures.

8.6 Validation tests –connections with losses

8.6.1 NS traces

The *NS* loss subset had 3396 samples (after filtering) and it was generated using 200 rounds of simulations (in comparison, the dataset used for in 8.4.1 had only 501 samples, even before filtering, and was generated using 100 rounds). The conditions of the simulations had to be changed only slightly to produce more connections with losses: the only difference between the two batches of simulations was that, for the first round, the simulation time was limited to 5 seconds while for the second round the simulations were allowed to run for 10 seconds.

The neural network was trained with the 4-4-2 neural network, using the combination of parameters (η,τ,μ,c)=(0.07,0.0,0.5,0.1). The network converged relatively quickly, in 4160 cycles, to an MSE of 0.0288793. Figure 8.19 shows a plot of the real values vs. the output of the neural network model

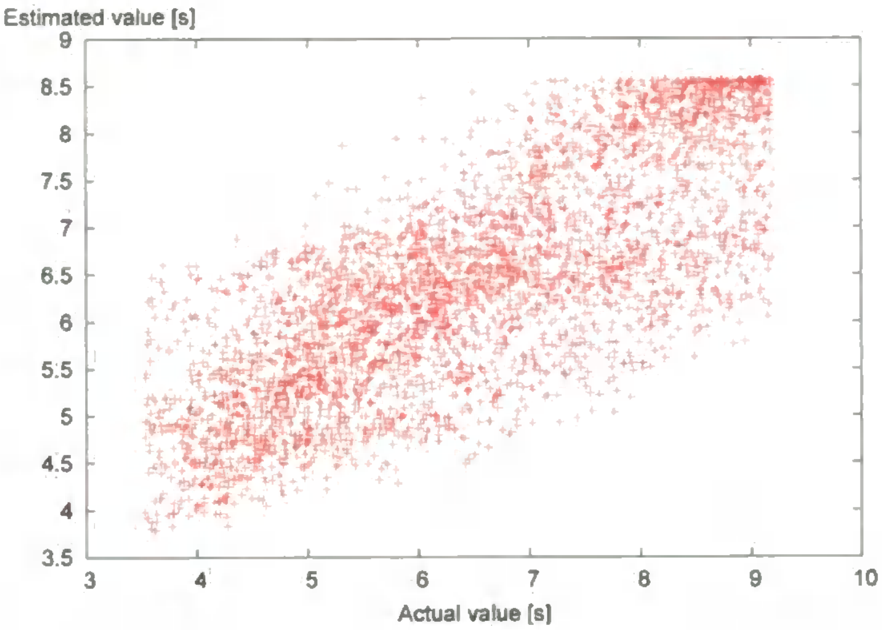


Figure 8.19 - Plot of the real values vs. the neural network estimated values, as resulting from the *NS* loss subset

The subset was applied in parallel to an implementation of the mathematical model in order to compare the accuracy of the two solutions. The comparison was made, similar to the no-loss section, by building the cumulative distribution of the relative errors, resulting from the two models. The resulting distributions are presented in Figure 8.20

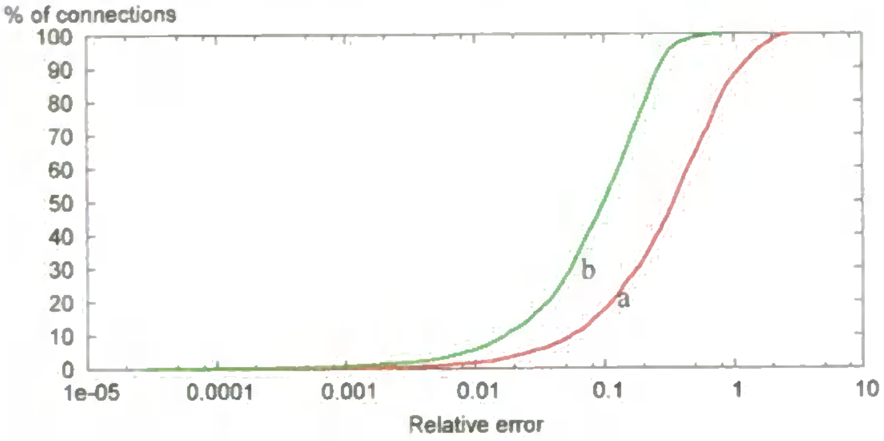


Figure 8.20 - Cumulative distributions of the relative error resulting from using the (a) mathematical model and (b) neural network model with the *NS* loss subset

The neural network outperformed the mathematical model throughout the dataset, a fact visible in the average values shown below in Table 8.4.

Model	Average relative error	Stdev. of relative error	Correlation
Mathematical	0.488822	0.469931	0.429845
Neural network	0.126957	0.108608	0.761171

Table 8.4 Comparison of the resulting average figures for the *NS* loss subset, using the mathematical and the neural network model

The statistical t-test produced slightly lower values than the comparison run for the no-loss subset: $\overline{D} = 0.368944$, $s_{\overline{D}} = 0.00851177$, but the t-value obtained, $t = \frac{\overline{D}}{s_D} = 43.3452$, is still above the 0.01% threshold $P_{0.1\%} = 3.2905$. The conclusion was that the errors from the neural network model were 36.8% lower in comparison to the mathematical model, with 99.9% confidence limits of 34.7% and 39.1%.

8.6.2 UoP traces

The validation test repeated the optimum training session from 8.5.2. The UoP subset containing only connections that encountered losses, counting 5991 samples, was trained with the 5-10-5-1 neural network, using the combination of parameters (η,τ,μ,c)=(0.07,0.0,0.5,0.1). Figure 8.21 shows a comparison between the real values and the output of the neural network model.

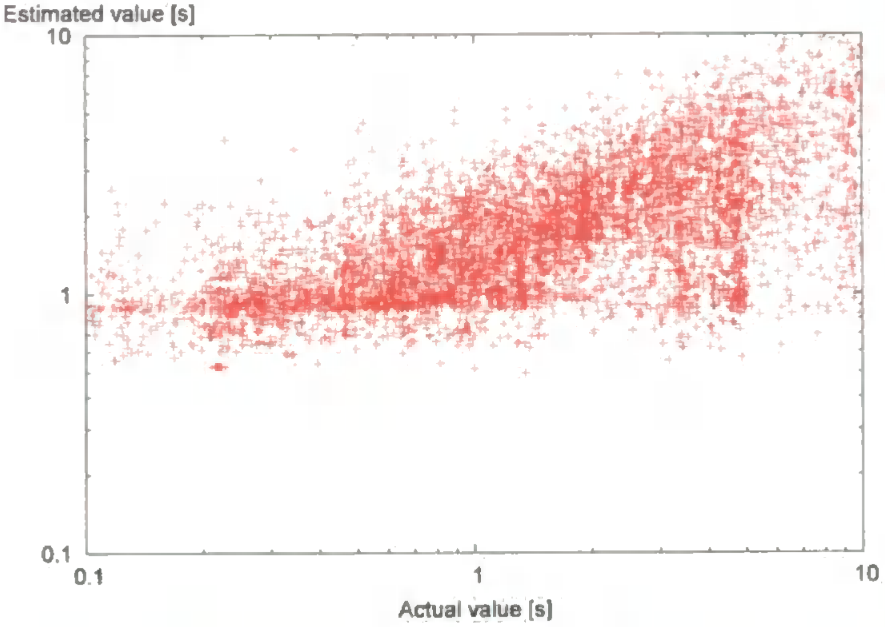


Figure 8.21 - Plot of the real values vs. the neural network estimated values, as resulting from the UoP loss subset

It may be seen that the above plot does not illustrate a very good mapping between the actual and estimated values. The mapping also seems to be incomplete from a domain perspective: a large proportion of the actual values are situated in the [0.1s;1.0s] domain, while the estimated value seems to have a lower-bound around the value of 1s. To determine whether the prediction is accurate for the rest of the values, a plot of the relative errors vs. the actual values was produced in Figure 8.22

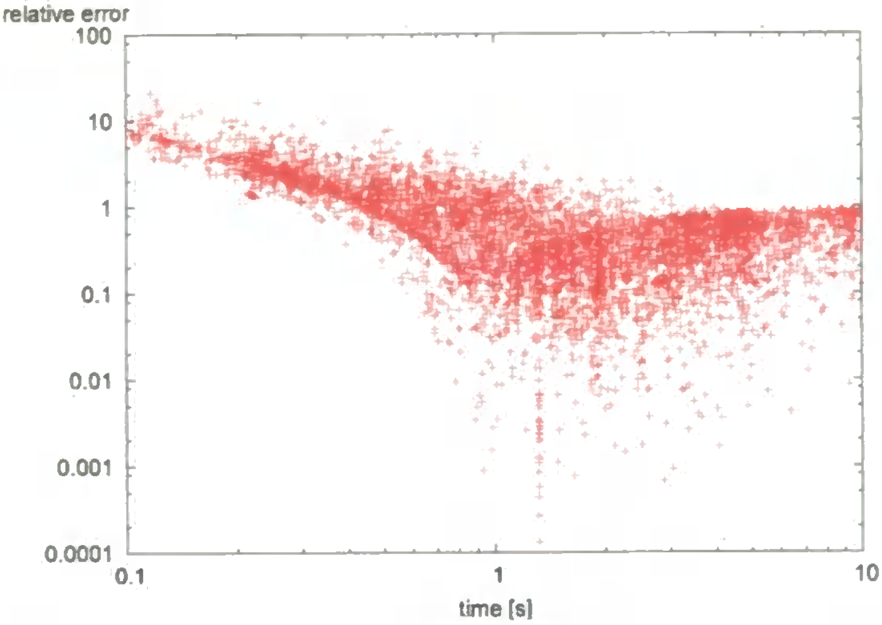


Figure 8.22 - Plot of the real values vs. the resulting prediction relative error, as resulting from the UoP loss subset

The plot shows that, although the method performs badly for connections which lasted less than 1 second, it provides better accuracy for the rest of data, with virtually all relative errors below 1. Similar conclusions may be drawn from the distributions of the actual and estimated values; the diagram with the two variables is shown below in Figure 8.23

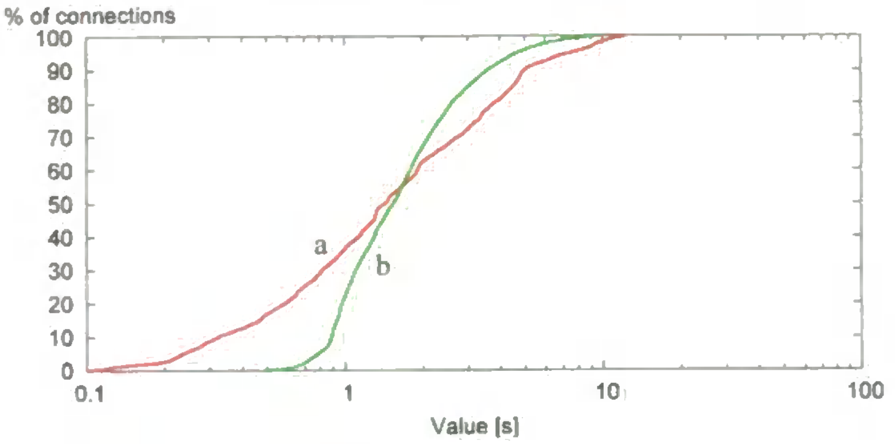


Figure 8.23 - Cumulative distributions of the (a) actual values and (b) estimated values from the UoP loss subset

The above distributions reflect the fact that the neural networks do not manage, in spite of the logarithmic scaling, to stretch over the entire domain of the estimated values. As expected, this improper mapping leads to worse results when calculating the relative error of the prediction method and comparing it with the mathematical model. The accuracy of the two models is presented in Figure 8.24

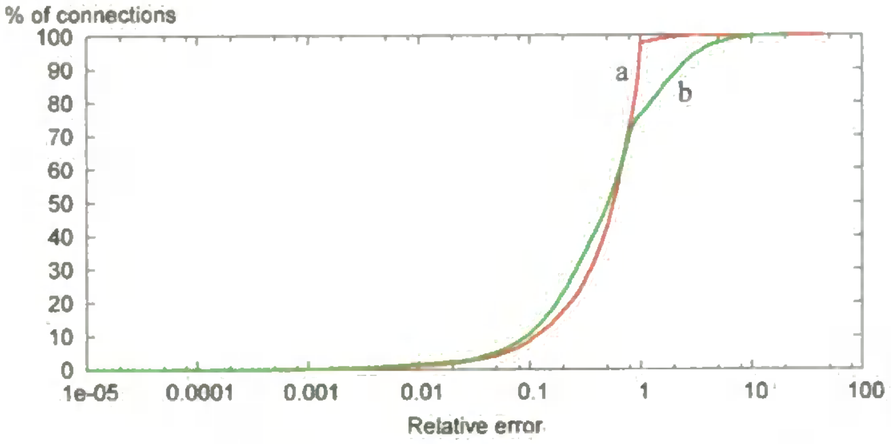


Figure 8.24 Accuracy of the (b) NN model and (a) mathematical model for the UoP loss subset

The figure shows that the neural network manages to outperform the mathematical model for more than half of the connections. This is likely to be the region unaffected by the mapping problem discussed earlier. Unfortunately, while the mathematical model appears to be limited to a maximum relative error of 1, the neural network does increase above that level. These errors, visible in Figure 8.22, were all at least partially due to the limited efficiency of the neural network. The difference between the two models appears in their average values too, shown in Table 8.5:

Model	Average relative error	Stdev. of relative error	Correlation
Mathematical	0.584002	0.748376	0.467215
Neural network	0.915117	1.34159	0.604007

Table 8.5 Comparison of the resulting average figures for the UoP loss subset, using the mathematical and the neural network model

The statistical t-test confirmed the reduced accuracy for the neural model. The hypothesis made was that the neural network model led to higher relative errors than the mathematical model. The resulting figures from the calculations were: $\overline{D} = 0.331114$, $s_{\overline{D}} = 0.0183278$, which produced a t-value of $t = \frac{\overline{D}}{s_D} = 18.0663$. It was therefore concluded that was that the neural network model produced relative errors which were 33.1% higher in comparison to the mathematical model, with 99.9% confidence limits of 28.3% and 37.8%.

Correlating Figure 8.24 with Figure 8.22, it can be concluded that the error is due to the erroneous approximation of the <1s connections. The only solution to ameliorate this error would be to use a scaling solution that would outperform the results obtained from logarithmic scaling or to split the duration of the variable into decades and produce separate models for each decade. Both these solutions are mentioned in Chapter 9, as part of the future work.

8.7 Applications

As an overall conclusion from the performed validation tests, the proposed neural-based model provides a better alternative to mathematical models in terms of accuracy. The increased accuracy, together with the robustness of the method, provides a better described relationship between application performance and the conditions that influence it. This improvement opens new avenues in the areas that relate to performance provisioning, such as:

- Network planning. A pre-trained neural network can be employed in preliminary stages of network planning, to estimate the application performance based on certain sets of network conditions and specific endpoint implementations.
- Network control. The model may be use as a predictive element in network management schemes when varying the parameters of the network. For example, it may provide a set of alternatives to balance the delay and loss introduced by routing queues in order to maintain certain performance for the above applications.
- TCP Implementations testing. The presented approach may be used to study the application performance provided by new types of TCP implementations. Through its adaptive character, the neural-based method may be used to compare the efficiency of new implementations versus traditional ones.

8.8 Summary

This chapter presented the validation tests used to benchmark the TCP performance modelling method proposed in Chapter 7. The validation studied the accuracy of the proposed neural network model when applied on two types of traces: purely synthetic TCP connections and TCP connections captured from real traffic.

The analysis started by discussing the obstacles met during the data collection stage and the issues that were likely to impair the model accuracy, but which were identified and minimised during the data analysis stage.

The available traces from each source (*NS*-generated, RYL, and UoP) were each split into two subsets each, one containing connections with losses and one containing connections without losses. The loss-no loss split was required due to the differences in the models for connections with and without losses. The two types of data, with or without losses, were studied in separate rounds of tests. Each round commenced with a preliminary study that determined best values for the optimum for the training-testing split, the training algorithm and neural network structure to use, the stopping criteria, and the optimum neural network learning parameters

The validation tests consisted of using the resulting parameters to train the respective dataset, then to test its accuracy. The results were then observed and compared with the output of a mathematical model implementation. The validation tests have shown that the proposed model provides an overall better accuracy when compared against the mathematical model using the three factors mentioned above. The tests performed on the no-loss subsets indicated a nearly ten-fold improvement of the average relative error in the case of *NS*-generated data between the mathematical and the neural-based model. The improvement was also high in the case of RYL traces, where the relative error obtained with the mathematical model decreased by approximately 70% when applying the neural model, and by 40% in the case of real traffic collected from the UoP backbone. The results obtained from the trace subsets that included connections with losses were two-fold: the *NS*-generated trace led to a 75% reduction of the overall relative error between the two models. Unfortunately, the UoP loss subset led to better prediction results when applying the mathematical model. However, after analysing the plots of the values and of the corresponding relative errors (Figure 8.21 and Figure 8.22), it was concluded that the poor accuracy may have been caused by limitations of the neural network. This conclusion was sustained also by the distribution of the relative error for the two models (Figurė 8.24).

The last part of the chapter listed some of the possible applications of the neural-based model, ranging from network planning to network control and TCP implementation testing.

This chapter concludes the description of the research undertaken as part of this research project. The following chapter will present the achievements of the research together with the limitations encountered, and will identify several promising directions the future work arising from this project.

Chapter 9. Review, future work, and conclusion

9.1 Achievements

In this thesis, novel research in two areas has been presented: network monitoring and performance prediction. The two areas relate strongly to recent trends of moving Internet access provisioning towards quality provisioning. The results provide the first two steps in this migration: evaluate current end-to-end network performance and relate the observed network characteristics to the resulting performance.

The research programme led to six significant outcomes:

1. A detailed understanding of the current state of the art in areas closely related to traffic analysis and performance modelling

   The research illustrated the emerging need for Internet quality provisioning, in spite of the lack of support from IP, its core protocol. As a result of the emerging need for quality, traffic performance evaluation and monitoring was identified as an important research theme. Based on these observations, a taxonomy of existing network performance measurement methods was produced, each category covering various aspects of inferring the network characteristics. The analysis identified the limitations of these methods and gathered them in a sum of characteristics relevant for the current Internet conditions and complexity.

   On a separate stream, the studies identified in the area of TCP modelling followed a single, mathematical-based, approach. The tests to validate such models were typically performed using synthetic environments or controlled endpoints and were based on long data transfers. In addition from the sound mathematical proof, the models found were suitable for steady-state transfers, implying long connections – which are unlikely to exist in current web traffic conditions.

2. A novel monitoring technique that allows single-point, non-intrusive, online TCP performance analysis, which aims to overcome some of the disadvantages of existing techniques, typically based on intrusive methods.

The technique used elements from existing tools to propose a novel method of TCP analysis that would bring together a comprehensive subset of the current requirements for traffic performance monitoring. The method included several novel elements, such as TCP timestamp analysis and congestion window evaluation. The TCP timestamp then provided the means for non-intrusive inference of bottleneck bandwidth for the monitored traffic.

The novelty of the proposed traffic analysis method was to provide comprehensive information about the end-to-end parameters of the network path transited by the packets from a single point of capture, without injecting any traffic. The only source of information was the packet arrival sequence for each flow combined, where necessary, with assumptions about the network characteristics or/and endpoint behaviour. The analysis also supplied means of localising any degradation/change in the network conditions by splitting the end-to-end path into two sub-paths, between the monitor and each of the endpoints, based on the analysed traffic. The approach was designed to function online, underlining the importance of continuous monitoring.

3. A prototype to benchmark the efficiency and accuracy of the proposed method.

The software prototype was developed under Linux due to the flexibility offered by the OS environment. The software was tested and adjusted throughout its development in order to interpret correctly the results of the observed packet arrival sequence. The program was developed specifically for online analysis but, due to the lack of large sources of traffic and the need for repeatability, it included built-in support for trace analysis, which constituted its main application during this research programme.

The traffic analysis method was applied to traces of traffic produced in a controlled environment; the purpose was to determine whether the inferred parameters match the actual network and endpoint parameters. The method was tested using two types of environments: emulated and simulated network conditions. In spite of some limitations of the environments used, the validation tests indicated that the method is accurate when estimating the end-to-end network parameters as inferred by the endpoints rather than the actual values. While the delay values were well approximated, some of the loss figures incurred significant errors due to the TCP endpoints erroneously inferring loss events. The proposed method was therefore extended to identify, subject to the position of the monitor, some of the erroneous retransmission events. In addition, statistical analysis revealed with a confidence factor of at least 99.9% that, for the analysed packet traces, TCP timestamp-based measurements provide more accurate RTT estimation when compared to acknowledgment-based measurements.

4. A viewpoint of end-to-end Internet paths characteristics, as seen from the perspective of the HTTP traffic collected from one point (the University of Plymouth) situated at the edge of the network, using exclusively non-intrusive analysis.

Continuing from the encouraging benchmarking results, the method was successfully applied on a combination of semi-controlled and real traffic packet traces in order to provide a snapshot of end-to-end characteristics of Internet paths and average values of TCP connection parameters at a specific moment, as observed through the perspective of TCP data transfers,. In spite of a wide variety of previous research, this study included several elements of novelty:

- it analysed traffic produced by an edge network consisting mainly of clients;
- it led to a detailed analysis of the typical figures for TCP congestion window;
- it observed the distribution of end-to-end path characteristics, bottleneck bandwidth in particular, as encountered by actual TCP clients from the edge network studied
- it performed a statistical evaluation of how the observed end-to-end path parameter distributions evolved in time.

The analysis focused on HTTP transfers due to the wide availability of such traffic, both to retrieve in the semi-controlled traces and to capture real traffic transfers. While investigating the results, the method was adjusted and improved in order to accommodate specific network and application protocol behaviour, while maintaining its robustness.

The traffic study was run offline on collected packet traces in order to provide repeatable results, available for further analysis. The trace collection included semi-controlled random web page retrievals in the first phase, followed by backbone data collection towards the end. The study concluded that, from the point of collection perspective, the packets from the analysed connections appeared to have been transported over network paths with high values for bottleneck bandwidth, paths that introduced low loss and delay for the transiting traffic. The conclusions were supported by average values and distribution analysis, while statistical tests were employed to investigate the variations over time for the studied parameters. The statistical tests identified changes in the end-to-end path parameters between two rounds of measurements with a statistical confidence of at least 99.9%. One of the likely causes of these changes, consisting of improvements in terms of inferred delay and bottleneck bandwidth, was a network connectivity upgrade that occurred between the two rounds of measurements. Also, the connection size analysis indicated that the typical HTTP interaction involved short-lived TCP connections. It also indicated that the conclusions from a small amount of traffic scale well for large traces analysis, as long as collection was made from similar environments, relatively closely spaced in time. Finally, certain recommendations were made, aimed to improve the efficiency of TCP under the current network conditions.

5. A novel model for inferring the relationship between TCP throughput and its influencing parameters, based on performance observed for prior connections.

   The approach used a neural network to approximate the relationship between TCP's resulting throughput and the factors that influence it, unlike previous models which used mathematical emulation of the TCP sender behaviour. The prediction used a relevant set of parameters, extracted using the traffic analysis stage: delay-related, loss-related, and TCP behaviour-related. The predicted variable in all cases was the duration of the data transfer. The prediction involved three major stages: data pre-processing, core neural network analysis, and interpretation of the results.

6. An assessment of the accuracy of the proposed model based on a series of evaluation tests, applied on various sources of traffic.

   The neural network model was tested separately on three types of packet traces: synthetic, semi-controlled, and real traffic. Two neural network models were produced, one suitable for predicting performance for connections without losses and one for predicting performance for connections that encountered losses. The relative error results have shown that the neural network led to more accurate results than the mathematical models for all types of traces in the case of connections without losses. The neural network model for connections with losses also led to better results than the mathematical model when trained and tested using the synthetic trace. The superiority of the neural network approach was confirmed in each case by statistical analysis, which involved paired T-tests applied to the relative errors produced by each model. All tests confirmed the initial conclusions, with a statistical confidence of at least 99.9%. However, the neural network failed to produce lower relative errors than the mathematical model when tested on real traffic traces which included connections with losses, issue which will be highlighted in the next section.

9.2 Limitations

There are a number of limitations of the study, as summarised below:

1. Complexity of the environments used for testing the monitoring method.

   It was observed that neither of the two environments used for evaluating the proposed traffic analysis method were fully satisfactory. The synthetic traces lacked realism in terms of endpoint interaction, such as implementation characteristics and processing delays, while the emulated environment lacked accuracy in terms of reproducing network events, such as bandwidth limitations and queuing. A hybrid approach, using the mixture of the two, may have been beneficial for the various tests performed. Unfortunately, as observed with the network emulation, the more complexity is added into the network, the more difficult it is to account for all sources of impairments.

2. Differences between the inferred network parameters and the real values

   The proposed traffic analysis technique provides a good approximation of the end-to-end path parameters as inferred by the TCP endpoints. However, the validation tests run for the method highlighted that the inferred parameters (loss in particular) may differ from the actual network parameters. Further research to establish the relationship between the TCP inferred parameters and the end-to-end path parameters would improve the accuracy of the method proposed.

3. Scalability, focus, targeted traffic, and timeliness of the end-to-end paths characteristics study

   The snapshot of the end-to-end path parameters was produced from a single endpoint, the UoP network, which had relatively good Internet connectivity. This configuration produced a collection of end-to-end paths all converging to the collection point, rather than a mesh between distinct endpoints, logically and geographically remote. More conclusive results would have been obtained, had a mesh infrastructure of points been employed in the analysis. Unfortunately, throughout the study, there was no opportunity to compare the resulting values from the UoP traffic with end-to-end results obtained from other endpoints, apart from overall results presented by other network measurement studies, which appeared to yield comparable average results. For this reason, it is difficult to evaluate how the presented results would scale for wider studies, collected from more than one edge network. However, while one of the endpoints was fixed (the collection point), the position of the remote host varied, which led to variations in the remote end of the path. On the positive side, the connectivity solution used had the advantage that it did not introduce any local bottlenecks (as a poorly connected network may have been), which led to an unbiased snapshot of the analysed end-to-end paths.

   The method used for TCP analysis, although providing detailed information about the end-to-end parameters, included no means for localising the source of the impairments, apart for the East/West split of the end-to-end path. This lack of localisation did not allow for any clustering of the results, as there were no means to aggregate any path segments. Brief path analysis using traceroute on a subset of remote hosts indicated a high number of different paths, all originating from a low number of distinct IP addresses.

The observed characteristics for the analysed paths are also subject to timeliness. While the analysis of the network traces provided a detailed view of end-to-end paths parameters as seen from the University of Plymouth viewpoint, there is no indication about how these parameters may vary in time. This limitation was actually highlighted in the study by the differences between the two rounds of experiments, as the average values for the analysed parameters changed for the traffic captured within the associated packet traces. Further research, as proposed in section 9.3, may provide indications about the timelines of such performance snapshots.

4. Neural network modelling limitations.

The proposed neural network model looked only at some of the aspects that may improve the resulting accuracy of the method. Further avenues should be explored, such as using other types of neural network, in order to optimise the results. As detailed in chapter 8, the results revealed that the neural network exhibited high errors when modelling connections with losses, which were likely to be due to limitations of the neural network modelling capability. Further exploration into the causes of these high errors will be considered as a valid direction for future research in the next section. Another limitation of the neural network study was that the mathematical model was not considered as a possible input; while this decision provided clear separation between the two approaches, usage of the mathematical inference may have yielded more accurate results.

The neural network validation was also limited in the sense that all data analysed was collected from the same environment. It is, therefore, difficult to assess the scalability of the neural network model, as no comparison term was available. In order to investigate the ability of the model to adapt to a collection of distinct environments, encompassing different sets of network conditions, further analysis of data from such environments would be necessary.

9.3 Future work

A number of promising areas for further research were identified that were outside the scope of the current study:

1. The combination of TCP analysis and HTTP analysis. This would provide advantages in two directions. The first is TCP profiling: the TCP analysis would be improved by including a level of profiling for specific TCP implementations. Although not completely accurate, there is a relationship between HTTP server implementations and corresponding TCP implementations. The server type from the HTTP headers would provide the initial information in building this correspondence. The profiling could be further strengthened using other characteristics of the TCP client, such as timestamp resolution. A second area is idle period inference: interpretation of the HTTP headers would allow clear identification of the idle periods between the download of successive objects. This would also increase the accuracy of timeout losses inference.

2. Analysis of network characteristics over time. This subject was approached in previous studies from an active perspective, but non-intrusive analysis would broaden the number of monitored paths. Also, in comparison, the non-intrusive approach would still require a single connectivity point, unlike active measurement that needs exponential increase in the number

of probes deployed over the Internet and additionally stresses the network. The information about network parameter variations can then be used to predict or, at least, bound the future characteristics of the network. A direct application of such a technique to the scope of this project would be to predict the quality of a VoIP call based on data exchange during the initial TCP signalling. This *apriori* estimation of the quality of the network may help to assess whether the application is likely to be successful (in the above example, whether the quality of the voice call will be satisfactory). The estimation may be based on either a mathematical, Markov-based, interpretation or a neural-network based model. A second, more generic application would be to study the aging speed of such network information in order to provide time-based performance prediction, rather than parameter-based.

3. Integration of the proposed monitoring method into a system that would control and improve the resulting performance. There are two possible directions of research in this area, both aiming to improve the user experience and related to dynamic management: either induce changes in the network characteristics or induce changes in the client behaviour. The first option would see the network parameters connected to a network management system. Under such a scheme, variations in a certain parameter would lead to variations in the controllable characteristics of the managed network: increase/decrease queues, modify policies at the routers, or balance traffic between alternative routes. A preferred approach for the second option, changing client behaviour, would be to apply changes transparently, using a proxy server that has knowledge of the past and current network characteristics. Such a proxy server would introduce additional delay or loss to force network decongestion; at the other extreme, it would modify the acknowledging or the congestion window increase policy to improve performance of short-lived flows over a reliable path.

4. Further research into the neural network prediction. A prime area of research should resolve

the prediction errors encountered for connections with losses. Various avenues may be explored to improve the prediction, such as deeper analysis of the connections incurring losses and combination of the neural network inputs with information resulting from the mathematical model. A second area of research relates to the TCP profiling discussed before. The neural network model would produce better accuracy if provided with some indication about specific implementation characteristics. Such characteristics should be mapped from their categorical scale onto a numerical scale in order to be presented as an input to the neural network. A third area of research would look into transferring the neural network model from network planning to network management. The model would provide management architectures with information about how network changes, e.g. increasing network delay with a certain percentage (through increasing queue sizes or redirecting traffic through alternate routes), would impact on the performance of the future transfers.

9.4 Conclusion

The concept of quality will become increasingly important as the Internet continues to evolve. This quality will be achieved not through controlled mechanisms, which are difficult to deploy throughout connected networks, but over the quality-unaware environment that IP provides. In this environment, the need to evaluate performance becomes critical, as customers want to establish whether they are provided with the quality that they pay for and, if not, to identify the weak links of their connectivity infrastructure.

This project bridges the gap between customers interested in the performance of their applications and the performance-unaware network environment. Through the proposed novel monitoring architecture, network management entities may evaluate the parameters of the live traffic flowing through the network and may take the appropriate decisions. The proposed analysis method was used within the research to produce a holistic image of Internet paths, as observed through traffic transiting a backbone collection point. Aside from the information provided, the observed network characteristics led to a series of recommendations that may improve current application performance.

With an increasing network quality awareness and need for performance throughout the Internet, the proposed work will help to evaluate the parameters of the network infrastructure and will provide a basis for improving the performance of current and emerging network applications.

References

[Acterna 2003] *, DataAnalyser 3600 homepage, http://www.acterna.com/united_kingdom/Products/descriptions/DA-3600/3600_traffic.html

[Agilent 2003] *, Agilent Advisor homepage, http://onenetworks.comms.agilent.com/lananalyzer/default.asp

[Aleksander and Morton 1990] Aleksander, I., Morton, H., "An Introduction to Neural Computing", Chapman and Hall, London, 1990

[Allman *et al* 1999] Allman, M., Paxson, V., Stevens, W., "TCP Congestion Control", Request for Comments 2581, April 1999

[Allman and Paxson 1999] Allman, M., Paxson, V., "On Estimating End-to-End Network Path Properties", SIGCOMM 99, Cambridge, Massachusetts, September 1999

[Allman and Falk, 1999] Allman M., Falk A., "On the Effective Evaluation of TCP" ACM Computer Communication Review, October 1999.

[Allman 2000] Allman, A., "A Web Server's View of the Transport Layer", ACM Computer Communication Review, 30(5), October 2000

[Amari *et al* 1996] Amari, S., Chen, T.P., Yang, H.H., "Statistical theory of overtraining – Is cross-validation asymptotically effective?", Advances in Neural Information Processing Systems, vol. 8, pp. 176-182, Cambridge, MA, MIT Press, 1996

[Amari *et al* 1997] Amari, S., Murata, N., Muller, K. R., Finke, M., and Yang, H. H., "Asymptotic Statistical Theory of Overtraining and Cross-Validation", IEEE Transactions on Neural Networks, Vol. 8, No. 5, pp. 985-996, 1997.

[AMP 2003] *, Active Measurement Project homepage, http://watt.nlanr.net/

[ARPA 1981a] *, "Internet Protocol", Request For Comments 791, September 1981

[ARPA 1981b] *, "Transmission Control Protocol", Request For Comments 793, September 1981

[Asaba *et al* 1992] Asaba, T., Claffy, K., Nakamura, O., and Murai, J., "An analysis of international academic research network traffic between Japan and other nations", Inet '92, pp. 431--440, June 1992

[Balakrishan *et al* 1997] Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M., and Katz, R., "Analyzing Stability in Wide-Area Network Performance", Proceedings ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, June 1997

[Balakrishan *et al* 1998] Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M., and Katz, R., "TCP behavior of a busy Internet server: Analysis and improvements", Proceedings of the IEEE INFOCOM, pages 252--262, March 1998

[Berners-Lee *et al* 1996] Berners-Lee, T., Fielding, R., and Frystyk, H., "Hypertext Transfer Protocol -- HTTP/1.0.", Request For Comments 1945, May 1996.

[Blake *et al* 1998] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W., Request for Comments 2475, "An Architecture for Differentiated Services", December 1998

[Bolot 1993] Bolot, J-C., "End-to-End Packet Delay and Loss Behavior in the Internet," *Proceedings of SIGCOMM '93*, pp. 289-298, September 1993

[Borella 2000] Borella, M., "Measurement and Interpretation of Packet Loss" Journal of Communications and Networking, Vol. 2, No. 2, pp. 93-102, Jun. 2000

[Borella 2003] Borella, M., "ipgrab homepage", http://ipgrab.sourceforge.net, 2003

[Bortzmeyer 2003] Bortzmeyer, S., *echoping* home page, http://echoping.sourceforge.net/

[Braden 1989] Braden, R., 'Requirements for Internet Hosts - Communication Layers', RFC 1122, Network Information Centre, SRI International, Menlo Park, CA, October 1989.

[Braden *et al* 1997] Braden, R., Zhang, L., Berson, S., Herzog, S., Jamin, S., RFC 2205, "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification", September 1997

[Bruce 2003] Bruce, A. M., *pchar* home page, http://www.employees.org/~bmah/Software/pchar/

[CAIDA 2003] *, "the Cooperative Association for Internet Data Analysis", http://www.caida.org, 2003

[Cardwell *et al* 2000] Cardwell, N., Savage, S., Anderson, T., "Modelling TCP Latency", Proceedings of IEEE INFOCOM, Tel Aviv, Israel, March 2000

[Carson 1997] Carson, M., "Application and Protocol Testing through Network Emulation", Internetworking Technologies Group, NIST, September, 1997, available from http://snad.ncsl.nist.gov/itg/*NIST Net*/slides/index.htm

[Carson 2003] Carson, M., "*NIST Net* home page", http://snad.ncsl.nist.gov/itg/*NIST Net*/

[Catania *et al* 1996] Catania, V., Ficili, G., Pallazo, S., Panno, D., "A Comparative Analysis of Fuzzy Versus Conventional Policing Mechanisms for ATM Networks", IEEE/ACM Transactions on Networking, vol. 4, no. 3, june 1996

[CERT 1999] *, CERT practice: Configure firewall packet filtering, http://www.cert.org/security-improvement/practices/p058.html, July 1999

[Cheng and Chang 1996] Cheng, R.G., Chang, C.J., "Design of a Fuzzy Traffic Controller for ATM Networks", IEEE Transactions on Networking, vol. 4, no.3, june 1996

[Chester 1990] Chester, D. L., "Why two layers are better than one", International Joint Conference on Neural Networks, vol. 1, pp. 265-268, Washington, DC, 1990

[Cochran and Snedecor 1980] Cochran, W. G., Snedecor, G. W., "Statistical methods", Iowa State University Press, 1980

[Clark 1982] Clark, D., "Window and Acknowledgment Strategy in TCP", Request For Comments 813, July 1982.

[Clevertools 2003] *, Netboys homepage, http://www.clevertools.com/products/netboys/

[CoralReef 2003] *, "CoralReef Software Suite", http://www.caida.org/tools/measurement/coralreef

[Crovella and Bestavros 1996] Crovella, M., and Bestavros, A., "Self-Similarity in WorldWideWeb Traffic: Evidence and Possible Causes," *Proceedings of SIGMETRICS '96*, Philadelphia, May 23-26, 1996

[Crowcroft and Wakeman, 1991] Crowcroft, J. and Wakeman, I. "Traffic Analysis of some UK-US Academic Network Data," *Proceedings of INET '91*, Copenhagen, June 1991

[Deering and Hinden 1998] Deering, S., Hinden, R., "Internet Protocol Version 6 (IPv6) Specification", Request for Comments 2460, December 1998

[Donnelly 2001] Donnelly S., "timestamping accuracy", e-mail to *tcptrace users* mailing list http://irg.cs.ohiou.edu/software/tcptrace/archive/0110.html, April 2001

[Doulgeris and Develekeros 1997] Doulgieris, C., Develekeros, G., "Neuro-Fuzzy Control in ATM Networks", IEEE Communications Magazine, may 1997

[Dovrolis 2003] Dovrolis, C., pathrate home page, http://www.cis.udel.edu/~dovrolis/bwmeter.html

[Evans 2001] Evans, M., "A Model for Managing Information Flow on the World Wide Web", PhD Thesis, University of Plymouth, 2001

[Fall and Varadhan 2003] Fall, K., Varadhan, K., "The ns manual", http://www.isi.edu/nsnam/ns/doc/, 2002

[Fayyad *et al*, 1996] Fayyad, U., Shapiro, G. P., Smyth, P., "From Data Mining to Knowledge Discovery in Databases", AI Magazine, Fall 1996

[Fielding *et al* 1997] Fielding, R., Irvine, U.C., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T., "Hypertext Transfer Protocol – HTTP /1.1", Request for comments 2068, January 1997

[Floyd and Fall, 1999] Floyd, S. and Fall, K., "Promoting the Use of End-to-End Congestion Control in the Internet", IEEE/ACM Transactions on Networking, Vol. 7, No. 4, pp. 458-472, August 1999

[Floyd and Padhye 2001] Floyd, S. and Padhye, J., "On inferring TCP behaviour", Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 27-31, 2001, San Diego, CA, USA. ACM, 2001

[Fraleigh *et al* 2003] Fraleigh, C., Moon, S., Lyles, B., Cotton, C., Khan, M., Moll, D., Rockell, R., Seely, T., Diot, C., IEEE Network, 2003.)

[Funahashi 1989] Funahashi, K., "On the approximate realisation of continuous mappings by neural networks, Neural Networks, vol. 2, pp. 183-192, 1989

[Garibaldi 1998] Garibaldi, J., 'Data Mining Algorithms: Applicability to Data Set Types', Proceedings of International Network Conference, Plymouth, 1998

[Garibaldi *et al* 1998] Garibaldi, J.M., Burn-Thomton, K.E., and Mahdi, A.E., "Pro-Active Network Management using Data Mining", in Proceedings of IEEE GLOBECOM 98, pp. 1208-1211, Sydney, Australia, 1998.

[Gaynor 1996] Gaynor, M., "Proactive Packet Dropping Methods for TCP Gateways, October 1996", http://www.eecs.harvard.edu/~gaynor/final.ps

[Haykin 1999] Haykin, S., "Neural networks: a comprehensive foundation" 2nd edition, Prentice Hall, 1999

[Hammer 2003] ***, Hammer Voice Quality Test Suite technical datasheet, http://www.empirix.com/NR/Empirix/NCResources/datasheet_ngnt_vqtestsuite_0302.pdf

[Heidemann *et al* 1997] Heidemann, J., Obraczka, K., Touch, J., "Modeling the Performance of HTTP Over Several Transport Protocols", ACM/IEEE Transactions on Networking, 5 5, 616-630, October, 1997

[Hertz and Krogh 1991] Hertz, J., Krogh, A., Palmer, R.G., "Introduction to the Theory of Neural Computation", Addison Wesley, 1991

[Hiramatsu 1990] Hiramatsu, A., "ATM Communications Network Control by Neural Networks", IEEE Transactions on Neural Networks, vol. 1, no. 1, march 1990

[Hyun *et al* 2003] Hyun, Y, Broido, A, Claffy, C."On Third-party Addresses in Traceroute Paths", Passive and Active Measurement Workshop, April 6-8, 2003, La Jolla, California

[InterMapper 2003] *, InterMapper: Network Monitoring and Alerting Software – website, http://www.intermapper.com

[IBM 2000] *, "HTTP Server: Persistent Connections", IBM iSeries HTTP server documentation, http://www-1.ibm.com/servers/eserver/iseries/software/http/services/persist.htm

[ISC 2003] *, "Internet Domain Survey", Internet Software Consortium website, http://www.isc.org

[ITU 2002] *, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", ITU-T Rec. X.680 (2002) | ISO/IEC 8824-1:2002

[ITU 1999] *, 'Recommendation H.323 - Packet-based multimedia communications systems', H.323 ITU Recommendation', ITU, September 1999

[Jain and Dovrolis 2003] Jain, M., Dovrolis, C., "End-to-End Available Bandwidth: Measurement methodology, Dynamics, and Relation with TCP Throughput", IEEE/ACM Transactions in Networking, August 2003

[Jacobson *et al* 1992] Jacobson, V., Braden, R., Borman, D., 'TCP Extensions for High Performance', Request for Comments 1323, May 1992

[Jacobson and Karels 1988] Jacobson, V., Karels, M., 'Congestion Avoidance and Control', Proceedings of ACM SIGCOMM '88, ACM, Stanford, CA, Aug 1988

[Jacobson 1990] Jacobson, V., 'Berkeley TCP Evolution from 4.3 Tahoe to 4.3 Reno', Proceedings of the 18th Internet Engineering Task Force, Vancouver, September 1990

[Jacobson 2003a] Jacobson, V., *pathchar* source code , ftp://ftp.ee.lbl.gov/pathchar

[Jacobson 2003b] Jacobson, V., tcpdump source code, ftp://ftp.ee.lbl.gov/tcpdump.tar.Z

[Jacobson 2003c] Jacobson, V., *traceroute* source code, ftp://ftp.ee.lbl.gov/traceroute.tar.gz

[Karn and Partridge 1991] Karn, P., Partridge, C., "Improving Round-Trip-Time Estimates in Reliable Transport Protocols", ACM Transactions on Computer Systems, Vol. 9, No. 4, November 1991, pp 364-373

[Kennington 2003] Kennington A., "Simulation software links", http://www.topology.org/soft/sim.html

[Keshav 1991] Keshav, S., "A Control-Theoretic Approach to Flow Control", Proceedings of ACM SIGCOMM, September 1991.

[Kleinrock 1976] L. Kleinrock, "Queueing Systems, Volume II: Computer Applications," John Wiley & Sons, 1976.

[Kumar and Spafford 1994] S. Kumar and E. Spafford, "An Application of Pattern Matching in Intrusion Detection," Technical Report 94-013, Department of Computer Sciences, Purdue University, March 1994.

[Lai and Baker 2000] Lai, K., Baker, M., "Measuring Link Bandwidths Using a Deterministic

Model of Packet Delay", Proceedings of ACM SIGCOMM '00, Stockholm, Sweden, August 2000

[Leland *et al* 1994] Leland, W., Taqqu, M., Willinger, W., and Wilson, D., "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, 2(1), pp. 1-15, February 1994.

[LBNL 2003] ***, 'libpcap 0.7.1', Lawrence Berkeley National Laboratory - Network Research Group, ftp://ftp.ee.lbl.gov/libpcap.tar.Z

[McCreary and Claffy 2000] McCreary, S, Claffy, K., ``Trends in wide area IP traffic patterns - A view from Ames Internet Exchange,", in ITC Specialist Seminar, Monterey, CA, 18-20 Sep 2000

[McDaniel 2001] McDaniel, B., 'TCP Timestamping - Obtaining System Uptime Remotely', e-mail message, http://securityfocus.com/archive/1/168637, March 11, 2001

[Morgan and Bourlard 1990] Morgan, N., Bourlard, H., "Continuous speech recognition using multilayer perceptrons with hidden Markov models", IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 1, pp. 413-416, Albuquerque, 1990

[MCI 2003] *, MCI (Microwave Communications, Inc.), "About MCI: Our Network: IP Latency Statistics", http://global.mci.com/about/network/latency/

[Mills 1992] Mills, D., "Network Time Protocol (Version 3): Specification,Implementation and Analysis," RFC 1305,Network Information Center, SRI International, Menlo Park, CA, Mar.

1992.

[MFN 2003] *, "MFN backbone architecture", http://www.mfn.com/network/ip_networkmaps.shtm, May 2003

[Microsoft 2000] *, "Microsoft Windows 2000 TCP/IP Implementation Details", white paper, http://www.microsoft.com/windows2000/techinfo/howitworks/communications/networkbasics/tcpip_implement.asp

[Minshal 1997] Minshal, G., *tcpdpriv*, ftp://ita.ee.lbl.gov/software/tcpdpriv-1.1.10.tar.Z

[Mogul 1992] Mogul, J., "Observing TCP Dynamics in Real Networks," *Proceedings of SIGCOMM '92*, pp. 305-317, August 1992

[Muuss 2003] Muuss, M., *ping* source code, ftp://ftp.arl.army.mil/pub/ping.shar

[Mukherjee 1994] Mukherjee, A., On the Dynamics and Significance of Low Frequency Components of Internet Load, Internetworking: Research and Experience, Vol. 5, pp. 163-205, December 1994

[Nagendra 1998] Nagendra, S., "Practical Aspects of Using Neural Networks: Necessary Preliminary Specifications", GE internal report 97CRD173, January 1998

[NIMI 2003] *, National Internet Measurement Infrastructure homepage, http://www.ncne.nlanr.net/nimi/

[*NIST Net* 2003] *, NIST Net mailing list archive, http://snad.ncsl.nist.gov/itg/NIST Net/NIST Net.archive.gz

[NNRI 2003] *, The National Regulatory Research Institute (NNRI), "Residential Perceptions of Internet Service Quality: Results of a Survey", January 2003.

[NS 2003] *, "The Network Simulator – ns2 homepage", http://www.isi.edu/nsnam/ns/

[NYI 2003] *, The New York Internet company (NYI), "Super ping – NYI network status page", http://whatsdown.net/superping.shtml

[Osterman 2003] Ostermann, S., tcptrace homepage, http://www.tcptrace.org

[Ott *et al* 1996] Ott, T., Kemperman, J. H. B., Mathis, M., "The Stationary Behavior of Ideal TCP Congestion Avoidance". ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps , August 1996.

[Padhye *et al* 1998] Padhye, J., Firoiu, V., Towsley, D., Kurose, J. – "Modelling TCP Throughput: A Simple Model and its Empirical Validation", Proceedings of SIGCOMM '98, Vancouver, CA, 1998

[Pappalardo 2002] Pappalardo, D, "Cable & Wireless ups latency SLAs in North America, Europe", Network World, June 2002, http://www.nwfusion.com/news/2002/132261_05-06-2002.html

[Paxson 1997a] Paxson, V., "Measurements and Analysis of End-to-End Internet Dynamics", PhD thesis, Computer Science Division, University of California, Berkeley, April 1997

[Paxson 1997b] V. Paxson, “Automated Packet Trace Analysis of TCP Implementations”, Proceedings of ACM SIGCOMM '97, September 1997, Cannes, France

[Paxson 1997c] Paxson, V., “End-to-End Routing Behavior in the Internet”, IEEE/ACM Transactions on Networking, 5(5), pp. 601--615, Oct. 1997.

[Paxson 1997d] Paxson, V., “Why We Don't Know How to Simulate the Internet”, Proceedings of the 1997 Winter Simulation Conference, Atlanta GA, U.S.A., Dec. 1997

[Paxson 1998] Paxson, V., “On calibrating measurements of packet transit times”, Proceedings of ACM SIGMETRICS '98, pp. 11-21, Madison, WI, June 1998

[Paxson *et al* 1998] Paxson, V., Mahdavi, J., Adams, A., Mathis, M. “An Architecture for Large-Scale Internet Measurement”, IEEE Communications, v.36, no.8, pp 48-54, August 1998,

[Paxson 1999] Paxson, V., “End-to-end internet packet dynamics”, IEEE/ACM Transactions on Networking, Volume 7 , Issue 3 , June 1999, pp 277-292

[Paxson *et al* 1999] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., Volz, B., Request for Comments 2525, "Known TCP Implementation Problems", March 1999.

[Paxson *et al* 2000] Paxson, V., Adams, A., Mathis, M., “Experiences with NIMI”, Proceedings of Passive and Active Measurement, 2000

[PIPEX 2003] *, PIPEX, "PIPEX – Latency Statistics", http://www.connection.pipex.net/support/network/latency.shtml

[Popescu and Shankar, 1999] Popescu, C. T., Shankar, A. U., "Empirical TCP Profiles and Application", 7th Intl. Conf. on Network Protocols (ICNP'99), Toronto, 1999

[Postel 1980] Postel, J., "User Datagram Protocol", Request For Comments 768, August 1980

[PMA 2003] *, Passive Measurement and Analysis homepage, http://pma.nlanr.net/PMA/

[Prechelt 1998] Prechelt, L., "Automatic early stopping using cross validation: quantifying the criteria", Neural Networks, 11, 1998, pp. 761-767

[Quallaby 2003] *, Quallaby corporation website, http://www.quallaby.com

[Ramaswamy and Gburzynski 1999] Ramaswamy, S., Gburzynski, P., "A Neural Network Approach to Effective Bandwidth Characterization in {ATM} Networks", in Performance Analysis of ATM Networks, IFIP vol. 4, Demetres Kouvatsos, editor, Kluwer Academic Publishers, 1999.

[Savage 1999] Savage, S., "Sting: a TCP-based Network Measurement Tool", Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems, pp. 71-79, Boulder, CO, October 1999

[Schemers 2003] Schemers, R. J., fping home page, http://www.stanford.edu/~schemers/docs/fping/

[Securiteam 2001] *, "TCP Timestamping - Obtaining System Uptime Remotely", http://www.securiteam.com/securitynews/5NP0C153PI.html, March 2001

[Shenker *et al* 1990] Shenker, S., Zhang, S., Clark, D., "Some Observations on the Dynamics of a Congestion Control Algorithm" ACM Computer Communication Review, 20(4):30–39, October, 1990

[Shunra 2003] *, Shunra's Network Simulation and Emulation Solutions, http://www.shunra.com

[Silberschatz and Tuzhilin 1995] Silberschatz, A., and Tuzhilin, A. 1995. "On Subjective Measures of Interestingness in Knowledge Discovery", Proceedings of KDD-95: First International Conference on Knowledge Discovery and Data Mining, 275–281. Menlo Park, Calif.: American Association for Artificial Intelligence.

[Schulzrinne *et al* 1996] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications", Request For Comments 1889, January 1996

[SNNS 2003], *, "Stuttgart Neural Network Simulator", http://www-ra.informatik.uni-tuebingen.de/SNNS/, 2003

[Stevens 1995] Stevens, R., Wright, G., "TCP/IP Illustrated, Volume 2: The Implementation". Addison Wesley Professional. 1995

[Stevens 1997] Stevens, R., Request For Comments 2001 "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", January 1997

[Sun and Ifeachor 2000] Sun, L., Ifeachor, E., "Perceived Speech Quality Prediction for Voice over IP-based Networks", Proceedings of IEEE International Conference on Communications (IEEE ICC'02), New York, April 2002, pp.2573-2577.

[Swan and Bacher 1997] Swan, A., and Bacher, D., rtpmon 1.0a7, ftp://mmftp.cs.berkeley.edu/pub/rtpmon/, University of California at Berkeley, January 1997

[Swingler 1996] Swingler, K., "Applying Neural Networks – A Practical Guide", Academic Press, 1996

[UUNET 2003b] *, "UUNET Latency statistics", http://www1.worldcom.com/uk/about/network/latency

[UKERNA 2003] *, UKERNA (United Kingdom Education & Research Networking Association) and JANET (Joint Academic Network) Home page, http://www.ja.net

[TDSLink 2003] *, TDSLink website, http://www.tdslink.com

[Thompson and Miller 1997] Thompson, K., Miller, G.J. 'Wide-Area Internet Traffic Patterns and Characteristics', IEEE network, November 1997

[TIRC 2003] *, The Insight Research Corporation, "IP Telephony: Service Revenue and OSS Expenditures for Voice over Packet Networks 2002-2007 - a market research report", October 2002

[Wakeman *e. al* 1992] Wakeman, I., Lewis, D., and Crowcroft, J., "Traffic Analysis of Trans-Atlantic Traffic," Proceedings of INET '92, Kyoto, Japan, 1992

[Wassenaar 2003] Wassenaar, E., *Nikhef ping* source code, ftp://ftp.nikhef.nl/pub/network/ping.tar.Z

[Weigend *et al* 1990] *Weigend*, A.S., Huberman, B., Rumelhart, D., "Predicting the future: A connectionist approach", International Journal of Neural Systems, vol. 3, pp. 367-375, 1990

[Wendland 2000] Wendland, R., "Re: a question about the deployment of SACK and NewReno TCP", e-mail message to *end2end-interest* mailing list, March 2000

[Wendland 2003] Wendland, R., "How prevalent is Timestamp option and PAWS?", e-mail message to *end2end-interest* mailing list, May 2003

[Willinger *et al* 1995] Willinger, W., Taqqu, M., Sherman, R., and Wilson, D., "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," Proceedings of SIGCOMM '95, pp. 100-113, Cambridge, MA, September 1995.

[Woods 2000] Woods, D., "Fishy Business", Network Computing, http://www.networkcomputing.com/1114/1114f1.html?ls=NCJS_1114bt , July 2000

[wget 2003] "GNU wget homepage", http://wget.sunsite.dk

[Yajnik *et al* 1999] Yajnik, M., Moon, S., Kurose, J., Towsley, D., "Measurement and Modeling of the Temporal Dependence in Packet Loss," Proceedings of IEEE INFOCOM '99, March 1999

[Yahoo 2003] *, "Random Yahoo Link page", http://random.yahoo.com/bin/ryl

[Zell *et al* 1994] Zell, A., Mamier, G., Vogt, M., Mach, N., Huebner, R., Herrmann, K.U., Soyez, T., Schmalzl, M., Sommer, T., Hatzigeogiou, A., Doering, S., Posselt, D.: SNNS Stuttgart Neural Network Simulator, User Manual. University of Stuttgart (1994)

[Zhang *et al* 1991] Zhang, L., Shenker, S., Clark, D., "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", Proceedings of ACM SIGCOMM '91, Zurich, Switzerland, September 1991

[Zhang *et al* 2000] Zhang, Y., Paxson, V., Shenker, S, "The Stationarity of Internet Path Properties: Routing, Loss, and Throughput", ACIRI Technical Report, May 2000

[Zhang *et al* 2001] Zhang, Y., Duffield, N., Paxson, V., Shenker, S, "On the Constancy of Internet Path Properties", Proceedings of ACM SIGCOMM Internet Measurement Workshop, November 2001.

Appendices

Appendix A – Real-time traffic monitor

The aim of this project with regards to real-time traffic was to gather the parameters that would provide a measurement architecture with the necessary network image in order to asses the performance of various real-time applications. Constructing such an image would require information about the following parameters:

- Delay - the time elapsed between the sending of a packet and its arrival at the destination;
- Jitter - the variance of the delay value;
- Packet loss - the number of lost packets, reported in the time elapsed, and the burstiness of the loss events.

Transport QoS has two main areas: end-to-end measurements and, in case there are changes in the level of parameters, fault localisation. Based on the diagram from Figure 3.1, a traditional approach of fault localisation for UDP traffic would require a 3-device configuration: two placed right at the endpoints and one along the path of the flows. This is due to the fact that UDP does not provide any reception feedback from the receiving (remote) endpoint. As a result, the endpoint devices would inform about end-to-end parameters and fault location would result from comparative measurements made by the third device. In the simplest case, the third device (the one placed along the path of the flows) would serve also as data gathering point, collecting the information provided by the endpoint-placed devices. This 3-device measurement configuration has two clear disadvantages: it would be intrusive (in the best case scenario, the endpoint clients need to send the data to the server even if they are doing non-intrusive monitoring) and it would require privileged positioning, i.e. at the endpoints, for some of the monitoring devices.

The QoS for transport can be determined from the audio flows within a call (which run on RTP). Current tools, such as ***Hammer VTQS*** (Voice Quality Test Suite) [Hammer 2003], ***Agilent (IP Telephony) Analyser*** [Agilent 2003], and ***rtpmon*** [Swan and Bacher 1997], base their calculations upon parsing both the RTP and/or RTCP and displaying the available data. Because they do not combine the two types of analysis, none of them can establish fault location without using the traditional approach mentioned above.

Each of the two flows, i.e. RTCP and RTP, has its own advantages and disadvantages when analysing the transport. RTCP does provide enough information to determine the end-to-end performance, but it has two drawbacks: it informs *only* about end-to-end parameters, therefore it cannot be used for localising faults and it has a poor granularity, because of its scalability requirements, for conferences with multiple participants. RTP has a very good granularity, but it is limited to the network sub-path sender-monitor (i.e. *Endpoint A* → *Monitoring point* and *Endpoint B* → *Monitoring point* in Figure 3.1), without being able to infer anything for the rest of the path (i.e. *Monitoring point* → *Endpoint A* and *Monitoring point* → *Endpoint B* in the same figure).

This section proposes a method to obtain a better view of the network performance, without using several devices and without injecting additional traffic into the network. To achieve its aim, the method monitors the RTP and RTCP flows and correlates the information they provide to reveal both the end-to-end performance and the fault localisation (when the monitored parameters change their value along the route

A.1 Monitoring procedure

The monitoring procedure comprises three steps: capture (and parsing), parameter inference / extraction, and correlation. First, the real-time flows (RTP) are identified and captured. The capturing module was described in section 3.5; identification of the RTP flows will be described in the next sub-section. In the inference phase, the parsed information from the RTP header fields and the RTCP reports is analysed / interpreted to obtain the performance parameters. The last phase, correlation of RTP and RTCP, is used to establish the location of the network degradation, if any.

A.1.1 Identification of the real-time flows

The monitor must first determine the ports on which the RTP flow(s) will be running within a VoIP call. The actual procedure depends heavily on the application used. For example, in the case of a VoIP call, the monitor must decode the call signalling, because the ports are dynamically allocated by the two endpoints during the call initiation. Even after narrowing the applications to VoIP, it depends on the stack used (e.g. Session Initiation Protocol (SIP, [Handley *et al* 1999]), or H.323 [ITU 1999]). In the case of SIP, the addressing and syntax are the same with HTTP; as a result, port identification can be performed by parsing the text tags from the call initiation messages. The case of H.323 is more complex, as the entire signalling dialogue between the corresponding endpoints is encoded using ASN.1 syntax [ITU 2002]

The task of identifying the flows via any of the above techniques was considered to be outside the scope of this project, due to the implementation time it required and the lack of novelty. During the validation process, section 6.1, the RTP port identification was done using the characteristics of the H.323 implementation involved. It was observed that Netmeeting, the client used to generate the H.323 test traffic, allocated ports for RTP flows using the even numbers starting with 48000. Also, the RTCP port for a specific real-time flow *n* is allocated next to the RTP port, $port_{RTCPn} = port_{RTPn} + 1$. Using this information, the parsing module was set to interpret as RTP / RTCP all the UDP traffic on even / odd ports above 48000, and to pair them according to the above rule.

A.2 Parameter inference and extraction

There are two types of analysis during this phase: one using the RTP packet headers, based on inference of the network events that lead to this succession of packets, and one using the RTCP reports, which involves only parsing and interpretation, as the parameters are already provided by these packets. Aside from the RTP header / RTCP reports, the monitor also uses the arrival timestamp of each packet to perform the timing calculations.

The monitor retrieves the associated flow object from the **Connection database** in order to perform the analysis. The RTP/RTCP flow objects in the **Connection database** contain the following information:

- Flow status: to indicate whether the flow is active or not. The variable was introduced in the proof-of-concept monitor developed, but no mechanisms were developed to modify it, due to the unavailability of an ASN.1 decoding facility.
- Flow information (FlowEnds): to identify the RTP/RTCP flow. Although, as mentioned in section 3.2, there is an unique correspondence between a flow and its corresponding

object in the database, this correspondence was slightly altered in the case of RTP/RTCP to simplify the analysis. Each RTP/RTCP pair was associated to a single object, as the two flows work together and the information provided by them needs to be correlated in the last part of the analysis.

- Packet accounting variables
  - Number of RTP packets reported lost by the receiving end, from the RTCP receiver reports
  - Number of lost RTP packets at the monitor, RTP sequence number of the last RTP packet
- Timing variables
  - Timestamp of the first / last captured RTP packet within the flow
  - Timestamp of the last RTP packet identified as lost.
  - RTP timestamp of the last captured RTP packet within the flow
- Delay variables
  - RTP timestamp resolution, based on the payload type field within the RTP header, used to convert the RTP timestamp in time units
  - Inter-arrival delay / jitter at the endpoint, from pair RTCP receiver reports
  - Last / average inter-arrival delay at the monitor
  - Inter-arrival / one-way jitter at the monitor
  - Timestamp of the last SR sent
- One-way jitter (*ow_jitter*), inter-arrival jitter (*ia_jitter*)
- Misordering information
- Skip sequences (*SkipSeqno*) - array used to determine if a packet is a duplicate or an out-of-order one. With the simplifying assumptions introduced, the array was subsequently removed from the analysis.

In addition to the flow-associated object, the following RTP fields were involved in the analysis:

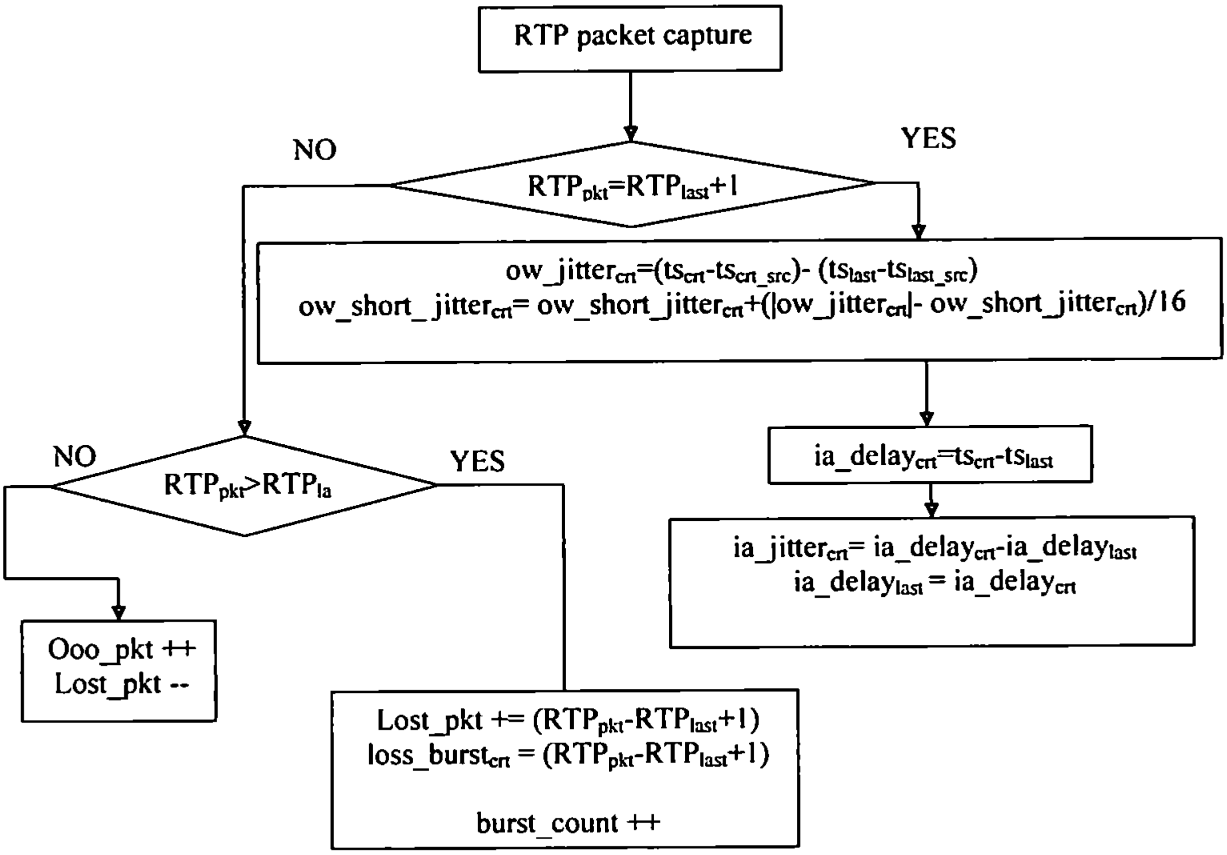
- RTP sequence number (RTP_{pkt}) – to identify lost / out-of-order segments
- RTP timestamp (ts_{pkt}) – to determine user timing information
- Payload type – to determine whether the sender changed the codec used since the last captured RTP packet.

The algorithm starts from the assumption that there are only five possible alternatives for each captured packet: in-sequence, 'future', out-of-order, lost, or duplicated segment. The monitor determines to which of these four categories the current packet belongs by comparing its RTP sequence number (RTP_{pkt}) with the expected RTP sequence ($RTP_{expect}=RTP_{last}+1$) and with the content of the SkipSeqno. The packet is:

- *in-sequence*, if RTP_{pkt} is equal with the expected sequence number
- *future*, indicating either one / several connected losses or a miss-ordering event, if RTP_{pkt} is higher than RTP_{expect}. To simplify the analysis, it is assumed that the packets carrying the missing sequence number were lost but they are marked as skipped in SkipSeqno
- *out-of-order*, if RTP_{pkt} is lower than RTP_{expect} and the sequence number is listed in SkipSeqno.
- *duplicate*, if RTP_{pkt} is lower than RTP_{expect} and the sequence number is not listed in SkipSeqno.
- *lost*, if RTP_{pkt} is lower than RTP_{expect} and the sequence number is not listed in the SkipSeqno

The proposed states are valid from a theoretical perspective, but they are difficult to implement in practice due to the technical limitations. Unlike TCP, which retransmits all the lost segments, UDP, which provides the transport for RTP/RTCP, does not perform any recovery if segments are lost. As a result, SkipSeqno, that carries all the sequence number streams which were lost *or* misordered, will fill up if packet loss exists, no matter how large the array is (in fact, making it large only for this purpose would increase the monitor's memory requirements uselessly). One of Paxson's Internet studies [Paxson 1999] has shown that packet duplication is a very rare network phenomenon and indicates severe routing problems, network conditions under which the proposed method does not function properly anyway, because it might not capture all the packets. If it is assumed that packet duplication does not happen, there are only two types of network events to identify: packet misordering, where two / several successive packets arrive at the monitor in a different order than the one they were sent, and packet loss, where one / several packets are lost between the sender and monitor and will never arrive at the monitor. As a result, when the monitor captures a packet with a sequence number lower than expected, the packet is automatically interpreted as misordered and the number of lost packets is decremented. With this simplification, the SkipSeqno array is not required at all; the monitor can distinguish between the network events without historical knowledge of the transfer.

Once the associated flow object is retrieved from the database, the monitor, depending on the packet type, runs an analysis of the RTP header in conjunction with the flow object or parses the RTCP header using the above assumptions, then updates the flow variables. In the following, the case of RTP analysis will be discussed, as RTCP parsing is of less interest. A schematic diagram of the RTP algorithm is given below in Figure A.0.1.


RTP packet capture
NO
YES
RTP$_{pkt}$=RTP$_{last}$+1
ow_jitter$_{crt}$=(ts$_{crt}$-ts$_{crt\_src}$)- (ts$_{last}$-ts$_{last\_src}$)
ow_short_jitter$_{crt}$= ow_short_jitter$_{crt}$+(|ow_jitter$_{crt}$|- ow_short_jitter$_{crt}$)/16
NO
RTP$_{pkt}$>RTP$_{la}$
YES
ia_delay$_{crt}$=ts$_{crt}$-ts$_{last}$
ia_jitter$_{crt}$= ia_delay$_{crt}$-ia_delay$_{last}$
ia_delay$_{last}$ = ia_delay$_{crt}$
Ooo_pkt ++
Lost_pkt --
Lost_pkt += (RTP$_{pkt}$-RTP$_{last}$+1)
loss_burst$_{crt}$ = (RTP$_{pkt}$-RTP$_{last}$+1)
burst_count ++

Legend:
ts$_{crt/crt\_src/last/last\_src}$ – timestamp values, recorded for current packet at monitor / current packet at sender / last packet at monitor / last packet at sender.
ow_jitter$_{crt}$ – current measurement of one-way jitter
ow_short_jitter$_{crt}$ – current value of short term one-way jitter [Schulzrinne *et al* 1996]
ow_jitter_pkts – number of one-way jitter samples
RTP$_{pkt/last}$ – RTP timestamp of current / last packet
ia_delay – inter-arrival delay, as recorded at the monitor
ia_delay$_{crt}$ – current inter-arrival delay, as recorded at the monitor
ia_jitter$_{crt}$ – current inter-arrival jitter, as recorded at the monitor
ia_delay_pkts – number of samples for inter-arrival delay jitter calculation
lost_pkt – number of lost packets
loss_burst$_{crt}$ – number of packets lost in the current loss burst
burst_count – counter of the number of loss bursts
ooo_pkt – number of misordered packets

Figure A.0.1 - Network parameter inference based on RTP analysis

There are two types of parameters can be determined using the RTP header fields and the arrival timestamp of each packet: delay-related (inter-arrival delay, inter-arrival jitter, and one-way jitter) and accounting-related (lost packets and out-of-order packets). The inter-arrival delay is determined by subtracting the capture timestamps of successive in-order packets. The average value will be equal with the inter-send delay, unless there is skew or drift, as defined in [Mills 1992], between the clocks at the sender and receiver.

The *one-way delay jitter* would ideally relate to the absolute position of the packet in the stream; unfortunately, the monitor does not have any reference for establishing the absolute packet position. Using a single reference, such as the first captured packet, would slide the figure for the entire stream with the jitter of that packet. To solve this problem, the previous packet is used as reference; in the diagram in Figure A.0.1, ts_{last_src} and ts_{crt_src} are the timestamps of the current and last packet, calculated using the RTP sequence number, and ts_{last} and ts_{crt} are the capturing timestamps of the two packets. The algorithm is similar to the one used by RTCP to calculate and report jitter and it requires knowledge of the codec used, i.e. the inter-send delay between successive RTP packets. In addition, the monitor also calculates an *inter-arrival delay jitter*. The value is calculated by comparing the previous delay with the actual one; the resulting value is not affected by clock drift, but it is subject to the time delays between successive packet arrivals; it also cannot provide a jitter estimate in the case of packet misordering / loss. There are two one-way jitter variables: one carrying a short term value (ow_short_jitter), indicating the short-term jitter, and ow_jitter, which is a sum of modules of the jitter values. The ow_short_jitter is calculated using the formula recommended in [Schulzrinne *et al* 1996], in order to allow comparison with the short-term end-to-end jitter provided by RTCP.

The lost and out of order packets are both determined using the RTP sequence number: when a packet is captured, its sequence number is compared with the expected sequence number, falling in one of the categories previously mentioned. In addition, the monitor calculates the burstiness of losses, using the *burst_loss* variable, which measures the size of the current burst. At the end of the connection, the average size of a burst may be determined by dividing the number of lost packets with the burst_loss. The mechanism includes a short-term protection to account for misordering events, where one / several RTP presumed-lost packets arrive later and must be discarded as losses. If the current burst of presumed-loss packets arrives later, a $loss\_burst_{crt}$ variable, initially carrying the size of the burst, is decreased with each packet; when it reaches zero, the *burst_loss* is also decremented. The mechanism works only as long as there is only one misordered packet / train of packets at one time; for more complex misordering schemes, the algorithm would require an array of $loss\_burst_{crt}$ values instead of a single variable.

From the obtained variables, the one_way jitter, loss, and the inter-loss delay are of interest for a voice quality model. In addition, the average inter-arrival delay may be used to determine an eventual clock skew between the clocks at the sender and at the monitor. Also, out-of-order packets, depending on the policy of the receiver, would either be considered lost or, if the receiver has a long queue and they arrive in time, delayed packets with high jitter.

The RTP specification provides an option for codecs to stop sending RTP packets during the silence periods in order to reduce the amount of traffic produced[1]. Such silence periods have to be detected, otherwise they would be interpreted by the monitor as very high inter-arrival delays. The detection is made using the (silence) marker field from the RTP packet which, when set in a packet, indicates the end of a silence period.

A.2.1 Correlating RTP analysis with RTCP parsing

By correlating the two sets of parameters, obtained from RTP and RTCP, it is possible to determine whether or not a specific problem (e.g. a high packet loss rate) is caused by a problem which exists in the East Subnetwork or the West Subnetwork. Figure A.0.2 presents the captured flows.

[1] An alternative is receiver-generated comfort noise, based on spectral information provided by the sender.

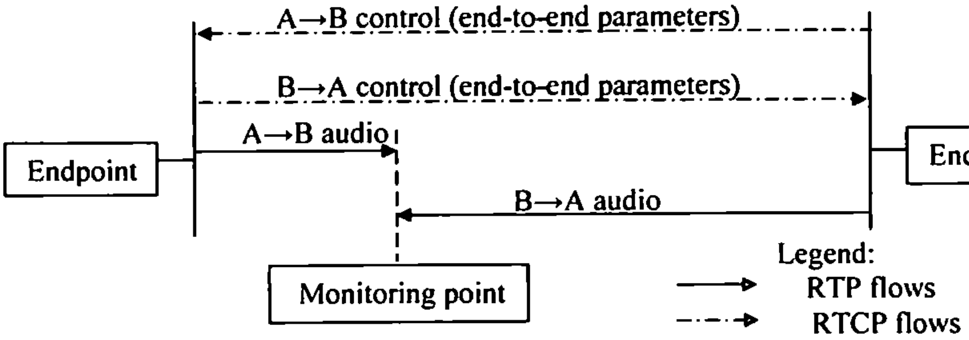


Figure A.0.2: RTP and RTCP flows monitoring

The RTP streams, as captured on the monitoring point, are: A→B (after passing through the West sub-network, see Figure 3.1) and B→A (after passing through the East sub-network). Therefore, by measuring the parameters of these flows, we can determine the performance of the West sub-network (from the A→B flow) and the East sub-network (from the B→A flow). Meanwhile, as mentioned, RTCP provides the end-to-end parameters, i.e. the performance of the entire A→B and B→A routes, but it has no indication about how these parameters change on the route (i.e. cannot establish where a faulty behaviour of the network determined a change in the values of the parameters).

Putting together the two sets, the parameters for the following segments result:

- A→B and B→A, end-to-end – from the RTCP flows
- A→monitoring point and B→monitoring point – from the RTP flows
- monitoring point→B and monitoring point→A – by subtracting the RTP obtained values from RTCP end-to-end parameters.

Therefore, by using both RTP and RTCP, the method provides both the end-to-end and the end-to-monitoring point parameters for a monitored real-time flow.

A.3 VoIP validation

The following set of parameters (symmetric for the two directions) was set on the ***NIST Net*** machine: 5% packet loss, 300 ms delay, 25 ms delay jitter, unlimited bandwidth, normal distribution for loss and delay. The measurements were based on a long capture session, with approximately 20,000 packets captured.

The software tools used in the experiment were: Netmeeting, for generating the flows, running at the two endpoints, ***tcpdump***, for packet capturing, running at the monitoring station. The H.323 call used the G.723.1, 6400 bits/second codec. In order to avoid repeating the test due to errors in the monitor, the entire session was captured using ***tcpdump*** and stored for offline analysis.

The process used the packet trace and the VoIP analysis implementation, and it included two stages. First, the RTP and RTCP flows within the trace were analysed, using the method described in section 3.3, to estimate the inter-arrival jitter and the packet loss. For each direction there were two different sets of results, one from the RTP analysis, indicating the properties of the path segment between the sender and the monitor, and one resulting from the RTCP parsing, which was showing the end-to-end properties of the path. In the second stage, the two sets of parameters were combined to identify the sub-path where the network conditions had degraded.

The trace analysis produced a set of results summarised in Table A.1. The *unimpaired* column indicates the measured values without any impairments (i.e. the network emulator does not introduce any delay, jitter, or/and loss). The 7ms delay was created by the testbed conditions: even without introducing any degradation, NetMeeting sending behaviour produced a 3ms delay jitter, and each of the two Subnets (East and West) created an additional 2ms jitter delay.

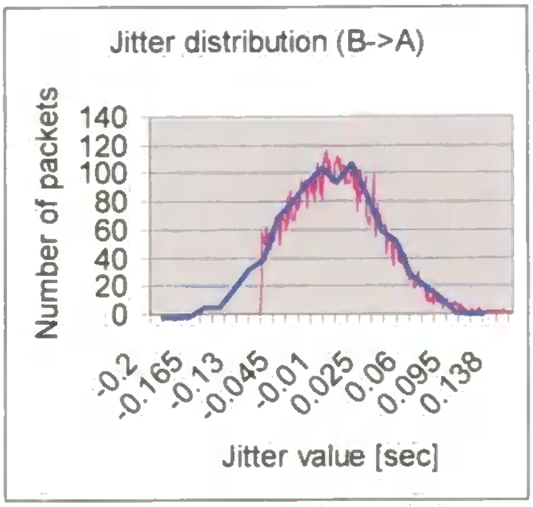
Parameter	unimpared	RTP results		RTCP results	
		A→B	B→A	A→B	B→A
packet loss [%]	0	0	5	5	5
jitter [ms]	7	5.4	28.6	30.9	31.1

Table A.1 - Throughput and packet loss statistics

Based on Table A.1, the loss estimation and localisation is straightforward. For the B→A direction, the end-to-end path value (5%) equals the value for the Endpoint A-to-monitor path; from here, the entire loss on this direction happens on the Endpoint A-to-monitor path. Similarly, the A→B values indicate that there is no alteration on the Endpoint A-to-monitor path, as the loss is 0% on that segment, but 5% on the end-to-end path.

The inter-arrival jitter, at the monitor, was also measured from the RTP flow, and used the receiver reports to calculate the end-to-end value. Comparing again the values obtained, it was observed that the A→B direction has a 5.4ms jitter delay, which would approximate the ideal 2+3 ms behaviour, due to sender and the West Subnetwork. The estimation for the 25ms overall jitter from A to B was slightly less accurate. The total figure of 30.9ms leads to a value of 23.9ms additional jitter (after removing the 7ms generated even during ideal conditions). Similarly, in the reverse direction, the value of the jitter is high for both end-to-end and subnet east measurements, giving a value of only 2.5ms (near the 2ms value observed under unimpaired conditions) for the West Subnetwork.

The jitter involved additional post-analysis, in order to determine whether the measurement was accurate throughout the call. Figure A.0.3 displays the resulting jitter distributions.



Legend:

━━━━ the injected jitter (approximate shape)

──── the measured jitter

Figure A.0.3: RTP B→A jitter distribution (from RTP analysis)

The injected jitter was evaluated by sending ping packets with 1 second inter-send delay, between the two hosts, followed by examination of the inter-arrival values. The resulting graph from Figure A.0.3 was named *approximate* because it was produced by rounding the actual values.

The jitter distribution for the A→B flow, which is captured after passing through the West Subnetwork, did not show any distribution, except a main spike at 2ms, produced by the network behaviour previously mentioned.

In the RTCP parsing, the values were extracted from the RTCP report blocks (the 'inter-arrival jitter' value).

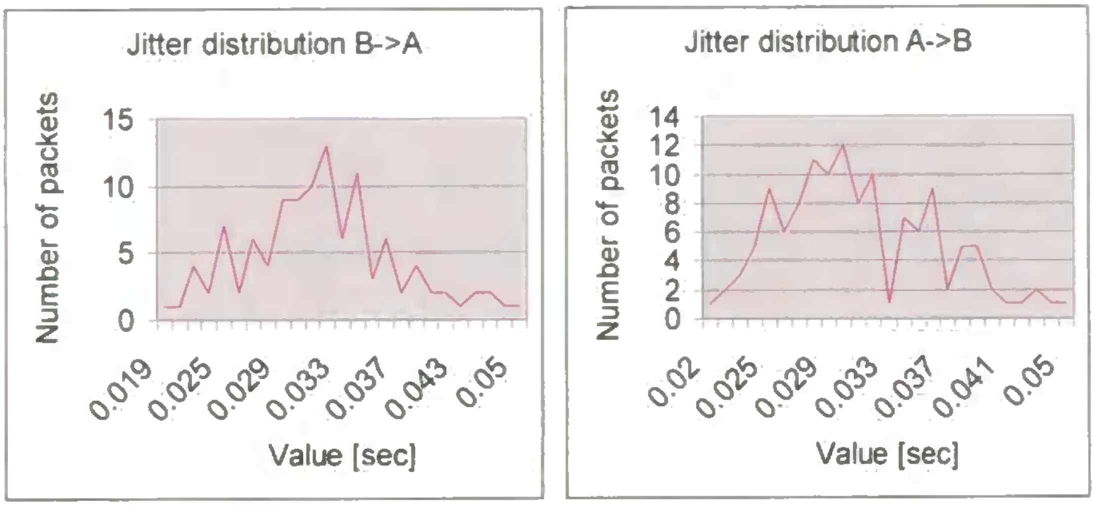


Figure A.0.4: RTP jitter distribution (from RTCP parsing)

As can be seen from Figure A.0.4, the distribution for the B→A flow can be approximated with a Normal (Gaussian) one, while the A→B flow shows no distribution of the jitter.

Considering the absolute values for the jitter, it results an average value of 28.6 ms, which, if subtracting the 3 ms caused by NetMeeting behaviour and the 2 ms due to Subnetwork East, leads to a value of 23.4 ms, which approximates the value of the emulated link, i.e. 25 ms. Concluding, the method identified the 5% loss and 25 ms jitter generated by the right side of the monitored route.

A.3.1 Limitations

As can be seen, the inference proved to be more accurate for packet loss than jitter. One of the minor causes of this may be the errors in the jitter measurement. However, the error is likely to have been caused by ***NIST Net*** behaviour, which, as described in section 5.2, produces packet misordering when it introduces jitter. As typical queuing should not lead to misordering, except for the case of alternate routes, the jitter was not calculated for misordered packets in order to speed up the analysis.

Appendix B - Scripts

Appendix B.1 – Neural network processing scripts

9.4.1.1. B.1.1 Main processing script (nn.sh)

```
#!/bin/sh
#script to prepare, run and display the results when applying a neural net to
a data file
#written by Bogdan Ghita 25/07/01

#usage nn.sh data_file neural_net
cdir="/home/bogdan/nn"

echo $0 $*


#set the initial values

zero=0
batch=0
single=0
bcycles=1

logdata=0
expdata=0
softmax=0
p1min=10
p2min=0
p1max=1000
p2max=1000
#p3min=100
#p3max=1000

arg1_step=50
arg2_step=10
#arg3_step=10

cycle_step=1000
crtcycles=$cycle_step
tt=0

filter=0
min_out_attr=0
max_out_attr=100
min_out_var=0
max_out_attr=100

randomise=0
no_scalling=0
outmax=0

test_prc=10
decimate=0
testnet=0
```

```
self_snns=0
self_err=0
self_corr=0
self_both=0

error_step=0.001
sqerr_prev=0
sqerr_crt=0
corr_prev=0
corr_crt=0

echo "0" >  $cdir/run/test_sqerr.tmp
echo "0" >  $cdir/run/train_sqerr.tmp
echo "0" >  $cdir/run/test_corr.tmp


usage () {
echo "usage:
      nn.sh -f data_file [-logdata | -softmax | -expdata ] -n neural_net
      or
      nn.sh -tt train_file test_file -n neural_net

      Options:
          [-r] - randomise dataset

          [-tp test_percent] - define the percentage of test subset from the
dataset

          [-ca min_outlier_percents max_outlier_percents] - remove outliers
of attributes
          [-cv min_outlier_percents max_outlier_percents] - remove outliers
of variable to estimate
          [-d decimate_percent] - reduce the dataset to decimate_percent
percents

          [-b cycles [-p1 param1_minimum param1_maximum] [-step_p1
param1_step] [-p2 param2_minimum param2_maximum] [-step_p2 param2_step]
            [--batch_outliers max_outlier_to_remove] ]
            [-c_step cycle_step]

          [-s param1 param2 cycles]


          [-cc] - apply cross validation ( split training set into 80%
estimation + 20% validation )
          [-self_err [-e error_step] ] - apply self control using the
variation in average square error, with a margin of error_step
          [-self_corr [-e error_step] ] - apply self control using the
variation in correlation, with a margin of error_step
          [-self_combined [-e error_step] ] - apply self control using the
variation of both correlation and average square error, with a margin of
error_step

Additional options:
      --no_scalling - do not perform scalling of the values to the min_value
- max_value interval, but to 0.0 - max_value interval

      "
}


while true
do
      case "$1" in
```

```
-h) usage
    exit;;
-logdata) logdata=1
    shift;;
-expdata) expdata=1
    shift;;
-softmax) softmax=1
    shift;;
-tt) tt=1
    traindata="$2"
    testdata="$3"
    data="$4"
    shift 4;;
  -f) data="$2"
    shift 2;;
-cc) cross=1
    shift;;
-ca) filter=1
    min_out_attr="$2"
    max_out_attr="$3"
    shift 3;;
-cv) filter=1
    min_out_var="$2"
    max_out_var="$3"
    shift 3;;
  -n) net="$2"
    shift 2;;
  -b) batch=1
    bcycles=$2
    shift 2;;
-c_step) cycle_step=$2
    crt_cycles=$2
    shift 2;;
-p1)p1min="$2"
    p1max="$3"
    shift 3;;
-step_p1) arg1_step="$2"
    shift 2;;
-step_p2) arg2_step="$2"
    shift 2;;
-p2)p2min="$2"
    p2max="$3"
    shift 3;;
  -q) quiet="true"
    devtty="/dev/null"
    shift;;
  -r) randomise=1
    shift;;
-self_err) self_err=1
    shift;;
-self_corr) self_corr=1
    shift;;
-self_snns) self_snns=1
    shift;;
-e) error_step="$2"
    shift 2;;
-tp) test_prc="$2"
    shift 2;;
-s) single=1
    param1=$2
    param2=$3
    bcycles=$4
    shift 4;;
-test) testnet=1
```

```
          shift;;
      -d) decimate=1
          dec_percent=$2
          shift 2;;
        -v)     verbose="true"; shift;;
      --no_scalling) no_scalling=1
          shift;;
        -*)     echo "run.sh: '$1' unexpected"; usage ; exit ;;
        *)      break;;


      esac
done

if [ $tt -eq 0 ]
then
    echo "dataset = $data.data"
else
    echo "train dataset = $traindata.data / test dataset = $testdata.data"
fi
echo "net = $net.net"

rm -f $cdir/run/*

if ([ -z $data ] || [ -z $net ]) && ([ -z $traindata ] || [ -z $testdata ])
then
usage;
exit
fi

if (!([ -e $cdir/data/$data.data ]) && ([ $tt -eq 0 ]))
then
echo "dataset $data not found"
exit
elif ([ $tt -eq 1 ])
then
    if !([ -e $cdir/data/tt/$traindata.data ])
    then
      ls $cdir/data/tt/$traindata.data
      echo "train data - $cdir/data/tt/$traindata.data not found"
      exit
    fi

    if !([ -e $cdir/data/tt/$testdata.data ])
    then
      echo "test data - $cdir/data/tt/$testdata.data not found"
      exit
    fi
fi



if  !([ -e $cdir/nets/$net.net ])
then
echo "network $net not found"
exit
fi


rm -f *_norm.pat
rm -f *_clean.pat
rm -f $cdir/$data.snns.out
touch $cdir/$data.snns.out
```

```
#work in the 'run' directory
    cd $cdir/run

        cp $cdir/data/$data.data $cdir/run/
        cp $cdir/data/$data.data $cdir/run/$data.orig

    if [ $tt -eq 0 ]
    then
#        cp $cdir/data/$data.data $cdir/run/
#        cp $cdir/data/$data.data $cdir/run/$data.orig
        cd $cdir/run

    else
        cp $cdir/data/tt/$traindata.data $cdir/run/
        cp $cdir/data/tt/$testdata.data $cdir/run/
    fi
    cp $cdir/nets/$net.net $cdir/run/

#rm -f ./*

nlines=`nl -n ln  $cdir/run/$data.data | tail -1 | tr -s ' '| tr -s '\t' |
cut -f1 -d' '`
if [ $tt -eq 0 ]
then
# decimate the file
    if [ $decimate -eq 1 ]
    then
        echo "keep only $dec_percent% of the file -
$(($nlines*$dec_percent/100)) out of $nlines"
      cat $cdir/shell/filter.awk | sed s/nlines/$nlines/g | sed
s/minlimit/0/g | sed s/maxlimit/$dec_percent/g > filter.awk.tmp
        cat $cdir/run/$data.data | awk -f filter.awk.tmp > $data.tmp
      cp filter.awk.tmp tmp.awk
        cp $data.tmp $cdir/run/$data.data
      mv $data.tmp $data.decimate
    fi

#filter the data first, if required
    if [ $filter -eq 1 ]
    then
      #switch temporary to shell directory
      cd $cdir/shell
      cp $cdir/run/$data.data $cdir/shell/
      #do the filtering
        #./filter.sh $data.data $min_out $max_out
        ./filter_all.sh $data.data $min_out_attr $max_out_attr $min_out_var
$max_out_var
      #mv data.filt data.filtered
      #move the result files to the run directory
      cp data.filt $cdir/run/$data.data
      cp data.filt $cdir/run/$data.filter
    fi


cd $cdir/run

# randomise the data set
    if [ $randomise -eq 1 ]
    then
        #count how many lines the data file has
      nlines=`nl -n ln  $cdir/run/$data.data | tail -1 | tr -s ' '| tr -s
'\t' | cut -f1 -d' '`

        #generate a column with random values with the size of the data file
      $cdir/c/rand $nlines > random.tmp

        #rearange the file in the randomised order
      paste -d ' ' random.tmp $data.data > tmp.data
      sort -g tmp.data | cut -f2- -d' ' > $data.data
      cp $data.data $data.random

        rm tmp.data
    fi

#produce the two neural net pattern files

    #scale the data, either from [minvalue;maxvalue] to [0.0;1.0]
    #use logarithms if required
    if [ $logdata -eq 1 ]
    then
        $cdir/tcl/scale.tcl $cdir/run/$data.data logdata
    elif [ $expdata -eq 1 ]
    then
        $cdir/tcl/scale.tcl $cdir/run/$data.data expdata
    elif [ $softmax -eq 1 ]
    then
        $cdir/tcl/scale.tcl $cdir/run/$data.data softmax
    else
        $cdir/tcl/scale.tcl $cdir/run/$data.data
    fi


    # produce the two files
    if [ $test_prc -eq 0 ]
    then
    $cdir/tcl/conv1snns.tcl $cdir/run/$data.scaled $test_prc
    else
    $cdir/tcl/conv2snns.tcl $cdir/run/$data.scaled $test_prc
    fi

#end of [ $tt -eq 0 ]
fi

#create a log file and put a header
    echo "#Batch started at "`date` #>> ./$data"_"$net.log


#batch processing
    if [ $batch -eq 1 ]
    then
      echo "batch processing"

      #initialise the network
      cat $cdir/batch/batch_multiple_init.bat | sed s/nnet/$net.net/g >
batch_multiple_init.tmp
      batchman -q -f batch_multiple_init.tmp

    #prepare a temporary batch script with current data
      if [ $tt -eq 0 ]
      then
          if [ $self_snns -eq 1 ]
          then
            if [ $test_prc -eq 0 ]
              then
                cat $cdir/batch/snns_multiple.bat | sed
s/strain/$data"_train.pat"/g | sed s/stest/$data"_train.pat"/g >
$cdir/batch/batch.tmp
            else
        cat $cdir/batch/snns_multiple.bat | sed
s/strain/$data"_train.pat"/g | sed s/stest/$data"_test.pat"/g >
$cdir/batch/batch.tmp
      fi
     else
      if [ $test_prc -eq 0 ]
       then
        cat $cdir/batch/batch_multiple_notest.bat | sed
s/strain/$data"_train.pat"/g | sed s/stest/$data"_test.pat"/g >
$cdir/batch/batch.tmp
      else
        cat $cdir/batch/batch_multiple.bat | sed
s/strain/$data"_train.pat"/g | sed s/stest/$data"_test.pat"/g >
$cdir/batch/batch.tmp
      fi
     fi
   else
     cat $cdir/batch/batch_multiple.bat | sed s/strain/$traindata.data/g
| sed s/stest/$testdata.data/g > $cdir/batch/batch.tmp
   fi
    mv $cdir/batch/batch.tmp $cdir/run

   #run the batch and log it
    arg1=$p1min
   arg2=$p2min
   arg3=`date +%s`
    outcrt=0
   cd $cdir/run

    # count the number of columns
    ncols=`head -1 $data.data | wc -w`

    # determine the minimum and maximum for the (logarithm) values of the
variable to be determined
    if [ $no_scalling -eq 1 ]
    then
     echo -n "0.0" > minmax.tmp
   else
     echo -n `cat $cdir/run/$data.data | sort -g -k$ncols  | uniq -c -
f$(($ncols-1))| tr -s ' '| tr '\t' ' ' | cut -f3- -d' '| head -1 | cut -
f$ncols -d' '` > minmax.tmp
   fi
   echo -n " " >> minmax.tmp
   echo `cat $cdir/run/$data.data | sort -g -k$ncols  | uniq -c -
f$(($ncols-1))| tr -s ' '| tr '\t' ' ' | cut -f3- -d' '| tail -1 | cut -
f$ncols -d' '` >> minmax.tmp

   while test $arg1 -le $p1max
   do

     while test $arg2 -le $p2max
     do

      if [ $self_snns -eq 1 ]
      then
         cat ./batch.tmp | sed s/arg1/$arg1/g | sed s/arg2/$arg2/g |
sed s/error_step/$error_step/g | sed s/bcycles/$cycle_step/g >
./batch.gen.tmp
        batchman -s -q -f ./batch.gen.tmp >> $cdir/$data.snns.out #>
/dev/null

        crtcycles=`tail -3 $cdir/$data.snns.out | head -1 | cut -f5 -
d' '`

        echo -n "saturation $arg1.$arg2 $crtcycles  train "
        $cdir/tcl/eval.tcl train.res `cat minmax.tmp`
        cp corr.tmp corr_train.tmp
        if [ 0 -lt $test_prc ]
        then
          echo -n "saturation   $arg1 $arg2 $crtcycles  test  "
          $cdir/tcl/eval.tcl test.res `cat minmax.tmp`
          #$cdir/tcl/correlation.awk corr_test.tmp

        fi
        $cdir/tcl/correlation.awk corr_train.tmp
        #rm $cdir/run/test_sqerr.tmp
      else
      #...do the while-cycles loop

      while test $crtcycles -le $bcycles
      do

         cat ./batch.tmp | sed s/arg1/$arg1/g | sed s/arg2/$arg2/g |
sed s/snumber/$arg3/g | sed s/bcycles/$crtcycles/g > ./batch.gen.tmp
        batchman -s -q -f ./batch.gen.tmp  >> $cdir/batchman.out #>
/dev/null

        # save the resulting net
        # cp tmp_net.net net_$arg1.$arg2.$crtcycles.net

         #---begin analysis


        # update the old values before running analysis

        sqerr_prev=$sqerr_crt
        corr_prev=$corr_crt

        #the snns early stopping method does not require any
evaluation
        # print results for the train set
        echo -n "$arg1      $arg2 $crtcycles  "
        echo -n "train      "

        # determine the average absolute and relative error

        minmax=`cat minmax.tmp`
        if [ $logdata -eq 1 ]
        then
          $cdir/tcl/eval.tcl train.res log `cat minmax.tmp`
        elif [ $expdata -eq 1 ]
        then
          $cdir/tcl/eval.tcl train.res exp `cat minmax.tmp`
        elif [ $softmax -eq 1 ]
        then
          $cdir/tcl/eval.tcl train.res softmax `cat minmax.tmp`
`cat meandev.tmp`
        else
          $cdir/tcl/eval.tcl train.res `cat minmax.tmp`
        fi

        # save the results
        # cp train_eval_val.res train.$arg1.$arg2.$crtcycles.res
        # determine the correlation between actual data and predicted

        $cdir/tcl/correlation.awk corr.tmp
        cp corr.tmp corr_train.tmp

        if [ 0 -lt $test_prc ]
        then
          # print results for the test set
          echo -n "$arg1  $arg2 $crtcycles  "
          echo -n "test   "

          # determine the error and correlation for the current
test set
          if [ $logdata -eq 1 ]
          then
            $cdir/tcl/eval.tcl test.res log `cat minmax.tmp`
          elif [ $expdata -eq 1 ]
          then
            $cdir/tcl/eval.tcl test.res exp `cat minmax.tmp`
         elif [ $softmax -eq 1 ]
          then
            $cdir/tcl/eval.tcl test.res softmax `cat minmax.tmp`
`cat meandev.tmp`
         else
            $cdir/tcl/eval.tcl test.res `cat minmax.tmp`
         fi
         cp corr.tmp corr_test.tmp

         # determine the correlation between actual data and
predicted for the test set
         $cdir/tcl/correlation.awk corr.tmp
         corr_crt=`$cdir/tcl/correlation.awk corr.tmp`
         cp corr.tmp corr_test.tmp
         sqerr_crt=`cat $cdir/run/test_sqerr.tmp`

        else
          #run the tests using the whole (train) dataset
          corr_crt=`$cdir/tcl/correlation.awk corr.tmp`
          cp corr.tmp corr_test.tmp
          sqerr_crt=`cat $cdir/run/train_sqerr.tmp`
        fi

        #save the results
        #cp test_eval_val.res test.$arg1.$arg2.$crtcycles.res
        #cp corr_test.tmp corr_test.$arg1.$arg2.$crtcycles.tmp
        #cp corr_train.tmp corr_train.$arg1.$arg2.$crtcycles.tmp
        # gnuplot -geometry 700x700 -persist $cdir/plot/corr.plot


        #check whether saturation was reached using either square
error or correlation.
        # 4 methods of early stopping
         # self_snns - use MSE, as reported by SNNS (above)
         # self_err - use MSE, as calculated using eval.tcl
         # self_corr - use correlation, as calculated using eval.tcl
         # self_both - use both MSE and correlation, as calculated
by eval.tcl
         # note: error must reach 0 and correlation must reach 1;
         # this is why the comparison (compare) is opposite for the
two cases


        if [ $self_err -eq 1 ]
        then
         err_res=`$cdir/c/compare $sqerr_prev $sqerr_crt
$error_step`
         #echo "$sqerr_prev $sqerr_crt $error_step => $err_res"

         if [ $err_res -eq 0 ]
         then
```

```
                      cat buff.tmp
                      crtcycles=$bcycles
                      #rm $cdir/run/test_sqerr.tmp
                      sqerr_prev=0
                      sqerr_crt=0
                  else

                      echo -n "saturation $arg1 $arg2 $crtcycles  train " >
buff.tmp
                      $cdir/tcl/eval.tcl train.res `cat minmax.tmp` >>
buff.tmp
                      $cdir/tcl/correlation.awk corr_train.tmp >> buff.tmp
                      if [ 0 -lt $test_prc ]
                      then

                        echo -n "saturation     $arg1 $arg2 $crtcycles  test
      " > buff.tmp
                        $cdir/tcl/eval.tcl test.res `cat minmax.tmp` >>
buff.tmp
                        $cdir/tcl/correlation.awk corr_test.tmp >> buff.tmp

                      fi
                  fi
                fi


                if [ $self_corr -eq 1 ]
                then
                  err_res=`$cdir/c/compare $corr_crt $corr_prev $error_step`
                  #echo "$corr_prev $corr_crt $error_step => $err_res"

                  if [ $err_res -eq 0 ]
                  then
                      cat buff.tmp
                      crtcycles=$bcycles
                      #rm $cdir/run/test_sqerr.tmp
                      corr_prev=0
                      corr_crt=0
                  else
                      echo -n "saturation $arg1 $arg2 $crtcycles  train " >
buff.tmp
                      $cdir/tcl/eval.tcl train.res `cat minmax.tmp` >>
buff.tmp
                      $cdir/tcl/correlation.awk corr_train.tmp >> buff.tmp
                      echo -n "saturation $arg1 $arg2 $crtcycles  test  " >
buff.tmp
                      $cdir/tcl/eval.tcl test.res `cat minmax.tmp` >>
buff.tmp
                      $cdir/tcl/correlation.awk corr_test.tmp >> buff.tmp

                  fi
                fi


                if [ $self_both -eq 1 ]
                then
                  corr_res=`$cdir/c/compare $corr_crt $corr_prev $error_step`
                  err_res=`$cdir/c/compare $sqerr_prev $sqerr_crt
$error_step`
                  #echo "$corr_prev $corr_crt $error_step => $err_res"

                  if [ $(($err_res + $corr_res)) -eq 0 ]
                  then
                      cat buff.tmp
                      crtcycles=$bcycles
                      #rm $cdir/run/test_sqerr.tmp
                      corr_prev=0
                      corr_crt=0

                  else
                      echo -n "saturation $arg1 $arg2 $crtcycles  train " >
buff.tmp
                      $cdir/tcl/eval.tcl train.res `cat minmax.tmp` >>
buff.tmp
                      $cdir/tcl/correlation.awk corr_train.tmp >> buff.tmp
                      echo -n "saturation $arg1 $arg2 $crtcycles  test  " >>
buff.tmp
                      $cdir/tcl/eval.tcl test.res `cat minmax.tmp` >>
buff.tmp
                      $cdir/tcl/correlation.awk corr_test.tmp >> buff.tmp

                  fi
                fi


                #---end analysis



            crtcycles=$(($crtcycles+$cycle_step))
            done
          #...end of the while-cycles loop
          fi

          crtcycles=$cycle_step
          arg2=$(($arg2+$arg2_step))
          #This is VERY important: don't forget to re-initialise the network
after each run;
          #Otherwise, the new round of experiments uses the old values
          batchman -q -f batch_multiple_init.tmp

          done

        crtcycles=$cycle_step
      arg2=$p2min
      arg1=$(($arg1+$arg1_step))
      done

    fi


#single processing
    if [ $single -eq 1 ]
    then
      echo "single processing"

      #prepare a temporary batch script with current data
      #cd $cdir/run
      argdate=`date +%s`
      cat $cdir/batch/batch_single.bat | sed s/nnet/$net.net/g | sed
s/strain/$data"_train.pat"/g | sed s/stest/$data"_test.pat"/g |\
                         sed s/arg1/$param1/g | sed s/arg2/$param2/g | sed
s/bcycles/$bcycles/g  | sed s/snumber/$argdate/g  > $cdir/run/batch.tmp

        batchman -s -q -f batch.tmp
      #>>./$data"_"$net.log

#---begin analysis
```

```
#second part: do the analysis of the test file

    # count the number of columns
    ncols=`head -1 $data.data | wc -w`

    # determine the minimum and maximum for the (logarithm) values of the
variable to be determined
    if [ $no_scalling -eq 1 ]
    then
      echo -n "0.0" > minmax.tmp
    else
      echo -n `cat $cdir/run/$data.data | sort -g -k$ncols  | uniq -c -
f$(($ncols-1))| tr -s ' '| tr '\t' ' ' | cut -f3- -d' '| head -1 | cut -
f$ncols -d' '` > minmax.tmp
    fi
    echo -n " " >> minmax.tmp
    echo `cat $cdir/run/$data.data | sort -g -k$ncols  | uniq -c -f$(($ncols-
1))| tr -s ' '| tr '\t' ' ' | cut -f3- -d' '| tail -1 | cut -f$ncols -d' '`
>> minmax.tmp


    # determine the average absolute and relative error
    echo -n "test:       "

    if [ $logdata -eq 1 ]
    then
      $cdir/tcl/eval.tcl test.res log `cat minmax.tmp`
    elif [ $expdata -eq 1 ]
    then
      $cdir/tcl/eval.tcl test.res exp `cat minmax.tmp`
    elif [ $softmax -eq 1 ]
    then
      $cdir/tcl/eval.tcl test.res softmax `cat minmax.tmp` `cat meandev.tmp`
    else
      $cdir/tcl/eval.tcl test.res `cat minmax.tmp`
    fi

    # determine the correlation between actual data and predicted
    $cdir/tcl/correlation.awk corr.tmp
    cp corr.tmp corr_test.tmp


    #echo -n "$arg1     $arg2 $crtcycles  "

    if [ $logdata -eq 1 ]
    then
      $cdir/tcl/eval.tcl train.res log `cat minmax.tmp`
    elif [ $expdata -eq 1 ]
    then
      $cdir/tcl/eval.tcl train.res exp `cat minmax.tmp`
    elif [ $softmax -eq 1 ]
    then
      $cdir/tcl/eval.tcl train.res softmax `cat minmax.tmp` `cat meandev.tmp`
    else
      $cdir/tcl/eval.tcl train.res `cat minmax.tmp`
    fi
    $cdir/tcl/correlation.awk corr.tmp
    cp corr.tmp corr_train.tmp


    gnuplot -geometry 700x700 -persist $cdir/plot/corr.plot
```

```
#---end analysis

fi

#footer for the log file
echo "#Batch ended at `date`" #>> ./$data"_"$net.log


#clear the temporary files
rm $cdir/run/*
```

9.4.1.2. B.1.2 Script to convert CSV files to SNNS format (conv2snns.tcl)

```
#!/usr/bin/tclsh
# conv2snns.tcl
# convert to format suitable for SNNS use.
# split file to two files
# one is for training and another is for test (*_train.pat, *_test.pat)

    set logdata 0
    set noscale 0

    if { ($argc > 2) } {
        error "Usage: conv2snns.tk originalfile"
    }

# percentage of test data
    set percent 10

    if { ($argc == 2) } {
        set percent [lindex $argv 1]
#     puts "$percent percents test data"
    }
    puts "$percent percents test data"


    set file_in [lindex $argv 0]
    set fin [open $file_in  r]

    set basename [string range $file_in 0 [expr [string last . $file_in] -1]]
    set file_out1 [format "%s%s" $basename "_train.pat"]
    set file_out2 [format "%s%s" $basename "_test.pat"]

    set fout1 [open $file_out1  w]
    set fout2 [open $file_out2  w]

    set count 0

    set I   0
    set i_test 0
    set i_train 0
    set count_for_test 0
    set count_for_train 0
    set errline 0

# do preliminary analysis of the data
    gets $fin line
    split $line
    set length [llength $line]
    set count 1
    while {[gets $fin line] >= 0 }  {

    # count is the total number of samples
      incr count

    # check for samples with wrong number of fields
        split $line
      if {$length != [llength $line]} {
          incr errline
          error “Wrong number of fields at line $errline : $length fields
expected, [llength $line] obtained”
      }
    }
    seek $fin 0

# set the sizes of the train / test sets
    set count_for_test [expr $count * $percent/100]
    set count_for_train [expr $count - $count_for_test]
    puts stdout “patterns: all = train + test :  $count = $count_for_train +
$count_for_test”

# create the headers
    set date [exec date]
    puts $fout1 “SNNS pattern definition file V1.4”
    puts $fout1 “generated at $date \n\n”
    puts $fout1 “No. of patterns      : $count_for_train”
    puts $fout1 “No. of input units  : [expr $length - 1]”
    puts $fout1 “No. of output units : 1\n”
    puts $fout2 “SNNS pattern definition file V1.4”
    puts $fout2 “generated at $date \n\n”
    puts $fout2 “No. of patterns      : $count_for_test”
    puts $fout2 “No. of input units  : [expr $length - 1]”
    puts $fout2 “No. of output units : 1\n”

# produce the files
    while {[gets $fin line] >= 0 }  {
        incr I
        split $line

      if {[expr $i%(100/$percent)] == 0 && $i_test < $count_for_test} {

          # write to test data set
          incr i_test
          set fout $fout2

          puts $fout “# Input pattern $i_test:”

          set crtarg 0
          while { $crtarg < [expr $length - 1]} {
              puts -nonewline $fout “[lindex [split $line] $crtarg] “
              incr crtarg
          }
          puts $fout “”
          puts $fout “# Output pattern $i_test:”
          puts $fout [lindex $line [expr $length - 1 ]]
      } else {

          # write to train data set
          incr i_train
          set fout $fout1
          puts $fout “# Input pattern $i_train:”
          set crtarg 0
          while { $crtarg <  [expr $length - 1] } {
              puts -nonewline $fout "[lindex [split $line] $crtarg] "
              incr crtarg
          }
          puts $fout ""
          puts $fout "# Output pattern $i_train:"
          puts $fout [lindex $line [expr $length - 1 ]]
      }
  }

close $fin
close $fout1
close $fout2
```

B.1.3 Script to normalise input data (scale.tcl)

```
#!/usr/bin/tclsh
#tcl script to normalise the input data - translate data from
[minvalue;maxvalue] to [0.0;1.0]
#written by Bogdan Ghita 6/08/2001

#usage scale.tcl dataset [logdata | softmax]


      set shift 1.0
        set logdata 0
      set expdata 0
      set softmax 0
        set noscale 0
      set lambda 1

        if { ($argc > 2) } {
                error "Usage: scale.tcl originalfile \[no_scale \| logdata \|
expdata \] "
        }

        foreach optarg $argv {
            switch $optarg {
                no_scale { set noscale 1 }
                logdata { set logdata 1 }
            expdata { set expdata 1 }
                softmax { set softmax 1 }
            }
        }

        set file_in [lindex $argv 0]
        set fin [open $file_in  r]

        set basename [string range $file_in 0 [expr [string last . $file_in]
-1]]
        set file_norm [format "%s%s" $basename ".scaled"]
      set fmean [open "meanstd.tmp" w]
        set fnorm [open $file_norm  w]

        set count 0

# normalise the file; the attributes and the output have to have values
between 0 and 1

    # get the number of columns
    gets $fin line
    split $line
    set length [llength $line]
    seek $fin 0

    # set the minimum and maximum initial values for all the columns
    for {set crtarg 0} { $crtarg < $length } {incr crtarg} {
        set minval($crtarg) 1000000.0
        set maxval($crtarg) 0.000001
      set sum2($crtarg) 0.0
      set sum($crtarg) 0.0
      set mean($crtarg) 0.0
      set stdev($crtarg) 0.0
    }


    # determine maximum and minimum values for each of the columns
    while {[gets $fin line] >= 0 }  {
      set count [ expr $count + 1 ]
        for {set crtarg 0} {$crtarg < $length} {incr crtarg} {
          set value [ expr ((1.0 * [lindex [split $line] $crtarg]) + $shift)
]
          set sum2($crtarg) [expr ( $sum2($crtarg)+ $value*$value )]
          set sum($crtarg) [expr ( $sum($crtarg)+ $value ) ]
          if { $maxval($crtarg) < $value } {
            #update maxval
                set maxval($crtarg) $value
            }
            if { $minval($crtarg) > $value } {
            #update minval
                set minval($crtarg) $value
            }

        }

    }
    seek $fin 0

    for {set crtarg 0} {$crtarg < $length } {incr crtarg} {
      set mean($crtarg) [ expr ($sum($crtarg)/(1.0*$count)) ]
      set t1 [ expr $sum2($crtarg)/(1.0*$count) ]
      set t2 [ expr (2*$mean($crtarg)*$sum($crtarg))/(1.0*$count) ]
      set t3 [ expr $mean($crtarg)*$mean($crtarg) ]
      #set stdev($crtarg) [ expr ($sum2($crtarg)/($count*$count)) -
(2*$mean($crtarg)*$sum($crtarg))/($count*$count)) +
(($sum($crtarg)*$sum($crtarg))*($sum($crtarg)*$sum($crtarg)))/($count*$count)
]
      set stdev($crtarg) [ expr sqrt($t1 - $t2 + $t3) ]
        puts "$crtarg -
$minval($crtarg)\t$maxval($crtarg)\t$mean($crtarg)\t$stdev($crtarg)"
      # = $t1 + $t2 + $t3"

    }


    #normalise to [0.0,1.0] based on minval, maxval, mean, stdev
      #linear:    y = (x-minval)/(maxval-minval)
      #log:       y = (log(x)-log(minval))/(log(maxval)-log(minval))
      #exp:       y = log10((9*(10x-10minval)/(10maxval-10minval))+1)
            10x <=> pow (10,x)
      #softmax:   y = 1/(1+exp(-(x-mean)/((lambda*stdev)/6.28318)))
    while {[gets $fin line] >= 0 }  {
        for {set crtarg 0} {$crtarg < [expr $length - 1] } {incr crtarg} {
          set value [expr ([lindex [split $line] $crtarg] + $shift) ]
          if { $logdata == 1 } {
            #log scaling
            puts -nonewline $fnorm "[expr ((log10($value) -
log10($minval($crtarg))) / (log10($maxval($crtarg)) -
log10($minval($crtarg))) )] "
          } elseif { $expdata == 2 } {
            #exp scaling
            puts $value
            puts exp($maxval($crtarg))
            puts -nonewline $fnorm "[expr log((1.71828*(exp($value)-
exp($minval($crtarg)))/(exp($maxval($crtarg))-exp($minval($crtarg))))+1.0)] "
          } elseif { $softmax == 1 } {
            #softmax scaling
            #xSigma = sx2 - 2 * xmean * sx1+ N*xmean*xmean
            puts -nonewline $fnorm "[expr (1/(1+exp(-($value-
$mean($crtarg))/(($lambda*$stdev($crtarg))/6.28318)))) ] "
          } else {
            #linear scaling
            puts -nonewline $fnorm "[expr (($value - $minval($crtarg)) /
($maxval($crtarg) - $minval($crtarg)))] "
          }


        }
        set value [expr ([lindex $line [expr $length - 1] ] + $shift) ]
        if { $logdata == 1 } {
          #log scaling
          puts $fnorm "[expr ((log10($value) - log10($minval($crtarg))) /
(log10($maxval($crtarg)) - log10($minval($crtarg))))] "
      } elseif { $expdata == 1 } {
          #exp scaling
          puts $fnorm "[expr log(((1.71828*(exp($value)-
exp($minval($crtarg))))/(exp($maxval($crtarg))-exp($minval($crtarg))))+1.0)]
"
      } elseif { $softmax == 1 } {
          #softmax scaling
          puts $fnorm "[expr (1/(1+exp(-($value-$mean([expr $length -
1]))/(($lambda*$stdev([expr $length - 1]))/6.28318)))) ] "
      } else {
          #linear scaling
          puts $fnorm "[expr (($value - $minval($crtarg)) / ($maxval($crtarg)
- $minval($crtarg)))] "

      }

    }

    puts $fmean "$mean([expr $length - 1]) $stdev([expr $length - 1]) "


    seek $fin 0
    close $fin
    close $fnorm
```

B.1.4 Script to evaluate the accuracy of the neural network output (eval.tcl)

```
#!/usr/bin/tclsh
# program to calculate the error produced by a neural network calculation
# written by Bogdan Ghita 1/08/2001
# input has to be an SNNS-produced result file
# the format of the file should be:
#      some header
#      #pattern_number
#      trained output
#      actual output

# usage: eval.tcl  resultfile \[log \| softmax\] minval maxval [mean stdev]
# minval, maxval - the limits of the original domain (from which it was
translated to \[0.0 ; 1.0\]

set log 0
set exp 0
set softmax 0
set lambda 1
set shift 1.0

foreach optarg $argv {
    switch $optarg {
      logdata      { set log 1 }
      expdata { set exp 1 }
      softmax { set softmax 1 }
    }
}

if {$argc == 1} {
    set minval 0.0
    set maxval 1.0
#    puts "Using default values: minval = $minval     maxval = $maxval"

} elseif {$argc == 3} {
    set minval [ expr [lindex $argv 1] + $shift ]
    set maxval [expr [lindex $argv 2] + $shift ]
    puts "Using input values: minval = $minval  maxval = $maxval"

} elseif {$argc == 4} {
    set minval [expr ([lindex $argv 2] + $shift) ]
    set maxval [expr ([lindex $argv 3] + $shift) ]
    puts "Using input values (logarithmic data): minval = $minval maxval =
$maxval"

} elseif {$argc == 6} {
    set softmax 1
    set minval [lindex $argv 2]
    set maxval [lindex $argv 3]
    set mean [lindex $argv 4]
    set stdev [lindex $argv 5]
#    puts "Using input values (logarithmic data): minval = [expr
exp($minval)]     maxval = [expr exp($maxval)]"
} else {
    error "usage: eval.tcl  resultfile \[log \| softmax\] minval maxval
\[mean stdev\]     "
}

#if {$log == 1}
#    puts "minval = $minval / exp(minval) = [expr exp($minval)]   maxval =
$maxval / exp(maxval) = [expr exp($maxval)]"
#} else {
#    puts "minval = $minval   maxval = $maxval"
#}

set file_in [lindex $argv 0]
set fin [open $file_in r]
set i 0
set pat 0
set err 0
set errsum 0.0
set avgsum 0.0
set relerrsum 0.0
set abserrsum 0.0
set sqabserrsum 0.0
set sqrelerrsum 0.0
set avgabssqerr 0.0
set avgrelsqerr 0.0

set basename [string range $file_in 0 [expr [string last . $file_in] -1]]
set fileres1 [format "%s%s" $basename "_eval_val.res"]
set fileres3a [format "%s%s" $basename "_eval_relerr.res"]
set fileres3b [format "%s%s" $basename "_eval_abserr.res"]
set fileres4 "corr.tmp"
set fileres5 [format "%s%s" $basename "_sqerr.tmp"]
set fileres6 "corr_raw.tmp"

set fout1 [open $fileres1 w]
#set fout2 [open $fileres2 w]
set fout3a [open $fileres3a w]
set fout3b [open $fileres3b w]
set fout4 [open $fileres4 w]
set fout5 [open $fileres5 w]
set fout6 [open $fileres6 w]

while {[gets $fin line] >= 0 }  {
    incr i

    if [string match #* $line] {
      incr pat

      if { $log == 1} {

          #scaling: y = ( log(x) - log(minval) ) / ( log(maxval) -
log(minval) )
          #reverse: x = exp ( log(minval) - y*log(minval) + y*log(maxval) )
          gets $fin line
          puts -nonewline $fout6 "$line "
          set val [expr (pow(10,($line*(log10($maxval)-
log10($minval))+log10($minval))) - $shift) ]
          gets $fin line
          puts $fout6 "$line"
          set pred [expr (pow(10,($line*(log10($maxval)-
log10($minval))+log10($minval))) - $shift) ]

      } elseif { $exp == 1 } {
          gets $fin line
          puts -nonewline $fout6 "$line "
          puts $line
          #puts [expr (((exp($maxval)-exp($minval))*(exp($line)-
exp($minval))/1.71828) + exp($minval)) - $shift ]
          set val [expr log(((exp($maxval)-exp($minval))*(exp($line)-
1.0)/1.71828) + exp($minval)) - $shift ]
          gets $fin line
          puts $fout6 "$line"
          #set pred [expr log(((exp($maxval)-exp($minval))*(exp($line)-
1.0)/1.71828) + exp($minval)) - $shift ]
          set pred [expr log(((exp($maxval)-exp($minval))*(exp($line)-
1.0)/1.71828) + exp($minval)) - $shift ]
      } elseif { $softmax == 1 } {
          #reverse: x = mean + (lambda*stdev*0.5*log(-1+(1/(1-y)))/PI)
```

```
     #0.5/PI=0.15915
     gets $fin line
     if { $line == 1.0 } {
       set line 0.99999
     }
     set val [ expr $mean + 0.15915*$lambda*$stdev*log(-1+(1/(1-$line)))
]
     gets $fin line
     if { $line == 1.0 } {
       set line 0.99999
     }
     set pred [ expr $mean + 0.15915*$lambda*$stdev*log(-1+(1/(1-
$line))) ]

   } else {
     gets $fin line
     set val [expr $line*($maxval-$minval)+$minval - $shift ]
     gets $fin line
     set pred [expr $line*($maxval-$minval)+$minval - $shift ]
   }

   set relerr [expr (abs($val-$pred))/$val]
   set abserr [expr (abs($val-$pred))]
   set sqabserr [expr ($val-$pred)*($val-$pred)]
   set sqrelerr [expr ($val-$pred)*($val-$pred)/($val*$val)]

   set avgsum [expr ($avgsum+$val)]
   set abserrsum [expr ($abserrsum+$abserr)]
   set relerrsum [expr ($relerrsum+$relerr)]
   set sqabserrsum [expr ($sqabserrsum+$sqabserr)]
   set sqrelerrsum [expr ($sqrelerrsum+$sqrelerr)]

   puts $fout1 "$val $pred $relerr"
#  puts $fout1 "$val $pred $err $sqabserr"

   puts $fout3a "$relerr"
   puts $fout3b "$abserr"
   puts $fout4 "$val $pred"
  }
}

set avgrelerr [ expr ($relerrsum*1.0/$pat) ]
set avgabserr [ expr ($abserrsum*1.0/$pat) ]
set avgabssqerr [ expr ($sqabserrsum*1.0/$pat) ]
set avgrelsqerr [ expr ($sqrelerrsum*1.0/$pat) ]

puts $fout5 "$avgabssqerr"
#puts -nonewline "[expr ($avgsum/$pat)]   $avgrelerr  $avgabserr
   $avgrelsqerr      $avgabssqerr      "
puts -nonewline "$pat $sqabserrsum  $avgrelsqerr     $avgabssqerr      "

close $fin
close $fout1
close $fout3a
close $fout3b
close $fout4
close $fout5
```

Appendix B.2 – NS scripts

B.2.1 Simulation script for a three-tier topology (net.tcl)

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
      $ns color 1 Blue
      $ns color 2 Red
      $ns color 3 Green
      $ns color 4 Yellow
      $ns color 5 Orange
      $ns color 6 Orange

#Enable trace support
      Trace set show_tcphdr_ 1

#Enable random seed
      set ns-random-seed 0
      set rng [new RNG]
      $rng seed 0

#Set topology parameters - static
      #set LanSize 30
      #set QueueLimit 20
      #set QueueLimitBackbone 20

      #set bw_core 10Mb
      #set bw_core_bneck 5MB
      #set bw_access 10Mb
      #set bw_access_bneck 2Mb



#Set topology parameters - random
      set LanSize [expr 2 * (10 + [ $rng integer 20 ] ) ]
      set QueueLimit [ expr $LanSize / 2 ]
      set QueueLimitBackbone [ expr 3 * $LanSize / 2 ]

      set bw_core 10000000
      set bw_core_bneck 5000000
      set bw_access 10000000
      set bw_access_bneck 2000000

      set delay_access 0.01
      set delay_gw 0.05
      set delay_core 0.1


#Open the NAM trace file
      set nf [open out.nam w]
      $ns namtrace-all $nf

#Open the Trace file
      set tf [open out.tr w]
      $ns trace-all $tf
```

```
#Create nodes
      puts -nonewline "creating the topology - LANs with $LanSize nodes..."

      #Backbone nodes
      set r0 [$ns node]
      set r1 [$ns node]

      #Gateways
      set gw0 [$ns node]
      set gw1 [$ns node]
      set gw2 [$ns node]
      set gw3 [$ns node]

      #Endpoints
      for {set i 0 } { $i < $LanSize } { incr i } {
            set a_tcp($i) [$ns node]
            set b_tcp($i) [$ns node]
            set c_tcp($i) [$ns node]
            set d_tcp($i) [$ns node]

            }

      puts "topology created"

#Create links

      puts -nonewline "Creating the links..."

      #backbone link
      $ns duplex-link $r0 $r1 [$rng uniform $bw_core_bneck $bw_core] [$rng
uniform [ expr $delay_core / 2 ] $delay_core ] DropTail

      #gateway-core links
      $ns duplex-link $gw0 $r0 [$rng uniform $bw_access_bneck $bw_access]
$delay_gw DropTail
      $ns duplex-link $gw1 $r1 [$rng uniform $bw_access_bneck $bw_access]
$delay_gw DropTail
      $ns duplex-link $gw2 $r1 [$rng uniform $bw_access_bneck $bw_access]
$delay_gw DropTail
      $ns duplex-link $gw3 $r0 [$rng uniform $bw_access_bneck $bw_access]
$delay_gw DropTail


      #access links
      for {set i 0 } { $i < $LanSize } { incr i } {

            #setup using static parameters
            #$ns duplex-link $gw0 $a_tcp($i) $bw_access $delay_access
DropTail
            #$ns duplex-link $gw1 $b_tcp($i) $bw_access $delay_access
DropTail
            #$ns duplex-link $gw2 $c_tcp($i) $bw_access $delay_access
DropTail
            #$ns duplex-link $gw3 $d_tcp($i) $bw_access $delay_access
DropTail

            #setup using random parameters
            $ns duplex-link $gw0 $a_tcp($i) $bw_access [$rng uniform [ expr
$delay_access / 2 ] $delay_access ] DropTail
            $ns duplex-link $gw1 $b_tcp($i) $bw_access [$rng uniform [ expr
$delay_access / 2 ] $delay_access ] DropTail
            $ns duplex-link $gw2 $c_tcp($i) $bw_access [$rng uniform [ expr
$delay_access / 2 ] $delay_access ] DropTail
            $ns duplex-link $gw3 $d_tcp($i) $bw_access [$rng uniform [ expr
$delay_access / 2 ] $delay_access ] DropTail

      }

      puts "links created"

#Set Queue Size of access links to 10
      $ns queue-limit $gw0 $r0 $QueueLimit
      $ns queue-limit $gw1 $r1 $QueueLimit
      $ns queue-limit $gw2 $r1 $QueueLimit
      $ns queue-limit $gw3 $r0 $QueueLimit
      $ns queue-limit $r0 $r1 $QueueLimitBackbone

#Monitor the queues (for NAM)
      $ns duplex-link-op $gw0 $r0 queuePos 0.5
      $ns duplex-link-op $gw1 $r1 queuePos 0.5
      $ns duplex-link-op $gw2 $r1 queuePos 0.5
      $ns duplex-link-op $gw3 $r0 queuePos 0.5
      $ns duplex-link-op $r0 $r1 queuePos 0.5


#TCP settings
      Agent/TCP/FullTcp set interval_ 100ms
      Agent/TCP/FullTcp set segsperack_ 2



puts -nonewline "Creating the clients..."

for {set i 0 } { $i < $LanSize } { incr i } {

      #Setup a0-b0 TCP connection
            #net a
            #a - sender

            #setting the intial window
                  set initwin [ expr [$rng integer 3] + 1 ]
                  Agent/TCP/FullTcp set initial_window $initwin


            set tcp_a($i) [new Agent/TCP/FullTcp]
            $tcp_a($i) set class_ 2
            $tcp_a($i) set packetSize 1460
            $ns attach-agent $a_tcp($i) $tcp_a($i)

            #net b
            #b0 - sink
            set sink_b($i) [new Agent/TCP/FullTcp]
            $sink_b($i) set segsperack_ 2
            $ns attach-agent $b_tcp($i) $sink_b($i)
            $sink_b($i) listen

            $ns connect $tcp_a($i) $sink_b($i)
            $tcp_a($i) set fid_ 1


      #Setup a0-b0 FTP application
            set ftp_a($i) [new Application/FTP]
            $ftp_a($i) attach-agent $tcp_a($i)
            $ftp_a($i) set type_ FTP
```

```
#Setup c0-d0 TCP connection
      #net c
      #c0 - sender

      set tcp_c($i) [new Agent/TCP/FullTcp]
      $tcp_c($i) set class_ 2
      $ns attach-agent $c_tcp($i) $tcp_c($i)

      #net d
      #d0 - sink
      set sink_d($i) [new Agent/TCP/FullTcp]
      $sink_d($i) set segsperack_ 2
      $ns attach-agent $d_tcp($i) $sink_d($i)
      $sink_d($i) listen

      $ns connect $tcp_c($i) $sink_d($i)
      $tcp_c($i) set fid_ 2


#Setup c0-d0 FTP application
      set ftp_c($i) [new Application/FTP]
      $ftp_c($i) attach-agent $tcp_c($i)
      $ftp_c($i) set type_ FTP

incr i

#Setup a1-c1 TCP connection
#net a
#a - sender
      set tcp_a($i) [new Agent/TCP/FullTcp]
      $tcp_a($i) set class_ 2
      $ns attach-agent $a_tcp($i) $tcp_a($i)


#net c
#c1 - sink
      set sink_c($i) [new Agent/TCP/FullTcp]
      $sink_c($i) set segsperack_ 2
      $ns attach-agent $c_tcp($i) $sink_c($i)
      $sink_c($i) listen

      $ns connect $tcp_a($i) $sink_c($i)
      $tcp_a($i) set fid_ 3


#Setup a1-c1 FTP application
      set ftp_a($i) [new Application/FTP]
      $ftp_a($i) attach-agent $tcp_a($i)
      $ftp_a($i) set type_ FTP


#Setup b1-d1 TCP connection
      #net b
      #b - sender
      set tcp_b($i) [new Agent/TCP/FullTcp]
      $tcp_b($i) set class_ 2
      $ns attach-agent $b_tcp($i) $tcp_b($i)


      #net d
      #d - sink
      set sink_d($i) [new Agent/TCP/FullTcp]
      $sink_d($i) set segsperack_ 2
      $ns attach-agent $d_tcp($i) $sink_d($i)
```

```
            $sink_d($i) listen

            $ns connect $tcp_b($i) $sink_d($i)
            $tcp_b($i) set fid_ 4


      #Setup b1-d1 FTP application
            set ftp_b($i) [new Application/FTP]
            $ftp_b($i) attach-agent $tcp_b($i)
            $ftp_b($i) set type_ FTP

}

puts "clients created"

#Set simulation limits
      set min_start_limit 0.0
      set max_start_limit 1.0
      set min_duration_limit 1.0
      #set max_duration_limit 5.0
      set max_duration_limit 10.0

for {set i 0 } { $i < $LanSize } { incr i } {

      #Control a-c TCP

            set time [$rng uniform $min_duration_limit $max_duration_limit ]
            set starttime [$rng uniform $min_start_limit $max_start_limit ]

            set start($ftp_a($i)) [expr $starttime]
            set stop($ftp_a($i)) [expr $starttime + $time]

            $ns at $start($ftp_a($i)) "$ftp_a($i) start"
            $ns at $stop($ftp_a($i)) "$ftp_a($i) stop"


      #Control b-d TCP
            set time [$rng uniform $min_duration_limit $max_duration_limit ]
            set starttime [$rng uniform  $min_start_limit $max_start_limit]

            set start($ftp_c($i)) [expr $starttime]
            set stop($ftp_c($i)) [expr $starttime + $time]

            $ns at $start($ftp_c($i)) "$ftp_c($i) start"
            $ns at $stop($ftp_c($i)) "$ftp_c($i) stop"

      incr i

      #Control a-c TCP
            set time [$rng uniform $min_duration_limit $max_duration_limit ]
            set starttime [$rng uniform $min_start_limit $max_start_limit]

            set start($ftp_a($i)) [expr $starttime]
            set stop($ftp_a($i)) [expr $starttime + $time]

            $ns at $start($ftp_a($i)) "$ftp_a($i) start"
            $ns at $stop($ftp_a($i)) "$ftp_a($i) stop"


      #Control b-d TCP
            set time [$rng uniform $min_duration_limit $max_duration_limit ]
            set starttime [$rng uniform $min_start_limit $max_start_limit]

            set start($ftp_b($i)) [expr $starttime]
            set stop($ftp_b($i)) [expr $starttime + $time]

            $ns at $start($ftp_b($i)) "$ftp_b($i) start"
            $ns at $stop($ftp_b($i)) "$ftp_b($i) stop"
}


#Run the simulation
      puts  -nonewline "Start simulation..."
      $ns run
      puts "simulation finished"
```

B.2.2 Loop script to produce a batch of traces (loop.sh)

```
#!/bin/bash
#script to produce a batch of ns simulation rounds
#syntax: loop.sh rounds

i=0
rm *.save
while [ $i -lt $1 ]
do
      echo "i = $i"
      echo "i = $i" >>crti.save
      echo "`date` -  start"
      ../ns net.tcl
      echo "`date`  - finished"
      cp out.tr out_$i.tr
      tar -cvzf out_$i.tar.gz out_$i.tr
      rm out_$i.tr
      cat out.tr | grep " 0 1 " >  res
      cat out.tr | grep " 1 0 " >> res
      sort -g -k2 res > trace
      cp trace trace_$i
      i=$(($i+1))
done
```

Appendix B.3 – wget data collection script

B.3.1 Main data collection script (ryl.sh)

```
#!/bin/bash
#shell to retrieve random links from ryl - Random Yahoo Link

rm counter
rm exp.crt
touch output_tmp.wget
count=1
echo "$count" > exp.crt
tstamp=`date +%y-%m-%d-%H.%M`
echo "$tstamp" > tstamp
mkdir ./$tstamp 2> /dev/null
mkdir ./$tstamp/pages  2> /dev/null
crtdir=./$tstamp
#mkdir ~/tmp/ryl

http_client=wget
http_client_opts="--cache=off --tries=1 -A "*.html,*.htm" --directory-
prefix=./$tstamp/pages -a ./$tstamp/wget.log http://random.yahoo.com/bin/ryl"
http_client_opts=" ./$tstamp/wget.log http://random.yahoo.com/bin/ryl --dump-
header $crtdir/headers_tmp.wget --include --max-time 120  --output
\"$crtdir/output_tmp.wget\" --show-error --http1.0 --stderr
$crtdir/error_tmp.wget -L"
http_client_opts=" http://random.yahoo.com/bin/ryl --dump-header
thishost=`hostname -i`

killall -9 tcpdump 1>> ./tmp.log 2> ./tmp.log

tcpdump -ieth0 -w ./$tstamp/ryl.dump host $thishost and port 80 1>> ./tmp.log
2>> ./tmp.log &
tcpdump -ieth0 -w ./$tstamp/ryl_srv.dump host $thishost -s 300 and port 80
1>> ./tmp.log 2>> ./tmp.log &
tcpdump -ieth0 -w ./$tstamp/ryl_bk.dump host $thishost and not port 80  1>>
./tmp.log 2>> ./tmp.log &

killall -9 killer.sh  1>> ./tmp.log 2>> ./tmp.log
./killer.sh  1>> ./tmp.log 2>> ./tmp.log &

echo `date` - Batch started >> ./$tstamp/ryl_out.log

while [ $count -le `cat ./exp.max` ]
do
    echo "$count" > exp.crt
    date_start="`date +%s`"
    echo -e -n "$count\t`date +%H:%M:%S`" >> ./$tstamp/ryl_out.log
#start_stop.log

    $http_client $http_client_opts  1>> $crtdir/pages/$count.wget 2>>
$crtdir/stderr.wget
    cat $crtdir/headers_tmp.wget >> $crtdir/headers.wget
    cat $crtdir/output_tmp.wget >> $crtdir/output.wget
    cat $crtdir/error_tmp.wget >> $crtdir/error.wget

    date_end="$((`date +%s`-$date_start))"
    echo -e "\t`date +%H:%M:%S`\t$date_end" >> ./$tstamp/ryl_out.log
#start_stop.log
    sleep 5
    count=$(($count+1))
    killall -15 killer.sh

done
```

B.3.2 Thread maintenance script (killer.sh)

```
#!/bin/sh
# script to kill wget client in case it takes longer than 2 minutes to
download a file.

expno=`cat exp.crt`
tstamp=`cat tstamp`
echo "killer.sh started"
counter=120
while [ $expno -lt 10000 ]
do

      if [ `cat exp.crt` -eq $expno ]
      then
          if [ $counter -lt 5 ]
      then
              killall -9 wget
                  echo `date +%H:%M:%S`" exp $expno - wget killed" >>
./$tstamp/wget.log
            echo `date +%H:%M:%S`" exp $expno - wget killed" >>
./$tstamp/ryl_out.log
                  counter=125
            else
                  counter=$(($counter-1))
                  echo $counter > counter
      fi
      sleep 1
      else
      expno=`cat exp.crt`
          echo `date +%H:%M:%S`" exp $expno finished - $((120-$counter))" >>
./$tstamp/ryl_out.log
      counter=125
      fi
done
```

Appendix C – Publications

The list below presents the papers written and published during the PhD work programme. Those highlighted with a '*' are not included in this appendix but are available from the Network Research Group website (http://www.network-research-group.org).

* “Endpoint study of Internet paths and web pages transfers”, Mr Bogdan V. Ghita, Dr Steven M. Furnell, Dr Benn Lines, Prof. Emmanuel Ifeachor Campus Wide Information Systems, vol. 20, no. 3, pp90-97, 2003

“Endpoint study of Internet paths and web pages transfers”, Mr Bogdan V. Ghita, Dr Steven M. Furnell, Dr Benn Lines, Prof. Emmanuel Ifeachor, Proceedings of the Third International Network Conference (INC 2002), Plymouth, UK, 16-18 July 2002, pp261-270, 2002

“Non-intrusive IP Network Performance Monitoring for TCP Flows”, Mr Bogdan V. Ghita, Dr Benn Lines, Dr Steven M. Furnell, Prof. Emmanuel Ifeachor Proceedings of IEEE ICT2001, Bucharest, Romania, pp290-295, 4-7 June, 2001

“Network Quality of Service Monitoring for IP Telephony”, Mr Bogdan V. Ghita, Dr Steven M. Furnell, Dr Benn Lines, Mr Dominique Le Foll, Prof. Emmanuel Ifeachor, Internet Research, vol. 11, no. 1, pp26-34, 2001

* “IP Networks Performance Monitoring of Voice Flows for IP Telephony”, Mr Bogdan V. Ghita, Dr Steven M. Furnell, Dr Benn Lines, Mr Dominique Le Foll, Prof. Emmanuel Ifeachor, Proceedings of the Second International Network Conference (INC 2000), Plymouth, UK, pp145-155, 3-6 July, 2000

* “Measurement of IP Transport Parameters for IP Telephony”, Mr Bogdan V. Ghita, Dr

Steven M. Furnell, Dr Benn Lines, Prof. Emmanuel Ifeachor, Proceedings of PG Net 2000 – 1st Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, Liverpool, UK, pp31-36, 19-20 June, 2000

"Procede D'Evaluation de la Bande Passante D'Une Liaison Numerique", Pattent pending, no. 03 50056, 19/03/2003, applied for by Actema IPMS.

Endpoint study of Internet paths and web pages transfers

B. V. Ghita, S. M. Furnell, B. M. Lines, E. C. Ifeachor

University of Plymouth, Plymouth, United Kingdom
e-mail: bghita@jack.see.plymouth.ac.uk

Abstract

This paper presents the findings of a pilot study to provide information about the characteristics of current networks and data transfers. The main aim of the study was to infer the properties of a large number of network paths. In addition, the study produced statistics relating to the average size of a typical web page and both under the restriction of a single-point connection. The study was performed in two steps: trace collection followed by TCP per-flow analysis. The trace collection used the functionality of a random link generator, combined with an automatic HTTP retrieval tool. The TCP analysis was applied to the collected traces and it involved an offline TCP per-flow method developed in previous research.

Keywords

TCP connection analysis, Internet characteristics, web page size, web transfer features.

1 Introduction

The current status of the Internet is one of the issues being researched intensively. The first concerted initiative to evaluate the properties of the Internet belongs to Paxson. He deployed his Network Probe Daemon (NPD), established a measurement mesh to evaluate the characteristics of network paths, and generated and analysed the transfers running through this mesh (Paxson, 1999). Several bodies, such as the Active Measurement Project (AMP, 2002) and the National Internet Measurement Infrastructure, built on the concept of NPD) (NIMI, 2002) projects aim to describe the Internet from a holistic perspective by employing complex measurement infrastructures. A different view is embraced by passive traffic surveys, which capture and analyse data from backbone segments / endpoint networks (Thompson and Miller, 1997). The need for such information comes from both the research and commercial domains. The rationale is similar for the two cases: the marketing directions, as well as the improvements of current Internet-related technologies, have to be based on actual information rather than assumptions or previous studies.

All of the aforementioned measurement initiatives are very successful in their place, and they aim to answer the question 'How does the overall Internet behave?'. The study presented in this paper seeks to discuss the Internet characteristics from a different perspective: how the Internet is seen from an endpoint network and what are the characteristics of the data that may be retrieved from the Internet by other hosts connected to that respective endpoint network. Concluding, the question that this study aims to answer is 'How does Internet behave for *my* Internet traffic?', as would be asked by an endpoint network user / administrator.

2 Traffic collection

The study discussed by this paper presents analysis results based on two data sources: real traffic and artificially generated traffic. In both cases HTTP was used as the focused application for reasons of availability and convenience. It was observed before starting the experiments that most of the network TCP traffic is web browsing, which confirms the results of previous studies (Paxson, 1999). Also, as will be discussed later, artificial and random HTTP traffic was convenient to produce.

For the first option, i.e. capture real traffic, the hosts (approximately 15) within the Network Research Group (NRG) at University of Plymouth were used. The connectivity of the machines within the NRG is convenient for traffic capture, as they are all connected to a switch, and the capture machine was attached to the uplink of the switch through a hub. The traffic collection was performed continuously during spring 2002 for a period of two weeks and included only web traffic between the hosts in the NRG and hosts outside the UoP network. The second option, i.e. generate artificial traffic, allowed a more controlled approach to the data collection. The traffic was produced using the Random Yahoo Link page (RYL, 2002) from the Yahoo website, a CGI script that redirects a request to a random WWW page, taken from the Yahoo search engine database. The HTTP client used to perform the requests was wget (wget, 2002), a command-line HTTP retrieval tool, and the requests were controlled through a Linux shell script. The experiments in this case were also performed in two stages, but separately from the network segment traffic capture discussed above. It is known that at least one major event, in terms of network infrastructure changes, happened between the two experiments: an upgrade of the UoP network from a 100MB backbone / 10MB access speed to 1000MB backbone / 100MB access speed. As will be seen in the results section, all the network parameters (bandwidth, loss, delay) are improved for the second set of results. For both experiments, the traffic was captured using tcpdump (tcpdump, 2002), which was set to keep only the HTTP connections (using a *tcp and port 80* filter expression). The level of the monitored traffic was low in all cases and tcpdump did not report any dropped packets throughout the experiments. In all experiments, the traces were filtered offline in order to remove the unfinished or reseted connections, which could not be used for consistent analysis; the resulting figures are presented in Table 1.

Traffic type	Number of connections collected			
	2001		2002	
	Raw	Filtered	Raw	Filtered
Wget generated	15106	12469	16844	13674
Real	-	-	14288	11322

Table 1 – Capture statistics for the traffic collection experiments performed

3 Analysis

One of the aims of this paper is to advance the traffic analysis from an overall study, currently preferred for convenience and simplicity, to per-flow examination, in order to get an insight of the network conditions that are behind the traffic. The overall analysis studies present only the total figures of traffic (overall throughput in bytes, packets, or flows per second), the distribution of traffic per application (based on the port numbers), or the distribution of the packet lengths. The only concerted efforts in the area of TCP per-flow analysis were the ones made by Paxson in (Paxson, 1997a), (Paxson, 1999), which are quoted by most articles when discussing current characteristics of the Internet.

Two types of analysis were applied to the collected traces: network performance-related, to reveal the end-to-end network paths characteristics, and connection-related, to classify the web pages in terms of size and content. The network performance analysis was performed using a previously developed tool, described in (Ghita et al, 2001); the method employed is similar to other TCP flow analysers, like tcpanaly (Paxson, 1997b) and tcptrace (Osterman, 2002), with improvements for single point monitoring and network parameters inference. The connection analysis investigated the size and the content (object-wise) of the web pages for two reasons: to determine the average size of a page (together with the containing objects, e.g. images) and to establish the efficiency in practice of the HTTP 1.1 pipelining capabilities.

4 The Random Yahoo Link experiments - Results

4.1 Network topology

The UoP network is connected to the Internet, as mentioned before, via the UK academic network, JANET. As a result, the first 8 hops of all paths are part of the JANET infrastructure, and, implicitly, were common for all connections but the routes diverged at the exit from JANET, depending on the destination host. A separate experiment was carried out to estimate the number of individual paths explored within the performed experiments. A traceroute was run on a random subset of the sites (350 out of the 2744 unique servers which were used during the spring 2002 round of experiments) to see the number of different individual paths. The results are shown in Figure 1.

The number of routes differing by at least one hop was found to be as high as 180, figure that is approximately half of the number of hosts probed (the number of unique hops decreases towards the end of the graph due to path size, with an average hop count of 22.2 hops). Concluding, although the study was performed from a single point, this additional measurement indicates that the survey analysed a fairly large number of different Internet paths.

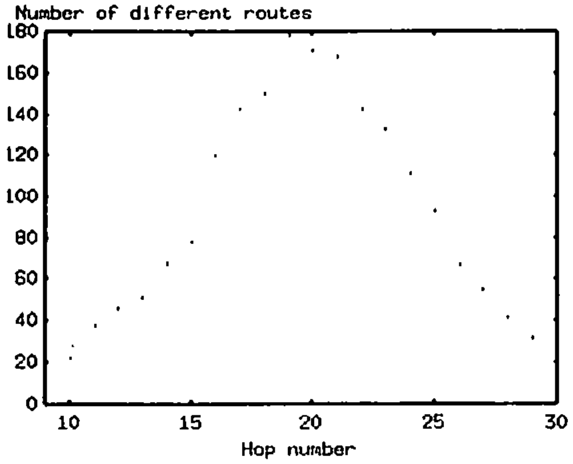


Figure 1 - Routing distribution, spring 2002 experiments

4.2 Round Trip Time results

The distribution of the RTT for the two sets of experiments is presented in Figure 2 (left). As can be observed, in both cases the average RTT values are very low for most of the connections, with an overall average of 200.5 ms for the first round of experiments and 136.5 ms for the second round. The difference between the figures may be associated with the network upgrade mentioned previously (unfortunately, there was no path information collected during the autumn 2001 experiments), as the shape of the distribution remained the same for the two sets of results.

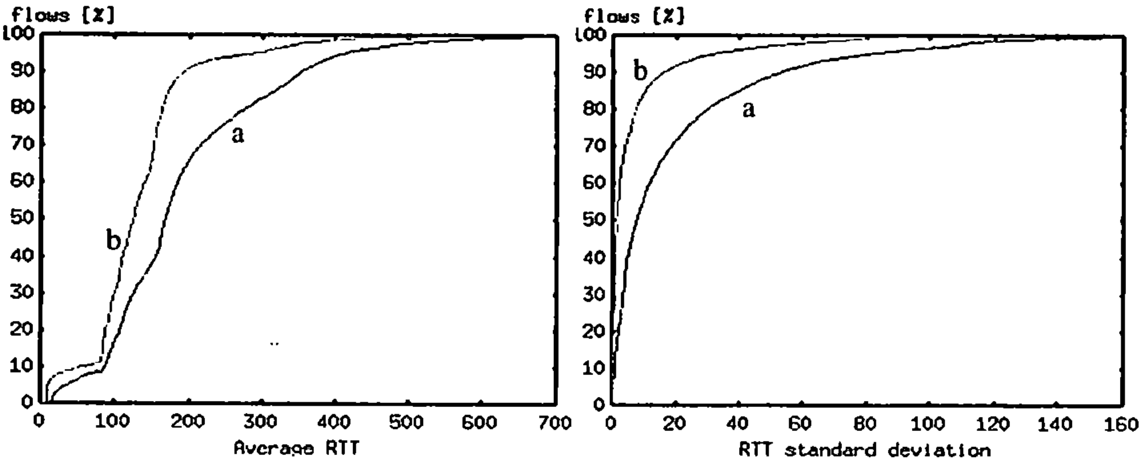


Figure 2 – left - RTT average [ms] cumulative distribution for: a) autumn 2001, b) spring 2002; right - RTT standard deviation [ms] cumulative distribution for: a) autumn 2001, b) spring 2002

Aside from the actual value of the RTT, the standard deviation of the RTT throughout a connection was calculated; the result is shown in Figure 2 (right). The average value of the standard deviation was 22.3 ms (10.4 % of the RTT averages) for the autumn 2001 round and 7.8 ms (4.7 % of the RTT averages) for spring 2002. The data results from spring 2002 indicate that, for 87% of the flows, the RTT standard deviation was 10 ms. This value is relevant as, at the moment, it is the default resolution for timers, at least for Linux based systems. Future work, aims to analyse the implications of these low figures for RTT estimation within the TCP clients, since the RTT variation plays an important role in the TCP retransmission mechanism (Jacobson and Karels, 1988). Since successive connections were made to different sites, conclusions could not be drawn with regard to the long / short term autocorrelation of either RTT average or RTT standard deviation.

4.3 Loss

Due to its self-adjusting behaviour (Jacobson and Karels, 1988), TCP performance is critically affected by loss. Nevertheless, previous studies (Paxson, 1997a) have shown that packet loss is low, at least for the analysed mesh of Internet paths. One of the purposes of this paper is to produce a similar study, but based only on traces collected from a single point and with no control over the senders. It may be argued that the survey carried out as part of this study was somehow limited, as the wget client does not support HTTP1.1. As a result, the objects from a page are downloaded in separate connections, which leads to smaller congestion windows. Further, the resulting figures for loss may be lower than the ones obtained for a long-lived connection, with larger congestion windows. The losses were split into visible retransmissions and inferred retransmissions. The first category, visible retransmissions, is represented by losses which are indicated by anomalies in the TCP segments sequence. The second category, inferred retransmissions, includes the losses that are not apparent from the sequence of succesive TCP segments (more details on this subject are given in (Ghita et al, 2001)). The second category was named *inferred* because the process of identifying a loss is not based on sequencing, but only on packet spacing. The technique is reliable for a simple HTTP 1.0 retrieval, where the reply is a single object. Additional problems arise if HTTP 1.1 is used, due to spacing introduced between retrievals of successive objects within the same connection. In this case, the method requires comprehensive information from the application layer; since this is currently under analysis, it is reserved for future work.

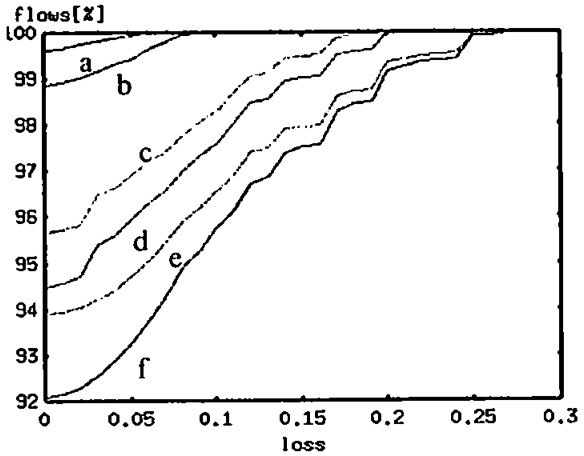


Figure 3 - Packet loss distribution for a) / b) visible retransmissions 2002 / 2001, c) / d) inferred retransmissions 2002 / 2001, e) / f) all losses 2002 / 2001

The distributions for both types of losses, as well as their sum, are displayed in Figure 3. The average figures for loss were: 0.18 / 0.83 / 1.1 %. (visible / inferred / total) for the 2001 experiments and 0.16 / 0.47 / 0.63 % (visible / inferred / total) for the 2002 trace. It is noticeable that the inferred losses accounted for the vast majority of the total loss, which may be caused by the short-lived character of most connections. This may be due to the small number of packets / connection which, again, leads to low values for congestion window and, implicitly, too few acknowledgements returned for triggering a retransmission when a loss happens.

The short-lived connections have an additional undesired effect: the accuracy of the measurement cannot go beyond the granularity of the download due to the low number of packets exchanged. For example, having a transfer consisting of 10 packets, the minimum detectable loss is 0.1, a situation also described in (Paxson, 1997a). To reduce this error granularity, we calculated the loss based on the total number of packets. The year 2001 tests had a total of 137297 packets, with 295 visible and 1033 inferred retransmissions, producing the overall packet loss figures 0.21 / 0.75 / 0.96 % (visible / inferred / total). For the 2002 tests, a number of 297 packets were visible retransmissions and 604 packets were inferred retransmissions; comparing this with the total of 129404 packets, results in an overall packet loss of 0.22 / 0.46 / 0.68 % (visible / inferred / total).

4.4 Bandwidth

An estimate of the total bandwidth was produced for each connection. The estimate used delay between pairs of consecutive packets inferred to be sent in a back-to-back manner, and it was based on the method proposed by Keshav in (Keshav, 1991). The problems that may occur due to clock granularity were avoided by using a microsecond kernel timer, the Kansas University Real-Time Linux (KURT) (Niehaus, 2002). The obtained figure might be affected by the problems associated with packet-pair bandwidth inference but, due to the unknown behaviour of the senders, it was not possible to apply the Receiver Based Packet Pair as described in (Paxson, 1997a) to avoid these problems.

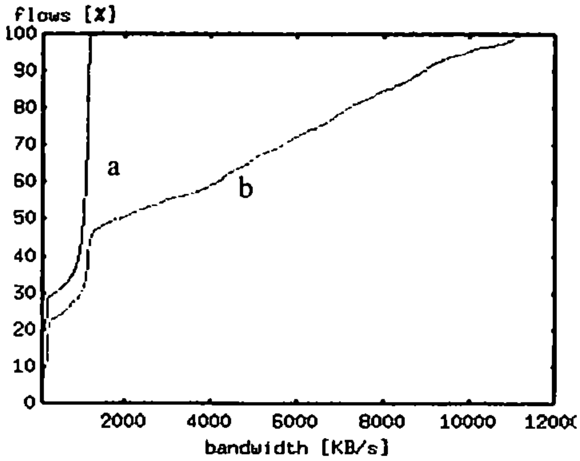


Figure 4 - Bandwidth cumulative distribution for a) autumn 2001 and b) spring 2002

From all network characteristics, the network upgrade mentioned earlier affected bandwidth the most. It can be noticed in the distribution from Figure 4 that bandwidth reached a maximum of approximately 1.2MB/s for the autumn 2001 round of experiments. This matches, in fact, the configuration of the network: at the time of the experiment, the connectivity of the desktops was 10Mb LAN. For the spring 2002, the maximum figure is 12MB/s, which reflects the tenfold increase in desktop bandwidth.

4.5 Congestion window analysis

The congestion window inference includes a high level of assumption in terms of TCP connection analysis. In our case, the task had an increased level of difficulty due to the characteristics of the monitored transfers: unknown senders, receiver-based capture, and no control over the endpoints / transfer. The fact that the senders use an unknown TCP implementation does not allow any inference in regards to profiling of the congestion window evolution. The intention was to produce a rough estimate of the congestion window, not to compete with *tcpanaly* (Paxson, 1997b), which includes more complex analysis but also requires traffic capture at / near the endpoints. The receiver-based capture brings with it uncertainty in regards to if, when, and as a response to what acknowledgement, the sender transmitted a data segment. Due to the variety of window increase policies and the uncertainty of which acknowledgements reached the server, the congestion window inference was based exclusively on timing between different trains of packets rather than acknowledgement dialogue. The actual method focused on isolating groups of packets that appear to be transmitted as part of the same round, based on the distance between successive in-sequence packets. The third problem, no control over the endpoints, differentiates the study from Internet measurement efforts (Paxson, 1999), (NIMI, 2002). Within measurement infrastructures, endpoints running dedicated clients transfer large files between them at regular intervals in order to determine the network characteristics. Within this study, all the senders were remote sites on the Internet and the objects transferred were various web pages residing on the servers; as a result, there was no control over the size / timing of the connections.

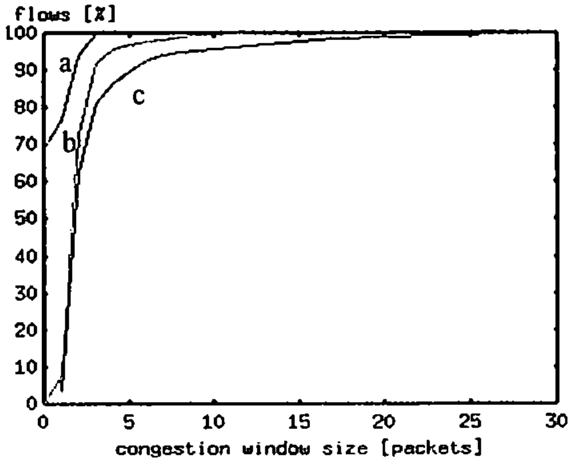


Figure 5 - Cumulative distribution of the a) initial, b) average, and c) maximum congestion window size

The resulting distribution is displayed in Figure 5. The average figures for the three variables (initial / average / maximum congestion window) were 1.91 / 3.47 / 4.95 for 2001 experiments and 1.77 / 3.16 / 4.52 for the 2002 experiments round. The difference between the figures can be attributed, again, to the network upgrade that reduced the packet loss and delay figures, as mentioned before.

5 The NRG network traces

5.6 Page content analysis

When the first round of experiments was run, the latest version at the time did not allow for a full download of the web pages (e.g. for a page with 4 images, only the HTML file was retrieved). At the second round of experiments, the newer version of wget had the facility to parse web pages and download the objects hosted on the same server with the page), which allowed a rough estimation of the actual content of the page. In the case of a HTTP1.1 client, these objects would be downloaded in a single connection. This gives an approximate indication of the actual length, in terms of size, of a connection.

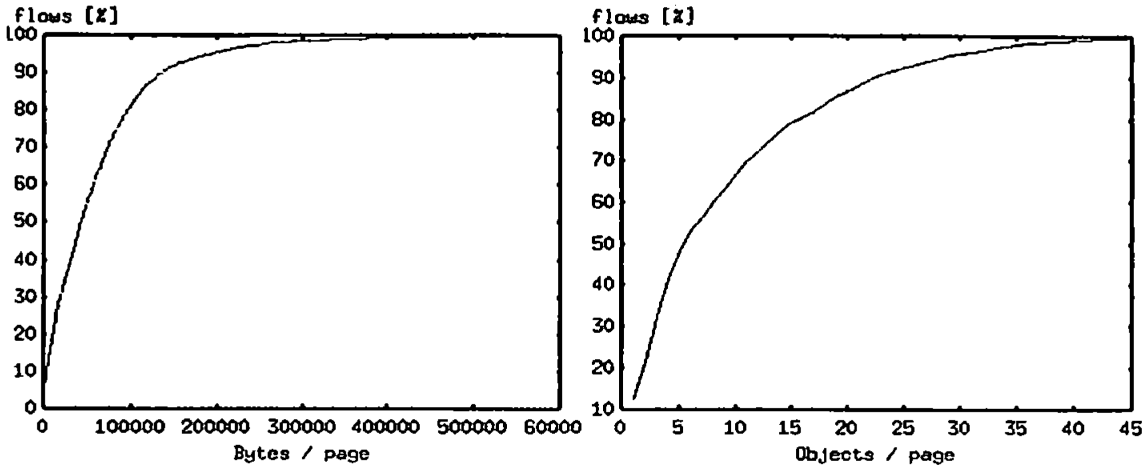


Figure 6 - Distribution of page content in bytes / page and objects / page

Figure 6 shows that most web pages have relatively large size (for some of his experiments, Paxson considered 100 KB files to be satisfactory for evaluating the properties of Internet). Also, from the distribution of objects per page, it may be concluded that full usage of HTTP 1.1 request pipelining would considerably reduce the overall time to retrieve the web page. The average figures for Figure 6 are 72607 bytes / page and 10.5 objects / page.

5.7 Connection analysis

Although a convenient and comprehensive tool, even the latest version of wget does not include some major functionality such as supporting frames and request pipelining (according to the author, there are no plans to expand it in the future in these areas). The set of traces captured from the traffic produced by the NRG members was therefore used for the connection analysis. The machines in the NRG were running either on a flavour of Linux (RedHat or SuSE) or Windows (NT4 or 2000 Professional), with Netscape Navigator 4.76-4.77 or Internet Explorer 5.0-5.5 as correspondent web clients. All the mentioned versions have HTTP1.1 enabled as standard, therefore they all should pipeline the requests whenever possible. The analysis of the captured traffic focused on the connection length, in order to determine the average length of an HTTP retrieval for the real traffic case. It may be argued that the amount of users involved in the study was relatively small; however, for future, it is aimed to compare these figures with results obtained from bigger, backbone collected traces. The result of the connection size analysis is displayed in Figure 7. It was observed that approximately three quarters of the flows had a download size of under 5KB with average numbers of 6220 bytes / connection and 7.12 packets / connection. These are very low figures, considering the previously estimated average of 72607 bytes / page obtained from the Random Yahoo Link experiments, and indicate that, in spite of the rich content of the Internet, the HTTP pipelining capabilities are not efficiently used.

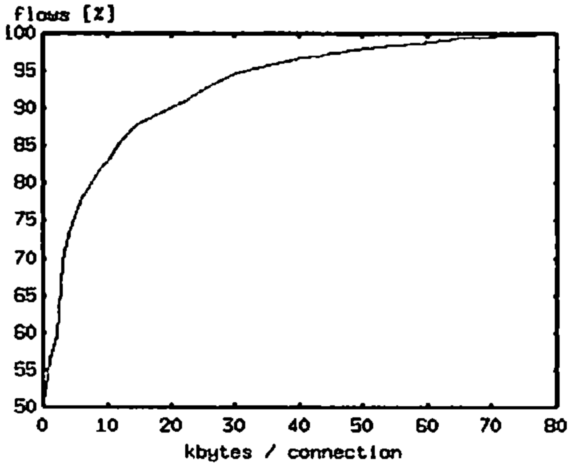


Figure 7 - Cumulative distribution of connection size

6 Conclusions

This paper presented the findings from an endpoint-based network per-flow trace analysis. In spite of its limited scope, the proposed analysis allowed characterisation of a fairly large number of network paths. The traffic was studied with a two-fold purpose: to evaluate the network conditions experienced by the flows and to determine the characteristics of a web page in terms of total content and number of elements per page. For the analysis, artificial traffic was mainly used. The experiment was carried out in two rounds: autumn 2001 and spring 2002. It used a random page generator combined with a command line HTTP retrieval tool, which was preferred instead of the real traffic due to the complexity in interpreting HTTP 1.1 pipelined transfers. Nevertheless, in order to evaluate the size characteristics of real transfers, a pilot traffic capture in a limited environment was performed.

The TCP analysis revealed a loss-free image of the Internet, with an average loss of 1.1% for the first round of experiments and 0.63% for the second round. The overall figures indicated even a smaller loss probability, of 0.96% and 0.96% respectively. The round trip delay values were also fairly low, with an average 200.5 ms for autumn 2001 and 196 ms for spring 2002 experiments and a standard deviation of 22.3 ms and 7.8 ms (4.7 % of the RTT averages) respectively. The page content analysis revealed that the average page size is approximately 70 KB, with an average of 10 objects / page that fully justify usage of HTTP 1.1 pipelining. However, the real network traffic showed much lower figures, of only 6220 bytes / connection and 7.12 packets / connection, which indicates that either the HTTP 1.1 pipelining mechanisms are inefficiently used or that current web pages are not suitable for pipelined requests.

For future work, it is primarily aimed to reduce the uncertainty of packet loss estimation and to expand the analysis towards connecting the TCP analysis with the HTTP retrieval, in order to be able to isolate individual retrievals of objects. Also, if possible, the per-flow analysis will be applied to larger traces to determine whether or not these findings are scalable and may be applied to traffic collected from core internet links.

7 References

AMP, The Active Measurement Project (AMP) homepage, http://moat.nlanr.net/AMP/, 2002

Ghita, B., Lines, B, Furnell, S., Ifeachor, E., "Non-intrusive IP Network Performance Monitoring for TCP flows", *Proceedings of IEEE ICT 2001*, 2001

Jacobson, V., Karels, M., 'Congestion Avoidance and Control', *Proceedings of SIGCOMM '88*, 1988

Keshav, S., "A Control-Theoretic Approach to Flow Control," *Proceedings of SIGCOMM '91*, pp. 3-15, 1991.

NIMI, The National Internet Measurement Infrastructure homepage, http://ncne.nlanr.net/nimi/, 2002

Niehaus, D., "KURT-Linux: Kansas University Real-Time Linux", http://www.ittc.ku.edu/kurt/, 2002

Osterman S., "tcptrace homepage", http://www.tcptrace.org, 2002

Paxson, V., "Measurements and Analysis of End-to-End Internet Dynamics", PhD thesis, 1997

Paxson, V., "Automated Packet Trace Analysis of TCP Implementations", *Proceedings of SIGCOMM '97*, 1997

Paxson V., "End-to-end internet packet dynamics", *IEEE/ACM Transactions on Networking*, vol 7, no 3, 1999

Random Yahoo Link (RYL), Random Yahoo Link Page, http://random.yahoo.com/bin/ryl, 2002

tcpdump, tcpdump public repository, http://www.tcpdump.org/, 2002

Thompson K., Miller G.J. 'Wide-Area Internet Traffic Patterns and Characteristics', *IEEE network*, nov 1997

Wget, GNU Wget home page, http://www.gnu.org/software/wget/, 2002

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.

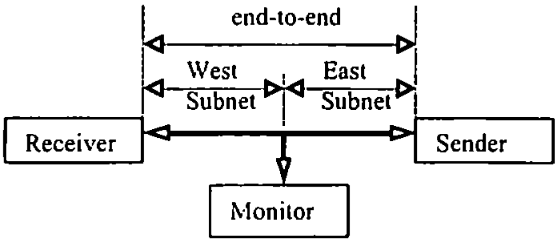


Figure 1 - Measurement for a sender-receiver configuration

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.

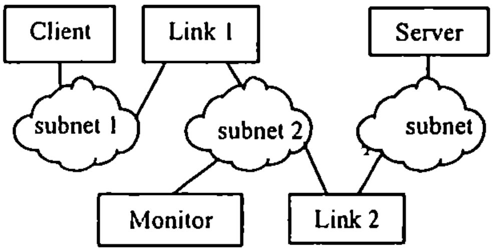


Figure 2 Test configuration

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:

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

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

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

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 →A) = 400 ms;
- R1(A →B) = 300 ms;
- R2(B →A) = 200 ms;
- R2(A →B) = 100 ms.

The measurable RTT value is:

$$RTT = R1_{B \to A} + R2_{A \to B} + R \;\; = 710ms \qquad (3)$$

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→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

Network Quality of Service Monitoring for IP Telephony

B.V.Ghita[1], S.M.Furnell[1], B.M.Lines[1], D.Le-Foll[2], E.C.Ifeachor[3]

[1] Network Research Group, School of Electronic, Communication & Electrical Engineering, University of Plymouth, Plymouth, United Kingdom
[2] Wavetek Wandel Goltermann, Plymouth, United Kingdom
[3] SMART Systems Research Group, School of Electronic, Communication & Electrical Engineering, University of Plymouth, Plymouth, United Kingdom

Abstract

This paper presents a non-intrusive method of determining network performance parameters for voice packet flows within a VoIP (Voice over IP, or Internet Telephony) call. An advantage of the method is that it allows not only end-to-end performance monitoring of flows, but also makes it possible to inspect the transport parameters a specific network or link when delay sensitive traffic transits through it. The results of a preliminary test, to check the validity of the method, are also included.

Keywords

Voice over IP, Quality of Service parameters, non-intrusive monitoring.

Introduction

Over the last two decades, the Internet has evolved from a few interconnected networks that linked research laboratories, universities, or military infrastructure, to an everyday tool which is easy to access and use by many people. The dramatic evolution can be assessed in terms of growth in the number of hosts and Internet applications. The initial use of the Internet was different to that of today. Contrasting two studies of Internet activity, from 1991 (Caceres et al, 1991) and 1997 (Thompson et al, 1997), it can be seen that the nature of activity has changed from applications such as telnet or file transfer to become dominated by web browsing (75%). The increased computational power of end-user stations has allowed new types of applications to be implemented. In addition, the speed and reliability of the Internet itself has been substantially enhanced due to the new technologies used. These advances have allowed application content to move from text to multimedia and real-time.

A major challenge in Internet development is how to support real-time applications, typified by Internet Telephony, within the existing structure. Internet Telephony aims to replace the traditional concept in telecommunications from data over voice to voice over data. The method for achieving this is to use the Internet as a transport carrier for voice, instead of the PSTN (Public Switched Telephone Network). The most obvious advantage is the low cost for long-distance phone calls.

An important barrier in the development of VoIP is the Internet Protocol (IP). IP works as a best-effort connectionless protocol. It was designed for data files that can tolerate delays, dropped packets and retransmissions; there are no guarantees about the delivery time or the reliability of a packet being transferred over the Internet. The most important aspects, when considering an audio conference are exactly those that Internet cannot guarantee: time and bandwidth. The quality of the resulting conference depends upon the satisfaction of these requirements. Within this context, the concept of Quality of Service (QoS) was introduced. Although the Internet represents an environment in which the QoS cannot be guaranteed, there are measurable parameters for a specific service, as presented in a QoS overview study (Stiller, 1995).

This paper presents an offline method of determining network performance parameters for voice packet flows within a VoIP call. An advantage of the method is that it allows not only end-to-end performance monitoring of the flows, but also makes it possible to inspect the behaviour of the network when faced with delay sensitive traffic.

QoS concept for VoIP and current state of monitoring

The QoS is the overall rating for a service. Measurement of QoS essentially includes measuring a number of application dependent parameters and then gathering them in a weighted sum. If we consider QoS for VoIP, the object of the analysis is the voice at the receiving end, with its two main characteristics, sound and interactivity. There are two main sources of impairments for the voice heard by the receiver. The first is the codec, which compresses the speech flow in order to send it over the network at a lower bandwidth than original. Aside from the positive result in terms of bandwidth utilisation, this process degrades the quality of the speech. The second source of impairment is the transport. After encoding, the audio flow is packetised and sent over the Internet. However, because of the Internet's structure, the arrival of the packets at destination cannot be guaranteed. The paper is focused upon a consideration of this latter impairment.

Building a list of performance parameters for a service should start by identifying the application that requires that specific service. For example, if the targeted application is a file transfer then the delay or jitter parameters are almost irrelevant when compared to throughput or packet loss. In a similar manner, for a real-time application, delay is far more important than the other parameters. The paper does not intend to prescribe a specific weighting here, but it is good to bear in mind their priorities when assessing the overall performance.

When considering QoS for VoIP applications, a network-related view of the performance should include the following parameters:

- delay - the time elapsed between the sending of a packet and its arrival at the destination;
- jitter - the variance of the delay value;
- packet loss - the number of lost packets, reported in the time elapsed;
- throughput - the amount of data transferred from one place to another or processed in a specified amount of time.

There are several suggested methods that can improve or guarantee the QoS for transport, such as DiffServ (Differentiated Services) (Nichols et al, 1998), Tenet (Ferrari et al, 1994), or QoS Routing combined with RSVP (Reservation Protocol) (Crawley et al, 1998). Unfortunately, none of them are applied on global basis because of the scale and complexity of the Internet. Therefore, it is vital to determine in such an environment whether or not a specific connection meets the requirements of a VoIP call.

Transport QoS has two main areas: end-to-end measurements and, in case there are changes in the level of parameters, fault localisation. An example is given in Figure 1 which shows, for an arbitrary division of the entire route of the packets, the end-to-end parameters, and two sets of parameters, 'East' and 'West'. The latter can be used to localise a fault in either 'East' or 'West' sub-network, by comparison with the end-to-end parameters.

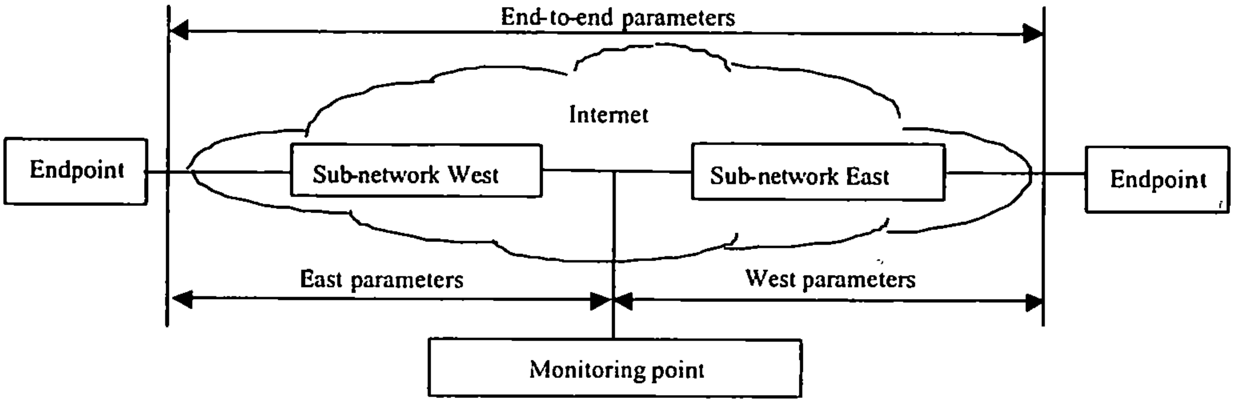


Figure 1: The Performance parameters for a general example of monitoring

In a traditional approach, the two aims would require a 3-tool configuration. For end-to-end measurements, testing clients should be put at both ends and, for fault location, a testing server should be placed at the monitoring point. After that, traffic should be collected by the end stations, then sent to the server, in order to be analysed and compared with the data collected by it. There are two main disadvantages with this approach:

- it is intrusive; in the best case, even if the endpoint clients are just monitoring, they have to send the data to the server in order to be analysed;
- it requires placement of monitoring devices at both ends.

The QoS for transport can be determined from the audio flows within a call (which run on RTP, Real Time Protocol). Current tools (e.g. Hammer VoIP Analysis System, HP Internet Advisor, rtpmon (Bacher and Swan, 1996)) base their calculations upon parsing both the RTP and/or the accompanying control flows (running on RTCP, Real Time Control Protocol) and displaying the available data. The main disadvantage is that none of these tools can establish fault location without using the traditional approach mentioned above. More than that, they do not build any relation between the end-to-end parameters, obtained from the RTCP flows, and the end-to-monitoring-point parameters, obtained from the RTP monitoring.

Considering these limitations, we aim to obtain a better view of the network performance, without using several devices and without injecting additional traffic into the network. This paper presents a non-intrusive method of determining the transport performance parameters for the real-time traffic within a VoIP call, using a single point of monitoring. The proposed method can reveal both the end-to-end performance and the fault localisation, if the monitored parameters change their value along the route, and also avoids both of the disadvantages identified.

Description of H.323 calls

VoIP is a relatively new concept and, therefore, most of the work performed in this area is still at a developmental stage. From the large range of standards for VoIP, the H.323 protocol stack (ITU, 1998), developed by ITU, was selected as the basis for the work presented in this paper.

The focus of the QoS for transport is, as mentioned, on the audio flows. Because of the H.323 call structure, which will be detailed below, these flows cannot be identified unless the entire call is monitored. The information exchanged in a H.323 conference is classified in streams, as follows: audio (coded speech), video (coded motion video), data (computer files), communication control (control data), and call control (signalling data).

We will consider the simplest case - a direct connection between two computer terminals, similar to a classic phone call. The call begins with a call signalling phase – signalling messages (Q.931 using H.225 specification) are exchanged, on specific ports. At the end of this phase, the call is established and a call control channel is opened, on ports dynamically allocated. The control channel then provides for various functions: capabilities exchange, logical channel signalling, mode preferences, master – slave determination. After the terminals decide which of them will act as a master for the call (in order to easily resolve conflicts), they exchange their capabilities and open an audio channel, using logical channel signalling. The logical channel is also opened on a dynamically allocated port, decided within the control messages. The audio flows run on the opened logical channel. When one of the users wants to terminate the call, the logical channel is closed, using call control, then specific call signalling messages are exchanged, and the call is closed.

The audio (as well as video) flows within an H.323 conference are transported using RTP, as it provides end-to-end network transport functions suitable for applications transmitting real-time data over multicast or unicast network services (Schulzrinne et al, 1996). It does not address resource reservation and does not guarantee quality-of-service for real-time services. In fact, the whole protocol is conceived not as a separate layer, but as a framework, to be integrated within other applications. RTP is usually run on top of UDP (User Datagram Protocol), an unreliable transport protocol. TCP (Transport Control Protocol), although reliable, brings additional delay problems, by delivering the packets in order and recovering the lost packets, and, therefore, is not recommended for carrying real-time flows.

RTCP is the control protocol for RTP. One of its functions is to provide information about the packets loss and inter-arrival jitter for the accompanying RTP flow. The information is provided periodically by all the senders / receivers within a conference using specific packets, and is based on the RTP flow measurements. The RTCP flow also runs on UDP.

Experimental method and implementation

Monitoring procedure

The monitoring procedure comprises three steps. First, the voice flows (RTP) are identified and then captured using one of the capture programs. In the monitoring phase, the RTP header fields and the RTCP packets are used to determine the performance parameters. Then correlation of RTP and RTCP is used to establish the location of the problem area. The stages are described in more detail in the following paragraphs.

Identification of the audio flows

The analysis is targeted on the audio streams. The ports on which the audio streams run can be determined only by capturing the connection establishment phase, then parsing the setup and control messages, which contain the audio stream ports as parameters. The parsing process is not straightforward, as the content of the setup and control messages is not header-like (using fields), but encoded using ASN.1 syntax.

Parameter measurement using RTP monitoring and RTCP parsing

The header fields of RTP packets are used as input to the analysis, together with the timestamp of the packet arrival, given by the capture program. The structure of the RTP header, as described in (Schulzrinne et al, 1996), is shown in figure 2.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="2">V=2</td><td>P</td><td>X</td><td colspan="4">CC</td><td>M</td><td colspan="7">PT</td><td colspan="16">sequence number</td></tr>
<tr><td colspan="32">timestamp</td></tr>
<tr><td colspan="32">synchronisation source identifier (SSRC)</td></tr>
<tr><td colspan="32">contributing source identifiers (CSRC)
...</td></tr>
</table>

Figure 2 The RTP packet format

The description of the fields is as follows:

- V – version of RTP (currently used is 2)
- P – padding, for indicating the existence of padding octets (last octet of padding indicates how many octets should be ignored)
- X – extension (there is a header extension after the fixed header)
- CC – number of CSRC identifiers that follow
- sequence number – is incremented by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence
- timestamp – reflects the sampling instant of the first octet in the RTP data packet
- SSRC – synchronisation source; the source of a stream of RTP packets, in order to make the sources independent upon the network address.

- CSRC – Contributing Sources; source of a stream of RTP packets that has contributed to the combined stream produced by an RTP mixer

Note: to the existing RTP data packet header can be added a RTP header extension.

Although the RTP packet has the timestamp field, this is less used in the analysis; it is an integer, and it is measured in sampling units (depending on the codec used). It is put by the sender and used by the receiver as a reference for the stream playing. The time analysis performed is based on the timestamp of the packet, put by the capturing device, at the monitor.

The following types of parameters can be determined using the RTP header fields and the arrival timestamp of each packet, taken from the packet capture program:

a. delay-related parameters:

- inter-arrival delay – by subtracting the capture timestamps of successive packets
- inter-arrival jitter – by comparing the previous delay with the current one
- one-way delay jitter – by comparing the inter-arrival delay with the sender delay (the interval between sending two sequential packets).

b. packet-accounting parameters

- lost packets and out of order packets – by comparing the expected sequence number with the sequence number of the incoming packet. The lost packets variable is increased, but the presumed lost packets sequence numbers are memorised, in case the packets were not lost, but only misordered.

c. flow speed parameters

- throughput – determined by dividing the actual received number of bytes by the time of the connection

The RTCP packets can be used as an instrument for end-to-end measurements. Their fields provide the values for inter-arrival jitter and lost packets; their structure is also defined in (Schulzrinne et al, 1996), but the header is structured, and too complex to be detailed within this article. RTCP flows perform the following functions:

- to provide feedback on the quality of the data distribution
- to help the receivers to associate and to synchronise multiple data streams from a given participant
- to allow each participant to keep track of all the other participants in the conference
- to convey minimal session control information

The RTCP reports are a very convenient tool for monitoring and they are, as mentioned, currently used in the available products. Nevertheless, the following observations can be made in relation to using RTCP to analyse the flows:

- it runs on UDP and, therefore, it is possible that a number of packets will not arrive, so no data will be available for that period of time.
- it has scalability problems (Rosenberg and Schulzrinne, 1998). The RTCP messages are limited to 5% of the whole traffic. In the case of a many-to-many conference, on

normal behaviour, there would be a low number of RTP messages per-terminal (in order to maintain the 5% limit) (Schulzrinne et al, 1996).

- it returns only end-to-end parameters and, therefore, cannot locate the cause of parameter changes (this problem exists regardless of the conference characteristics)

Note: the analysis is performed on a 'per-flow' basis. Prior to performing the analysis, the incoming packets (from several audio channels) are split into flows (each flow representing a channel). When saying successive packets, we refer to packets belonging to the same flow.

Correlating RTP analysis with RTCP content

By correlating the two sets of parameters, obtained from RTP and RTCP, it is possible to determine whether or not a specific problem (e.g. a high number of lost packets) is caused by a problem which exists in the East sub-network or the West sub-network. Figure 3 presents the captured flows.

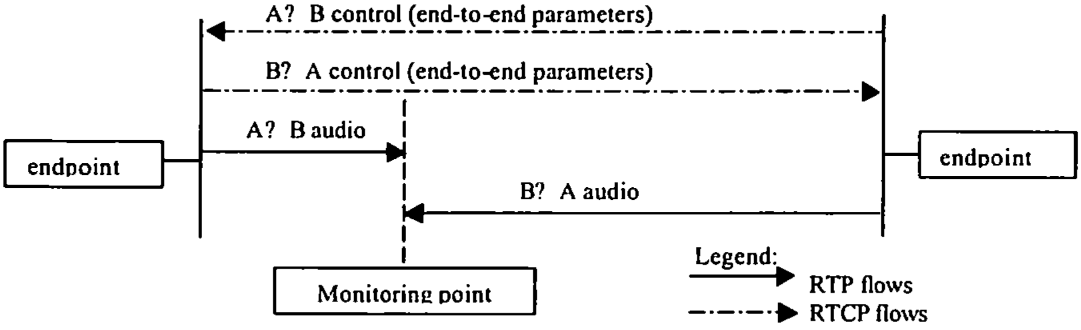


Figure 3: RTP and RTCP flows monitoring

The RTP streams, as captured on the monitoring point, are: A? B (after passing through the West sub-network) and B? A (after passing through the East sub-network). Therefore, by measuring the parameters of these flows, we can determine the performance of the West sub-network (from the A? B flow) and the East sub-network (from the B? A flow).

We have to bear in mind that the A? B direction does not fully characterise the behaviour of the network, as it can be very good for one direction and bad for the other (it does not have to be symmetrical in terms of performance). Meanwhile, as mentioned, RTCP provides the end-to-end parameters, i.e. the performance of the entire A? B and B? A routes, but it has no indication about how these parameters change on the route (i.e. cannot establish where a faulty behaviour of the network determined a change in the values of the parameters).

Putting together the two sets, we obtain parameters for the following segments:

- A? B and B? A, end-to-end – from the RTCP flows
- A? monitoring point and B? monitoring point – from the RTP flows
- monitoring point? B and monitoring point? A – by subtracting the RTP obtained values from RTCP end-to-end parameters.

Therefore, by using both RTP and RTCP, we obtain both the end-to-end and the end-to-monitoring point parameters for the monitored flows.

Implementation

In the first instance, the tcptrace program (Ostermann, 2000) was used within the monitoring module. Tcptrace is an offline analysis program, which uses tcpdump traces as input. Although the program had limited support for UDP (it was able to separate the UDP flows), and no support for RTP, it was considered a useful tool because of its per-flow analysis capabilities. The module was subsequently migrated to ipgrab (Borella, 2000) to reduce the complexity of the program (tcptrace includes a lot of functions, spread over various modules, most of them related with TCP analysis). Most of the analysis (e.g. the distributions), as described in the following section, was performed offline, under Microsoft Excel. As no equipment to simulate several calls was available, the analysis was performed for only a single VoIP call. The module will work for more than one call, but a proper filtration of the output should be added. In addition, the refresh period of the analysis (i.e. each packet) could create computational problems for a high number of flows. A proper solution would be to display the parameters at certain intervals (e.g. every second).

Special attention is given to the marker, payload type and timestamp fields within the RTP header. During a VoIP call, if there is no speech from the user, an endpoint does not send RTP packets. Therefore, when calculating the flow speed and the delay parameters, the silence periods should be ignored. The silence periods can be identified using the marker field: an RTP packet with the marker field set signals the end of a silence period. Also, if the payload characteristics are known (e.g. each RTP packet contains a 30ms frame), the delay between successive packets at the sender can be determined. Thus, by subtracting this value from the inter-arrival delay, we obtain the one-way delay jitter.

Validation

Experimental testbed configuration

A network testbed was constructed in order to validate the proposed method. Figure 4 presents the testbed configuration, which included two networks, connected through a faulty link. The monitoring point is placed on the route, at the exit point (after the router) of one of the networks.

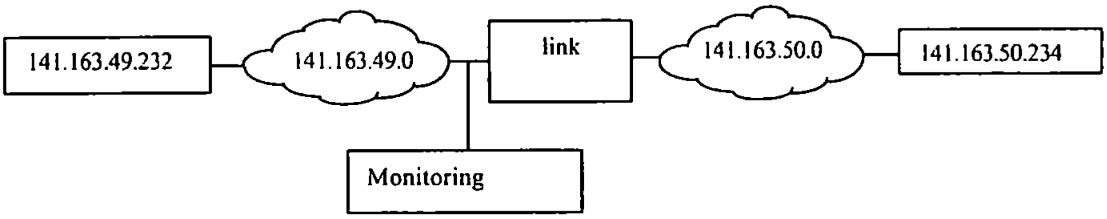


Figure 4: Network testbed configuration

The link is emulated using the NISTNet program (NISTNet, 2000). NISTNet emulates various network problems by forwarding packets, under specific parameters like packet loss, delay or jitter, between two network interface cards, on a Linux station. For our test, we used the following parameters (symmetric for the two directions): 5% packet loss, 300 ms delay, 25 ms jitter, unlimited bandwidth, normal distribution. The measurements were based on a capture session; number of packets captured: ~20000 (some of them were removed in order to eliminate the transitional behaviour).

The software tools used for generating, capturing and monitoring the VoIP flows were:

- NetMeeting (WinNT) – to establish and run a H.323 VoIP call;
- codec: Microsoft G.723.1, 6400 bits/second, continuous speech;
- tcpdump, ipgrab (Linux) – to capture packets transmitted over the network (between the two VoIP endpoints);
- the analysis module (Linux) – first developed within tcptrace, then transferred to ipgrab, to allow online capturing.

The measurements aim to locate the jitter and the packet loss by dividing the route of the packets, as presented in Figure 3 into sub-network East (network 141.163.49.0), and sub-network West (emulated link and network 141.163.50.0). After obtaining the various parameters, we will try to identify the fault location on the 141.163.50.0 network and link side of the route. In the following paragraphs, we will refer at 141.163.49.232 station as A and at 141.163.50.234 as B.

Results and value comparison

Table 1 presents the following information:

- normal – the normal behaviour, on a network without any loss;
- RTP results – the values determined from the RTP monitoring;
- RTCP results – the values determined from the RTCP parsing.

Parameter	normal	RTP results		RTCP results	
		A? B	B? A	A? B	B? A
throughput [bytes/sec]	800	800	760	760	760
packet loss [%]	0	0	5	5	5

Table 1: Throughput and packet loss statistics

A. Throughput and packet loss

The RTCP throughput is determined from the RTCP sender reports, using the 'sender octet count' which indicates how many octets were transmitted since the beginning of the call. The RTCP reports also, include report blocks, which give the performance parameters of the senders 'heard' by the emitter of the report. The RTCP packet loss is determined from these report blocks, using the 'cumulative number of packets lost' field.

It can be noticed that the B? A values differs, which indicates a 5% packet loss on that direction, located in the right side of the route. Also, the A? B values indicate that there is no alteration, in term of packet loss, in the left side of the route (the 141.163.49.0 network).

B. Jitter

From the RTP monitoring, the jitter was determined by subtracting the average interarrival delay from the interarrival delay for the current packet. The results are presented in Figure 5.

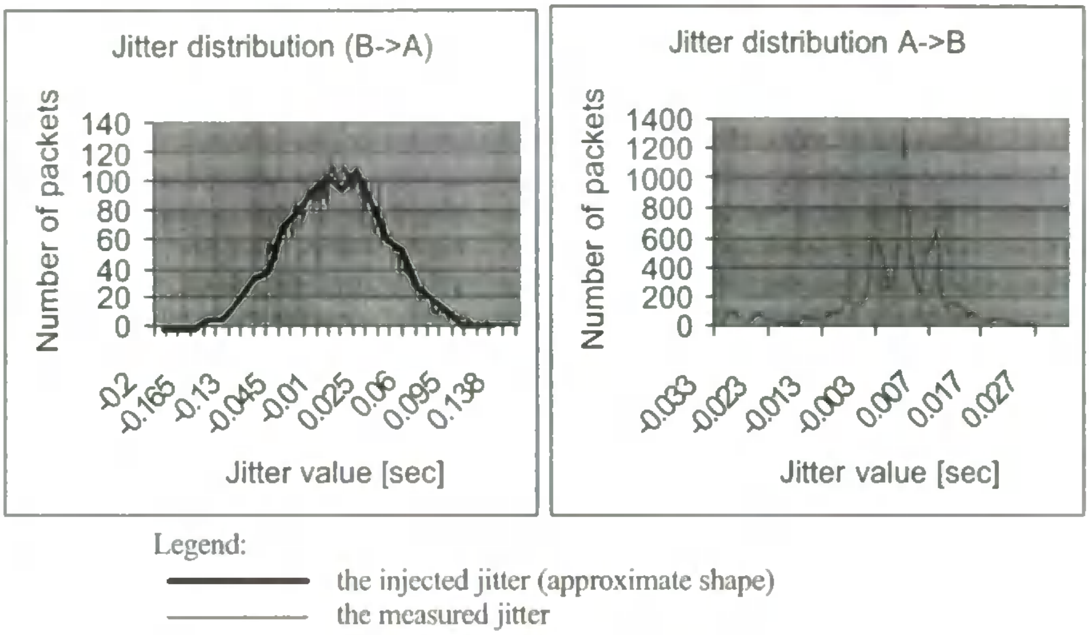


Figure 5: RTP jitter distribution (from RTP monitoring)

Note: In the left graph, the thick line indicates the shape of the average distribution (based on separate measurements on same environment). It can be seen that the measurements are valid.

In the RTCP parsing, the values were extracted from the RTCP report blocks (the 'interarrival jitter' value). The results are presented in Figure 6.

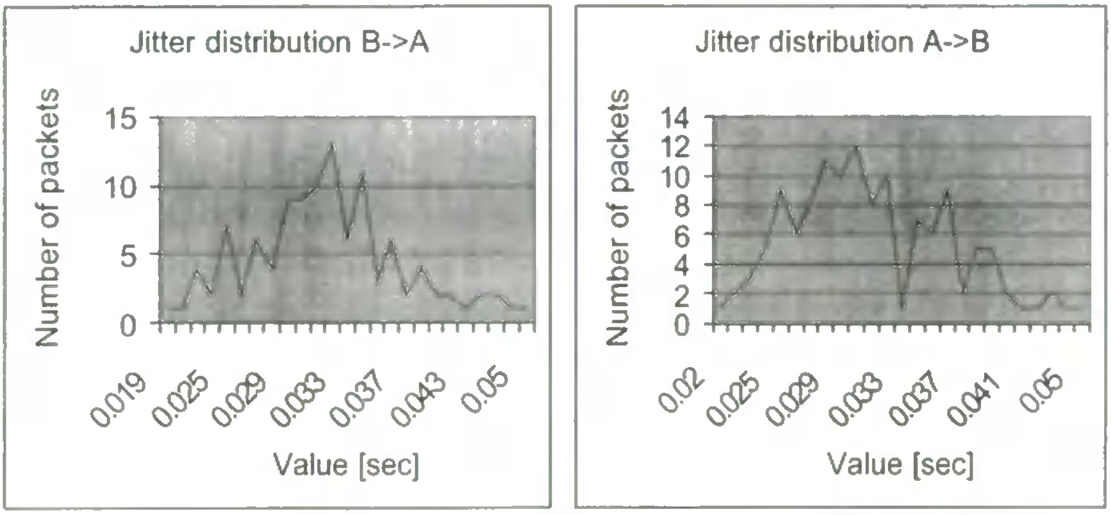


Figure 6: RTP jitter distribution (from RTCP parsing)

As can be seen from Figure 5, the distribution for the B? A flow can be approximated with a Normal (Gaussian) one (the interval (-inf;-0.6) could not be reproduced because of some measurement limitations), while the A? B flow shows no distribution of the jitter. For both of the flows, there is an additional 3 ms jitter, caused by NetMeeting behaviour: although the packets inter-arrival delay should be constant (60 ms), from time to time, the program transmits a voice packet after 30 ms. The measurement is more accurate for packet loss han jitter because of the errors in the measurement of jitter, as well as because the out-of-order packets were not considered in the analysis.

If we consider the absolute values for the jitter, it results an average value of 28 ms, which, if we extract he 3 ms caused by NetMeeting behaviour, it results the value of the emulated link: 25 ms. As a conclusion, the tool, together with the results analysis, identified the 5% loss and 25 ms jitter generated by the right side of the monitored route.

Although the monitoring tool was built, and these preliminary tests were performed, a full assessment requires further analysis in a real or simulated VoIP environment. Such an environment would include several simultaneous conferences, running between endpoints situated at different locations, over various routes.

Conclusions and further work

This paper has described an off-line method to measure the QoS transport parameters for a H.323 VoIP call from a single point, by non-intrusive monitoring, and we presented a test performed in order to validate our method. The jitter and packet loss analysis seems promising, but further work is required to determine, monitor and analyse the other parameters. Also, a specific change in the performance parameters group can be related with a specific network event (e.g. a congested router). Therefore, analysis of the dynamics of the calculated parameters is required.

There are also other parameters still to be measured. In measurement systems for POTS (Plain Old Telephone Systems), a useful parameter for the call performance is the round trip time (RTT) delay (i.e. the time needed by a signal to go from one end to the other and then back). There is no direct possibility to determine such a parameter for H.323. calls because the standard is built for multicast conferences (multi-to-multi conferences), and so it does not include mechanisms for single end-to-end connection; the flows between the endpoints do not run in pairs, there is no correlation between them (they run independently). There are several methods to determine RTT for VoIP calls:

- Using the setup and control messages; they run on TCP, and the values obtained might differ from the (theoretical) ones for UDP
- Using RTCPs' 'delay since last source report(SR)' field.
- Correlating the RTP and RTCP flows. The RTCP packets include a 'extended highest sequence number received' field. If the value of this field is correlated with the sequence number of the sender, together with its timestamp, the RTT can be measured.

As future work, we aim to:

- refine the described method in order to cover all the possible situations; e.g.: due to method limitation, we were not able to identify correctly jitter higher than the inter-arrival time,
- determine a good estimate for RTT, based on the RTCP reports,
- advance the correlation of RTP and RTCP flows in order to narrow the region of fault location from East/West network down to a link or a sub-network,

- investigate, using intelligent analysis methods, if the traffic performance parameters at one moment can give an estimate for the future level of performance

The method presented, together with the additional objectives above, aims to achieve the perfect monitoring approach, which has to be single point, non-intrusive, measures all the performance parameters, fully locates the source of network degradation, and predicts the future behaviour of the network. By doing this, we can determine if the IP network offers, currently as well as in the future, to the IP telephony users the quality they require, and, if not, where the problem resides.

References

Bacher D. (1996), 'rtpmon: A Third-Party RTCP Monitor', ACM Multimedia '96.

Borella M. (2000), 'ipgrab homepage', http://home.xnet.com/~cathmike/MSB/Software/index.html.

Caceres R., Danzig P.B., Jamin S., and Mitzel D.J. (1991), 'Characteristics of Wide-Area TCP/IP Conversations', Proceedings of ACM SIGCOMM '91.

Ferrari D., Banerjea A., and Zhang H. (1994), 'Network Support for Multimedia – A Discussion of the Tenet Approach', Computer Networks and ISDN Systems, December 1994.

ITU. (1998), 'Packet based multimedia communication systems', H.323 ITU Recommendation, February 1998.

Nistnet. (2000), 'The NIST Net home page', http://snad.ncsl.nist.gov/itg/nistnet/index.html

Ostermann S. (2000), 'tcptrace homepage', http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html.

Schulzrinne H., Casner S., Frederick R., and Jacobson V. (1996), RFC 1889 - 'RTP – A Transport Protocol for Real-Time Applications', RFC depository, January 1996.

Crawley E., Nair R., Rajagopalan B., and Sandick H. (1998), RFC 2386 - 'A Framework for QoS-based Routing', RFC depository, August 1998.

Nichols K., Blake S., Baker F., and Black D. (1998), RFC 2474 - 'Differentiated Services Field', RFC depository, December 1998.

Rosenberg J. and Schulzrinne H. (1998), 'Timer Reconsideration for Enhanced RTP Scalability', Proceedings of IEEE Infocom 1998, March 29 - April 2 1998.

Stiller B. (1997), 'Quality of Service Issues in Networking Environments', internal report, http://www.cl.cam.ac.uk/ftp/papers/reports/TR380-bs201-qos_isues.ps.gz, September 1995.

Thompson K., Miller G.J., and Wilder R. (1997), 'Wide-Area Internet Traffic Patterns and Characteristics', IEEE network, November-December 1997.

ENTS - BREVETS

BREVALEX

Inventions · Marques · Designs · Dessins & Modèles · Copyright · Droits d'Auteur · Licensing · Licences

BREVATOME

DEMANDE DE BREVET FRANÇAIS

n° 03 50056
du 19/03/2003

pour : PROCEDE D'EVALUATION DE LA BANDE PASSANTE D'UNE LIAISON NUMERIQUE

au nom de : ACTERNA IPMS
SP22335/HM

3, Rue du Docteur Lancereaux - 75008 Paris
☎ 33 (0)1 53 83 94 00
📠 33 (0)1 45 63 83 33
www.brevalex.com
brevets.patents@brevalex.com
marques.trademarks@brevalex.com

Société de Conseils en Propriété Industrielle - Industrial Property Attorneys Firm
SARL au Capital de 186 000 Euros - RCS Paris 305332637 - TVA n° FR 54305332637

PROCEDE D'EVALUATION DE LA BANDE PASSANTE D'UNE LIAISON NUMERIQUE

DESCRIPTION

Domaine technique

L'invention se situe dans le domaine des télécommunications et concerne plus spécifiquement un procédé d'évaluation de la bande passante entre un premier et un deuxième point susceptible d'échanger des paquets de données via une liaison numérique dans un réseau de télécommunication comportant une pluralité de sous-réseaux.

L'invention concerne également un dispositif destiné à mettre en oeuvre le procédé.

L'invention trouve une application dans les réseaux de télécommunication tels que le réseau Internet.

ETAT DE LA TECHNIQUE ANTERIEURE

Une méthode connue pour évaluer la bande passante dans un réseau de télécommunication consiste à transmettre d'un premier point du réseau un fichier via le protocole FTP (pour File Transfer Protocol) comportant un marquage temporel et à mesurer la vitesse de réception de ce fichier par un deuxième point dudit réseau. L'émission d'un fichier de grande taille via le protocole FTP dans une liaison génère une surcharge du réseau. Par ailleurs, la charge générée par les utilisateurs du réseau au moment de la mesure étant inconnue, un transfert de fichier de petite taille via le protocole FTP ne garantit pas une utilisation optimale de la bande passante disponible. Tous ces facteurs contribuent à rendre aléatoire la mesure de la vitesse de réception des fichiers, et partant, la bande passante disponible au moment du transfert via le protocole FTP par le deuxième point du réseau.

Une autre méthode connue dans l'art antérieur consiste à mesurer le temps absolu de transmission d'un fichier de données entre les deux points du réseau dont le temps est mesuré en chaque point avec la plus grande précision possible. Cette méthode est certes plus précise mais présente un coût élevé dans la mesure où elle nécessite l'utilisation d'un système de mesure du temps de grande précision à chaque extrémité du réseau tel que délivrée par exemple par un système de transmission du type GPS (pour Global Positionning System).

Le but de l'invention est de pallier les inconvénients de l'art antérieur décrits ci-dessus au moyen d'une méthode et d'un dispositif simple, peu coûteux et susceptibles d'être utilisés entre n'importe quels points du réseau.

Un autre but de l'invention est d'isoler et de localiser sans ambiguïté un point de congestion lorsque les données échangées entre deux points d'un réseau transitent par plusieurs sous-réseaux.

EXPOSÉ DE L'INVENTION

L'invention préconise un procédé d'évaluation de la bande passante entre un premier point et un deuxième point comportant des terminaux susceptibles d'échanger des paquets de données numériques dans un réseau de télécommunication comportant une pluralité de sous-réseaux.

Le procédé selon l'invention comporte les étapes suivantes :

pour chaque sens de transmission à travers l'un au moins desdits sous-réseaux,

a. associer aux paquets émis quasi-simultanément un même identifiant,

b. horodater et enregistrer les paquets reçus,

c. identifier et trier les paquets reçus avec le même identifiant,

d. sélectionner le plus grand nombre entier possible m de groupes de paquets ayant le même identifiant,

e. mesurer les intervalles de temps séparant les instants de réception par le deuxième point des paquets des groupes sélectionnés,

f. calculer la bande passante en fonction du nombre de paquets des groupes sélectionnés et de la durée totale de transmission de ces paquets.

En identifiant des paquets émis quasi-simultanément dans le flux transmis du premier vers le deuxième point de la liaison, on se place dans les conditions réelles d'utilisation des usagers du réseau dans lesquelles l'estimation de la bande passante mesurée reflète l'encombrement réel de la liaison au moment de la mesure.

Dans un mode préféré de réalisation, la bande passante est calculée par l'expression suivante :

$$\overline{BW} = \frac{1}{m}\sum_{j=1}^{m}\left[\frac{1}{n_m}\sum_{i=1}^{n_m-1}\frac{l_{i,m}}{t_{(i+1),m}-t_{i,m}}\right]$$

où

- li,m représente la longueur du paquet de rang i du m$^{\text{ième}}$ groupe de paquets,
- ti représente le marquage temporel du paquet de rang i du m$^{\text{ième}}$ groupe de paquets,
- ti+1 représente le marquage temporel du paquet de rang i+1 du m$^{\text{ième}}$ groupe de paquets,
- nm représente le nombre de paquets du m$^{\text{ième}}$ groupe de paquets.

Pour améliorer la précision de l'évaluation, le procédé est appliqué sur un nombre de groupes de paquets supérieur à 1.

Dans une première variante de réalisation de l'invention, l'évaluation de la bande passante est réalisée en temps réel.

Dans une deuxième variante de réalisation, l'évaluation de la bande passante est réalisée en en temps différé.

Dans une application particulière du procédé de l'invention, le réseau de télécommunication est du type IP.

L'invention concerne également un dispositif d'évaluation de la bande passante entre un premier point et un deuxième point susceptible d'échanger des paquets de données numériques dans un réseau de télécommunication comportant une pluralité de sous-réseaux.

Ce dispositif comporte :

- des moyens de marquage des paquets émis,
- des moyens d'horodatage des paquets reçus,
- des moyens de tri des paquets reçus,
- des moyens pour mesurer les intervalles de temps séparant les instants de réception par le deuxième point des paquets émis,
- des moyens pour calculer la bande passante.

BRÈVE DESCRIPTION DES DESSINS

D'autres caractéristiques et avantages de l'invention ressortiront de la description qui va suivre, prise à titre d'exemple non limitatif en référence aux figures annexées dans lesquelles :

- la figure 1 illustre schématiquement une liaison numérique dans un réseau de télécommunication dans lequel est mis en oeuvre le procédé selon l'invention,
- la figure 2 est un schéma bloc d'un module d'analyse de paquets selon l'invention.

EXPOSÉ DÉTAILLÉ DE MODES DE RÉALISATION PARTICULIERS

L'invention va maintenant être décrite dans une mise en oeuvre dans le réseau Internet.

La figure 1 illustre schématiquement une liaison numérique bidirectionnelle 1 entre un premier terminal A et un deuxième terminal B connectés respectivement à un premier réseau local 4 et à deuxième réseau local 6 et échangeant des données numériques à travers un premier sous-réseau 6 et un deuxième sous-réseau 8 selon le mode TCP (pour Transmission control Protocol) ou selon le mode UDP (User Datagram Protocol). A chaque extrémité de la liaison numérique 1 entre les terminaux A et B sont agencés respectivement un premier et un deuxième modules (10, 12) de marquage des paquets de données émis par le terminal A (respectivement B) et un module d'analyse (14, 16) des paquets de données reçus par le terminal A
(respectivement B).

La figure 2 illustre un schéma bloc d'un module d'analyse selon un mode préféré de réalisation comportant une interface d'adaptation 20 reliée à la liaison IP 1 via un coupleur 22, un module 24 d'extraction de paquets de données de la liaison 1, un module 26 d'acquisition desdits paquets, un module 28 d'horodatage des paquets extraits destiné à associer à un groupe de paquets émis quasi-simultanément un même identifiant temporel, une mémoire 30 destinée à stocker les paquets horodatés, un module 32 de tri des paquets ayant le même identifiant temporel, un module 34 de sélection destiné à isoler les groupes de paquets ayant le même identifiant temporel et le plus grand nombre de paquets reçus, un module 36 de mesure du temps de transfert inter paquet et un module 38 de calcul de la bande passante.

En fonctionnement, chacun des terminaux A ou B peut être simultanément émetteur et récepteur. Les données échangées transitent par les réseaux 6 et 8 dont les encombrements respectifs à un instant donné dépendent du nombre d'utilisateurs connectés. Le marquage des paquets est obtenu suite à une requête envoyée par le terminal récepteur au terminal émetteur. Il peut être réalisé, par exemple, par l'activation de l'option d'horodatage décrite dans la norme RFC 1323.

Pour évaluer la bande passante disponible de bout en bout, le module 24 d'extraction isole les paquets de données transmis pendant un laps de temps très court du terminal émetteur vers le terminal récepteur et transmet ces paquets au module d'horodatage 28 qui associe à chaque paquet une date d'émission. Les paquets sont ensuite stockés dans la mémoire 30. Le module 32 trie les paquets portant la même date d'envoi et les transmet au module 34. Ce dernier sélectionne un nombre entier de groupes parmi les groupes triés comportant le plus grand nombre de paquets et transmet ces groupes au module de mesure 36 qui mesure les intervalles de temps séparant la réception des différents paquets successifs. Les intervalles mesurés sont ensuite transmis au module 38 de calcul de la bande passante qui calcule en temps réel la bande passante de la liaison en fonction de la longueur totale des paquets analysés et de la durée de transmission de ces paquets.

Pour évàluer la bande passante disponible dans chaque sous-réseau, l'analyse des paquets reçus est effectuée par le troisième module 18 agencé entre les sous-réseaux 6 et 8.

REVENDICATIONS

1. Procédé d'évaluation de la bande passante entre un premier point et un deuxième point susceptible d'échanger des paquets de données numériques dans un réseau de télécommunication comportant une pluralité de sous-réseaux, procédé caractérisé en ce qu'il comporte les étapes suivantes :
pour chaque sens de transmission à travers l'un au moins desdits sous-réseaux,

pour chaque sens de transmission à travers l'un au moins desdits sous-réseaux,

a. associer aux paquets émis quasi-simultanément un même identifiant,
b. horodater et enregistrer les paquets reçus,
c. identifier et trier les paquets reçus avec le même identifiant,
d. sélectionner le plus grand nombre entier possible m de groupes de paquets ayant le même identifiant,
e. mesurer les intervalles de temps séparant les instants de réception par le deuxième point des paquets des groupes sélectionnés,
f. calculer la bande passante en fonction du nombre de paquets des groupes sélectionnés et de la durée totale de transmission de ces paquets.

2. Procédé selon la revendication 2, caractérisé en ce que la bande passante est calculée par l'expression suivante :

$$\overline{BW} = \frac{1}{m}\sum_{j=1}^{m}\left[\frac{1}{n_m}\sum_{i=1}^{n_m-1}\frac{l_{i,m}}{t_{(i+1)m}-t_{i,m}}\right]$$

où :

- li,m représente la longueur du paquet de rang i du $m^{ième}$ groupe de paquets,
- ti représente le marquage temporel du paquet de rang i du $m^{ième}$ groupe de
- paquets,

- ti+1 représente le marquage temporel du paquet de rang i+1 du m$^{\text{ième}}$ groupe
- de paquets,
- nm représente le nombre de paquets du m$^{\text{ième}}$ groupe de paquets.

4. Procédé selon la revendication 4, caractérisé en ce que le nombre m est supérieur ou égal à 1.

5. Procédé selon l'une des revendications 1 à 4, caractérisé en ce que le marquage des paquets de données est réalisé au point d'émission sur requête du point de réception.

6. Procédé selon l'une des revendications 1 à 5, caractérisé en ce que l'évaluation de la bande passante est réalisée en temps réel.

7. Procédé selon l'une des revendications 1 à 5, caractérisé en ce que l'évaluation de la bande passante est réalisée en temps différé.

8. Procédé selon l'une des revendications précédentes, caractérisé en ce que le réseau de télécommunication est du type IP.

9. Dispositif d'évaluation de la bande passante entre un premier point et un deuxième point échangeant des paquets de données dans un réseau de télécommunication comportant un module de marquage des paquets émis et un module d'analyse des paquets reçu, caractérisé en ce que le module d'analyse comporte :

- des moyens d'horodatage des paquets reçus,
- des moyens de tri des paquets reçus,
- des moyens pour mesurer les intervalles de temps séparant les instants de réception par le deuxième point des paquets émis,
- des moyens pour calculer la bande passante.

10. Module d'analyse de paquets de données reçus dans un réseau de télécommunication, caractérisé en ce qu'il comporte :

- des moyens d'horodatage des paquets reçus,
- des moyens de tri des paquets reçus,

- des moyens pour mesurer les intervalles de temps séparant les instants de réception par le deuxième point des paquets émis,
- des moyens pour calculer la bande passante.

ABRÉGÉ DESCRIPTIF

L'invention concerne un procédé d'évaluation de la bande passante entre un premier point et un deuxième point susceptibles d'échanger des paquets de données numériques dans un réseau de télécommunication comportant une pluralité de sous-réseaux.

Le procédé selon l'invention comporte les étapes suivantes

pour chaque sens de transmission à travers l'un au moins desdits sous-réseaux,

- associer aux paquets émis quasi-simultanément un même identifiant,
- horodater et enregistrer les paquets reçus,
- identifier et trier les paquets reçus avec le même identifiant,
- sélectionner un nombre entier m de groupes de paquets ayant le même identifiant,
- mesurer les intervalles de temps séparant les instants de réception par le deuxième point des paquets émis,
- calculer la bande passante en fonction du nombre de paquets des groupes sélectionnés et de ladite durée totale de transmission desdits paquets.

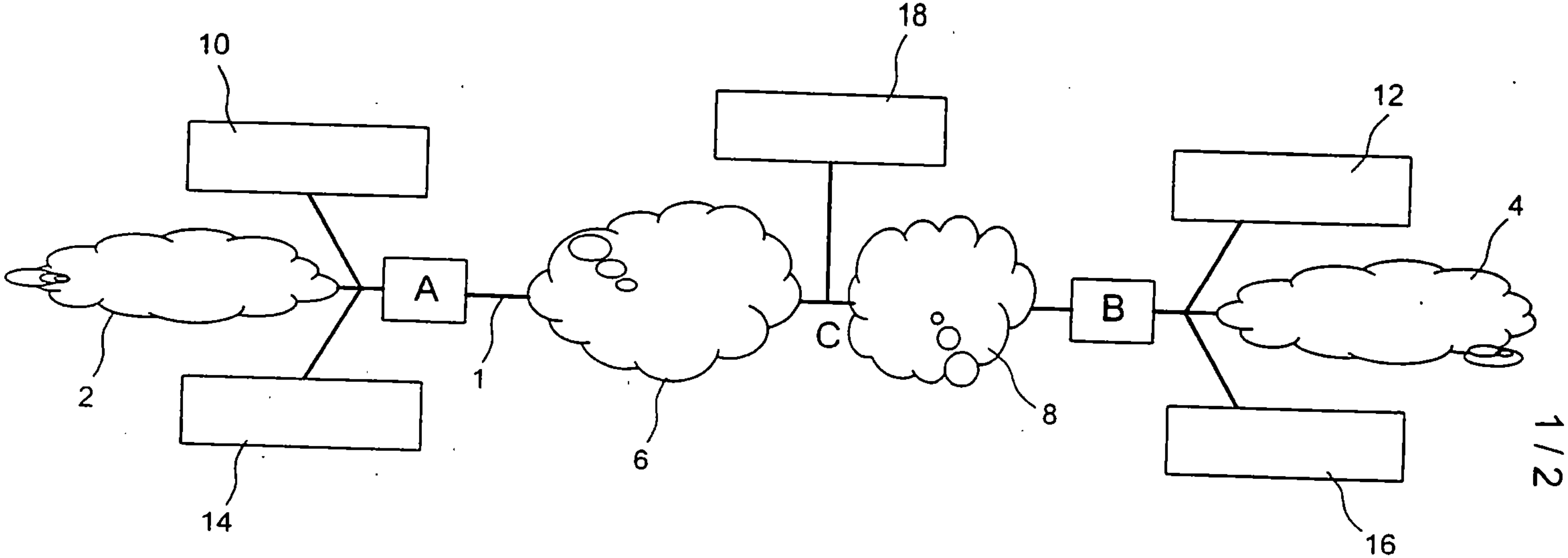


FIG. 1

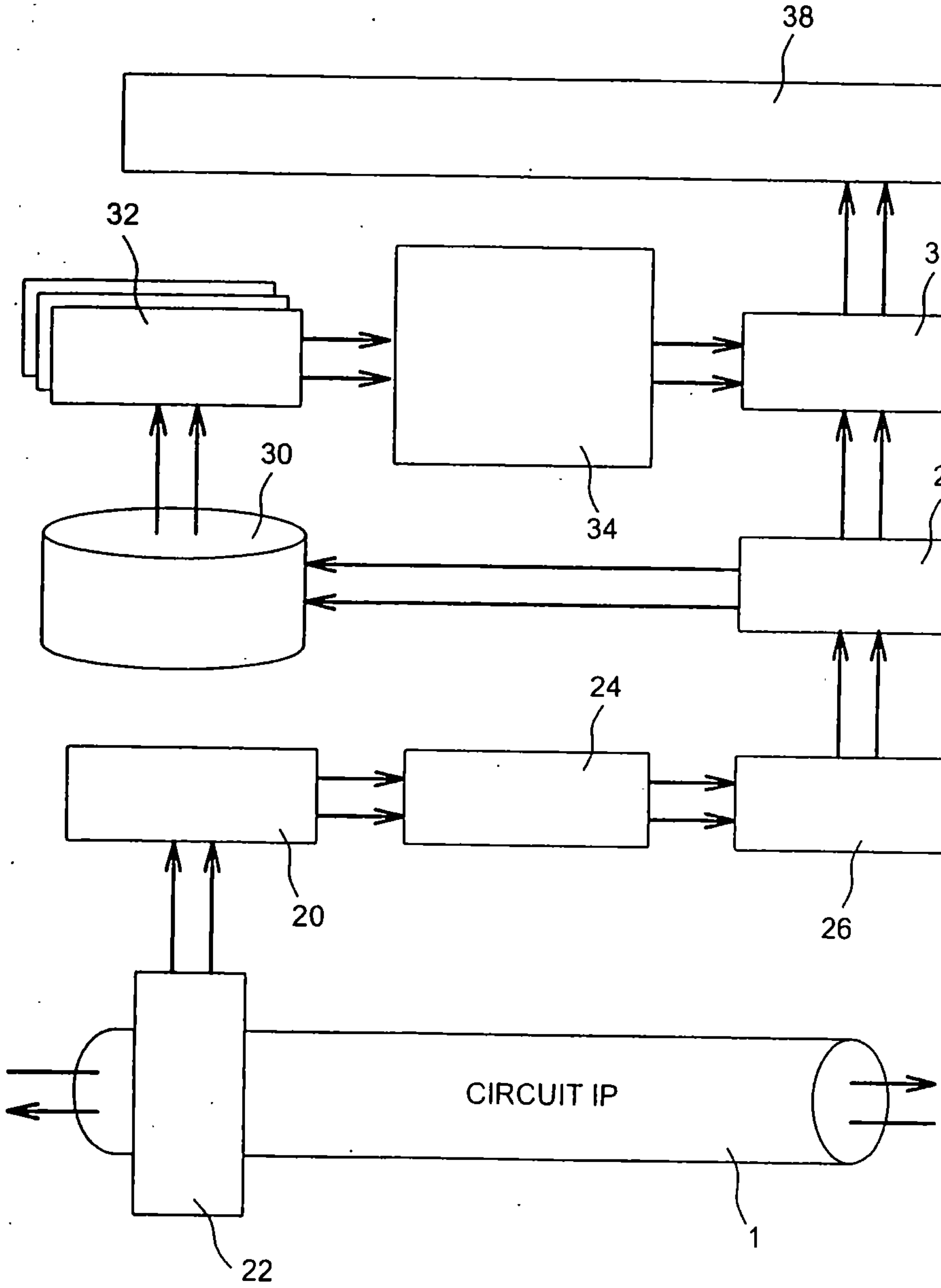


FIG. 2