A WEB BROWSING WORKLOAD MODEL FOR SIMULATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Lourens O. Walters
May 2004

Supervised by
Prof. Pieter S. Kritzinger

Abstract

The simulation of packet switched networks depends on accurate web workload models as input for network models. We derived a workload model for traffic generated by an individual browsing the web. We derived the workload model by studying packet traces of web traffic generated by individuals browsing the web on a campus network.

We attempted to model aggregate traffic generated by many web users browsing the web on a campus network, by decomposing the traffic into its constituent elements i.e. traffic generated by individual users. We furthermore identified elements within the traffic generated by individual users which contributed to the characteristics of the aggregate traffic stream. We identified parameters which were not directly influenced by network specific characteristics such as latency and throughput, in order to ensure that our model was as general as possible.

We found that web traffic was extremely complex. The dynamic behaviour of client and server side scripts introduced dependencies in the data. Our model was more detailed than any existing web workload model at the time the study was conducted, but did not take into account the behaviour of web client and server scripts. There is room for improvement in our model here.

We tried to break down aggregate web traffic into parts which contain observations which were independent from each other. An analysis of autocorrelation between observations within parameter datasets showed that dependencies exist between observations in most of the parameter datasets. The dynamic behaviour of scripts might explain some of the dependencies in the parameter datasets.

We implemented a measurement system which measured data on a campus network by extracting selected information from IP, TCP and HTTP message headers. The system extracted parameter datasets for our workload model from the captured data. The approach of capturing selected information from TCP/IP packets transmitted between web clients and servers as opposed to capturing all the data transmitted, avoided the problem of extremely large amounts of data accumulating over small periods of time. Capturing all the data transmitted between web clients and servers required large amounts of storage space and processing power which were not available to us. By using our measurement system, we were able to record data for a 30 day period, capturing web traffic generated by 6 692 hosts on a campus network.

The measurement system could extract parameter datasets in real-time, or write selected data to secondary storage in order to extract parameter datasets off-line. The real-time version of the measurement system could not extract parameter datasets during peak traffic hours. We used the

off-line version of the measurement system to obtain parameter datasets for the study. We believe that with certain optimisations the real-time system would be able to extract parameter datasets in real-time.

We extracted parameter datasets from data recorded to secondary storage by the measurement system. We used the packet trace method to record data to secondary storage. Because of the nature of the packet trace method of measurement we did not have sufficient information in the recorded data to extract parameter datasets.

We used a heuristic algorithm to extract parameter datasets from the incomplete data. The heuristic algorithm was novel as it used information from TCP, IP and HTTP package headers to recontruct a user's browsing behaviour. This had not been done before. The algorithm used a list of characteristics of web client requests which we compiled by studying packet traces of traffic generated by web users. The algorithm inferred user behaviour from the list of web client request characteristics. By using the algorithm we were able to extract parameter datasets from the incomplete measured data.

We analysed the extracted parameter datasets by using visual techniques and goodness-of-fit measures. We tested several families of mathematical functions in order to find a function which fits the model parameter data well. The parameter datasets were very large. They typically contained millions of entries. The commercial statistical analysis packages we had at our disposal could not analyse datasets with millions of entries. We overcame the problem posed by the size of the datasets by implementing our analysis routines in the R statistical analysis environment. The R statistical analysis environment is a freely available open source software package. We implemented the Anderson Darling and λ^2 goodness-of-fit statistics in the R statistical analysis environment for eight mathematical families of functions. We also implemented the Q-Q and P-P plots in order to visually analyse the data.

We found that the Anderson Darling statistic could not be used to analyse large datasets. The Anderson Darling statistic did not have p-value tables for datasets with more than 200 observations. We used the λ^2 statistic as an indicator of goodness-of-fit. The λ^2 statistic indicated that there was evidence against a perfect fit between parameter datasets and any of the eight mathematical function families we tested for. At least one mathematical function did fit each dataset very well, albeit not perfectly well. The visual evidence provided by Q-Q and P-P plots corroborated this finding. We tabulated the data for each of the parameter datasets. Random values could be generated from the tabulated values.

Acknowledgements

I would like to express my thanks to:

- Betti, for loving me.
- My family: Christi, Laubi, Cristina, Fernando, Thys, Laubscher and Oumie, for putting up with me.
- Prof. Pieter Kritzinger, for your help and support. Your incredible work ethic and pursuit of excellence kept me on my toes. Thank you for supporting me on the long road to the completion of this work.
- Dirk Staehle, Kenji Leibnitz and Prof. Tran-Gia, for receiving me so warmly in Würzburg and for providing me with the idea that developed into this dissertation.
- Ridwaan, Rifaat, Michael, Daniel and Coenrad, for helping me set up the data measurement system on the university's backbone network.
- Ian, Elton, Mike, Farrel, Marc, Andy, Mwelwa, Simon, Nico, Ben, Oksana, Jesse, Andrew, Justin and Jo, for the stimulating friendship that you provided me with during my time in the DNA Group.
- Sam and Matthew, for keeping the computers and network running smoothly in the lab.
- Jeffrey September, for lending me a machine to use for the data measurements.
- Prof. Tim Dunne, Allan Clark and Dr Markus Siegle, for lending your ears and providing advice on statistics.
- Prof. Paul Barford, for your friendly help via email on the implementation of the Anderson Darling and λ^2 statistics in the S-Plus language, and the usefulness of the Anderson Darling statistic in general.
- Prof. Mark Crovella, for your friendly help via email and Perl code for the Hill estimator. Thanks also for your help with the "scaling estimator" and the *aest* program.

- Prof. Sidney Resnick, for your friendly help via email on the analysis of heavy tailed tailed distributions. Thanks also for the lecture notes and advice on how to use the Hill estimator in the S-Plus environment.
- Dr Vern Paxson and Prof. Anja Feldmann, for your friendly help via email on the use of the BLT tool for data measurements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Structural modelling is an approach to network traffic modelling which takes into account underlying characteristics of traffic streams. For example, using this approach Internet traffic can be separated according to **source-destination**, **application** or **user** specific traffic. Source-destination specific traffic is traffic transmitted between pairs of hosts on a network, application traffic is traffic generated by different applications used on a network, and user specific traffic is traffic generated by different users on a network. The **structural** approach to network traffic modelling is based on the principle that network traffic separated into component parts according to some criteria, affords a more accurate traffic model than a model derived from the aggregate network traffic. This approach contrasts the black box approach to modelling which analyses aggregate traffic as a monolithic body of data. By modelling **component traffic streams** it is often possible to shed light on characteristics of aggregate traffic streams e.g. Willinger *et al.* [WPT98] found a strong connection between self-similarity of aggregate network traffic and the occurrence of heavy-tailed, infinite variance distributions within individual **source-destination** network connections.

An example of the **structural** modelling of **application** specific traffic is that of the work by Paxson and Floyd [PF95, Pax94]. They modelled Internet traffic as generated by individual Internet applications in order to ascertain whether the common model of teletraffic arrivals, the Poisson process, was an appropriate model for data networks. The work was done during 1994 and 1995 and covered Internet applications that were predominantly used at the time: `TELNET`, `NNTP`, `FTP` and `SMTP`. The results showed that many aspects of Internet application data e.g `TELNET` packet inter-arrivals and `FTP` data connection arrivals were bursty over many time scales, and that the assumption of Poisson arrivals for these processes was not appropriate.

We employed the structural approach to modelling data traffic to define a **detailed** characterisation of web traffic generated by a single host on a campus network. Our characterisation of web traffic can be described as **detailed** because it takes into account nuances in web traffic patterns generated by users. An example of such a nuance is the difference in file size between textual HTML files downloaded from a server, and the file sizes of a series of graphical files downloaded after an HTML file has been downloaded and parsed by a web browser. Another example of such a nuance is

the difference in the size of inter-arrival times between requests for graphics files contained in a web document, and the size of inter-arrival times between HTML files contained in a web document.

One might ask why a **detailed** web traffic model is necessary, or why modelling nuances in web traffic patterns is important? Surely modelling numerous small request packets following closely after one another is not important when by far the largest percentage of web traffic consists of much larger response packets? One reason for modelling nuances in web traffic patterns is that the resultant model is a more **accurate** characterisation of web traffic than one which ignores nuances in the traffic patterns. It is a difficult task to accurately model a network channel with traffic generated by many web users. If an accurate traffic model for a single web user can be constructed, aggregate traffic can be generated by aggregating the traffic generated by many of these single traffic sources.

Another compelling reason for modelling nuances in web traffic patterns is that nuances in web traffic patterns have a **large impact** on the behaviour of network protocols. For example the interaction of the TCP and the HTTP, and in particular the slow start mechanism of the TCP, has a considerable influence on the performance of the HTTP [HOT97]. The TCP is fundamentally a bulk transfer protocol and is poorly suited to frequent, short, request-response-style traffic such as that of HTTP traffic. Short connections, such as those required to transmit small HTTP request packets interact poorly with the TCP's slow start congestion avoidance algorithm which causes increased latency for web users [HOT97]. A **detailed** web traffic model includes the transmission of small HTTP request packets. Using a **detailed** model in simulations would account for the interaction between the TCP and the HTTP.

Several recent simulation studies have taken these facts into consideration by using detailed web traffic, TCP and radio interface models e.g. Staehle *et al.* [SLT01] used simulation to show that the QoS of Internet Access with GPRS in its first phase is comparable to that of a modem with a speed of 32kbps, for medium traffic loads. Using a bidirectional web traffic model Kalden *et al.* [KMM00] showed that GPRS provides bandwidth-efficient support for bursty applications such as web access.

The **detailed** web traffic model we derived consists of eleven **parameters**. Each of the parameters was modelled by a function of a specified mathematical family. The **mathematical family** and **model constants** of the parameters were derived from empirical data by means of goodness-of-fit and maximum likelihood techniques. The empirical data were obtained by means of measurements taken on a campus network.

At this point we would like to clarify the usage of the terms **parameter** and **model constant** in this dissertation. We refer to an observable (countable or measurable) feature of web traffic that is used within our web traffic model as a **parameter** of the web traffic model. This is in keeping with the usage of this term in the computer science and engineering fields. The term parameter is however also used in the statistical science field to describe a numerical constant that is unknown but whose estimation from data will in some sense characterise the data, given a particular class of mathematical models for the data. We need to describe these numerical constants in this work, and decided to refer to them as **model constants** instead of parameters, seeing as we already use the term "parameter" in its computer science related sense.

1.1 Motivation and Objectives

The work by D. Staehle, K. Leibnitz and P. Tran-Gia [SLTG99], surveyed common models of web traffic for simulation purposes. A web traffic model composed of parameters taken from various studies was proposed in this work, and used in a simulation study [SLT01]. The model suffered from the drawback that it was composed of parts taken from different studies conducted under different conditions. The studies were conducted over a period ranging between 1997 and 2000, and used different approaches to measuring and characterising web traffic.

It might be argued that web traffic models such as the one used by Staehle *et al.*, which were composed of various parameters taken from other studies, provide a reasonable approximation to real web traffic. Without studies to validate these models, the accuracy of these web traffic models is questionable.

In order for a web traffic model to be accurate, it has to be based on recent measurements taken under conditions similar to those being simulated. The web traffic model should be validated against data measured under the same conditions as those that the data the model was derived from was captured.

The objectives of our research were the following:

- **Identify** parameters to be used in a **detailed** web workload traffic model.
- **Obtain** data to analyse.
  - **Implement** a tool which captures web traffic generated by **hosts** i.e. individual machines, on a campus network.
  - **Implement** a tool which processes data recorded during measurement, extracting datasets for model parameters.
- **Identify** statistical distributions which best fit the data contained in parameter datasets.
  - **Develop** a methodology for analysing parameter datasets, overcoming issues such as the **very large size** of parameter datasets. Many of the datasets contained several million observations.
  - **Implement** statistical functions to be used in analysis in a suitable programming language. We had to implement the statistical functions ourselves as many of the functions we used in our analysis were not available in "off-the-shelf" statistical analysis packages or "off-the-shelf" packages could not process datasets containing millions of observations.
  - **Analyse** parameter datasets.
- **Develop** methodology for validating web traffic model. The actual validation of the web traffic model will be part of a future project.

The ultimate goal of our work was to develop a **parameterised web workload model** for simulation purposes.

1.2 Related Work

During the latter half of the 1990's, World Wide Web traffic dominated the Internet and became the main focus of traffic modelling. Work on traffic modelling started to take into consideration the self-similar nature of network traffic soon after the publication of the seminal paper on the self-similarity of Ethernet traffic by Leland *et al.* [LTWW93].

Crovella *et al.* [CB96] showed that web traffic is self-similar, and that the self-similarity is in all likelihood attributable to the heavy-tailed distributions of transmission times of documents and silent times between document requests.

The data traces used in the Crovella study were recorded by Cunha *et al.* [CBC95]. These traces served as the basis of the workload generator SURGE developed by Barford *et al.* [BC98b]. The objective of the SURGE workload generator was to generate traffic representative of that found on the World Wide Web, in order to exercise web servers and networks. SURGE generated web traffic equivalent to a set of real users accessing a web server.

Taqqu *et al.* [TWS97] proved that aggregate World Wide Web traffic as found on Internet links can be modelled by super-positioning many ON/OFF traffic sources where the ON and OFF periods were drawn from heavy tailed distributions. This method afforded the generation of traffic traces for simulation in linear time. Traffic generated by one of the ON/OFF traffic sources in the above-mentioned model was representative of a single web user [LTWW93].

Deng [Den96] proposed an ON/OFF traffic model to be used for the simulation of traffic generated by an individual browsing the web. He derived distributions for the parameters of the model by means of analysing datasets measured on a corporate network. He used probability plots to gauge the goodness-of-fit of analytic distributions to the datasets. The model had the advantage of being simple and of generating self-similar traffic due to the heavy tailed nature of the ON and OFF distributions.

The work by Mah [Mah97] and Choi *et al.* [CL99] on web traffic models for simulation purposes was of particular relevance to us as they proposed **detailed** web traffic models. Section 1.2.1 and Section 1.2.2 discuss these models.

1.2.1 Mah Workload Model

Mah [Mah97] developed an empirical model of web traffic. The parameters modelled were represented by their empirical cumulative distribution functions (as opposed to distribution functions of a specified mathematical family). The Inverse Transformation Method was applied to these functions in order to generate the relevant random numbers. Traffic from the web client to the web server (requests) as well as traffic in the opposite direction (responses) were modelled i.e. bidirectional traffic.

Table 1 shows the parameters modelled by Mah. The parameters were listed in the order they occurred within the model. We used the word `user` to refer to a web user (person) making requests by using a `web client` (browser software).

User Request Size
Web Client Request Size
User Response Size
Web Client Response Size
Number of Web Client Requests
Think Time
Consecutive Document Retrievals per Server
Server Popularity

Table 1: Parameters Modelled by Workload Model Developed by Mah

During a simulation traffic is generated according to Mah's model as follows:

1. A web server is selected according to the `Server Popularity` table (Table reporting relative popularity of servers).
2. A period of length equal to the `Think Time` parameter elapses.
3. A user clicks on a web page generating a request of size equal to the `User Request Size` parameter, which is sent to the web server.
4. The web server receives the request, and responds with a document of size equal to the `User Response Size` parameter.
5. The web client receives the response, and responds by generating a number of requests equal to the `Number of Web Client Requests` parameter, each of size equal to the `Web Client Request Size` parameter.
6. The web server receives each of the requests, and responds with a document of size equal to the `Web Client Response Size` parameter, for each request.
7. The process returns to Step 2 and repeats for a number of times equal to the `Consecutive Document Retrievals per Server` parameter before a new server is chosen in Step 1.

The model does not use inter-arrival times. Measurement of inter-arrival times is affected by factors specific to the network on which measurements are taken. For instance, inter-arrival times are influenced by TCP flow control and congestion control algorithms. These algorithms behave differently in networks with different latency and bandwidth. Different cache management algorithms also affect inter-arrival times. Web proxy cache server/s on different networks commonly use different algorithms to manage cache.

HTTP requests and responses transmitted between web client and server is the lowest level of the Open Systems Interconnection (OSI) network model characterised by Mah's model. One would however expect a **detailed** model of web traffic to model TCP characteristics. The measurement of TCP characteristics depend on network specific factors such as TCP flow control and congestion

control algorithms. TCP characteristics therefore were not modelled by Mah as measuring these characteristics would have resulted in the loss of applicability of the workload model to different types of networks.

Simulation of web traffic using Mah's model must therefore include a simulation of TCP algorithms as well as web proxy caching algorithms if an accurate model of web traffic at the TCP/IP level is needed.

Mah's model was derived from packet traces taken on a campus network. Mah used heuristics to extract datasets for certain of the parameters from the measured data. It was necessary to use heuristics, as the packet traces Mah used to extract datasets from, contained incomplete information about a web user's browsing behaviour. The lack of data about a web user's browsing behaviour was compensated for by using heuristic methods to extract datasets from measured data.

1.2.2 Choi *et al.* Workload Model

Choi *et al.* [CL99] developed an **analytic** "behavioural model of web traffic". By **analytic** model is meant a model for which the parameters of the model are represented by distribution functions of a specified mathematical family. The distribution function for each model parameter was chosen by means of applying the visual Quantile-Quantile plot technique to datasets.

The parameters of Choi's model characterise the behaviour of web users and model unidirectional traffic from a web server to a web client. The model is shown in Figure 1.

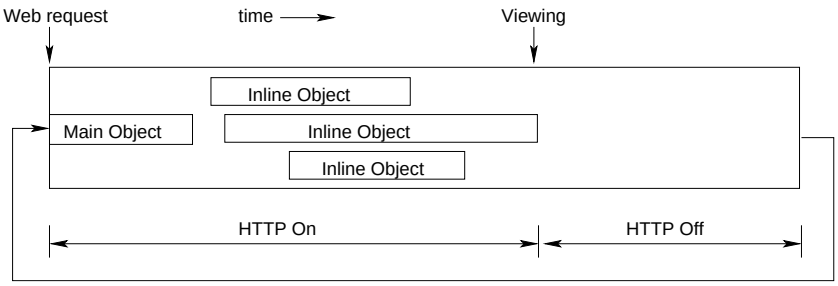


Figure 1: Web Workload Model Developed by Choi *et al.*

Mah's model assumed that a typical web page consisted of a single HTML document (`main object`) followed by several graphics documents (`in-line objects`). The model was based on an unit called a `web request`, which is a set of pages resulting from a user request i.e. a main object and corresponding in-line objects as shown in Figure 1. A `web request` (`HTTP On` period) is followed by a period of time equal to `Viewing Time` (`HTTP Off` period), after which another `web request` is generated.

Table 2 reports parameters modelled by Choi *et al.* The parameters were listed in the order they occurred within the model.

Request Size
Main Object Size
In-line Object Size
Parsing Time
Number of In-line Objects
In-line Inter-arrival Time
Viewing Time (HTTP OFF)
Number of Cached Web Requests
Number of Non-cached Web Requests

Table 2: Parameters Modelled by Workload Model Developed by Choi *et al.*

The `Parsing Time` parameter models the time it takes a browser to parse the HTML code contained in a main object. The `Number of Cached Web Requests` parameter models how many consecutive requests are cached i.e. the requested page is locally cached, and the `Number of Non-cached Web Requests` parameter models how many consecutive requests are not cached. The significance of these two parameters is that `web requests` which are cached locally don't result in network traffic being generated, as the relevant main and in-line objects are retrieved from the local cache.

Traffic is generated as illustrated in Figure 2.

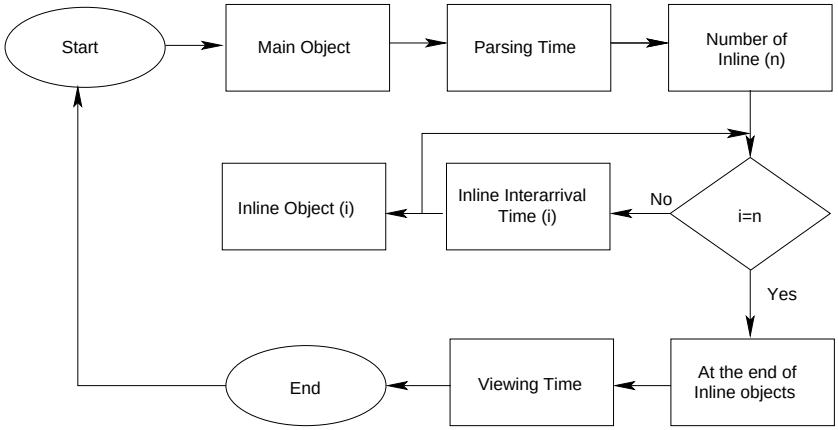


Figure 2: State Transition Diagram of Traffic Generation by Workload Model of Choi *et al.*

Generated traffic is unidirectional, from web server to web client. The `Request Size` parameter in Table 2 is not used during traffic generation. A main object of size equal to the `Main Object Size` parameter is generated, followed by a time period of length equal to the `Parsing Time` parameter elapsing. A number of in-line objects equal to the `Number of In-line Objects` parameter, each of size equal to the `In-line Object Size` parameter is then generated. The inter-arrival time between in-line objects is of length equal to the `In-line Inter-arrival Time` parameter. After all the in-line objects have been generated a time period of length equal to the `Viewing Time` parameter

elapses before the process repeats itself.

The caching parameters: `Number of Cached Web Requests` and `Number of Non-cached Web Requests` supplement the model by adding local caching behaviour to the model. A two state renewal process is used, representing cached and non-cached web requests. As long as the process is in the cached state, network traffic for web requests is not generated i.e. web requests are retrieved from local cache. Traffic is generated when the process is in the non-cached state.

Distributions for the model parameters were derived from data obtained by means of packet traces measured on a campus network. TCP, IP and HTTP headers were captured and analysed. Most of the parameter datasets were obtained by using heuristic methods.

1.3 Concluding Remarks

Mathematical models are often classified into two classes, **white box** and **black box** models. The two classes differ in the amount of *a priori* information that is used to construct the model. Practically all problems fall somewhere between these two classes. A distinction can however be made based on these classes, between models which are constructed from more, or less *a priori* information.

When detailed information about network traffic is not considered during the modelling process, the resultant model can be classified as a black box model. Black box models commonly consider traffic as a single body of opaque data transferred across a network. Detailed aspects of complicated interactions between hosts on a network are not modelled. A black box traffic model characterises salient characteristics of traffic, but detailed characteristics of traffic are lost. Black box models have few **model constants** and random numbers can easily be generated from them. Appendix E lists commonly used black box traffic models.

Traffic generated by black box models suffer from a loss of accuracy, which is a considerable disadvantage of this approach. It is a known fact that using more detail in modelling a physical phenomenon results in a more accurate model of the physical phenomenon. The **white box** modelling approach to network traffic modelling incorporates detailed information about traffic in models. The increased level of detail in these models results in more accurate traffic being generated by these models. The **structural** approach to network modelling, which takes into consideration underlying characteristics of traffic streams, is an example of the **white box** traffic modelling approach.

There is however a tradeoff between model detail and computational complexity. The more parameters the model has the more complex it is to analyse or solve. The model we developed is to be used in simulation studies. We decided on using a model with eleven parameters. By employing the **structural** approach to traffic modelling it was possible to model characteristics of traffic generated by web clients and servers in great detail. Simulation environments using a structural model for web traffic can generate traffic patterns which are representative of the complex ever-changing nature of real web traffic.

It is necessary to estimate model constants used in the structural model from the actual network

that is being simulated. The model constants can be estimated by capturing some traffic from the network being simulated, and calculating values for the model constants from the captured data. This method ensures that the **structural** web traffic model generates traffic which is equivalent to that found on the network being simulated. It is therefore possible to use the **structural** model to generate traffic equivalent to web traffic found on different types of networks e.g. wireless, Local Area Network or Wide Area Network.

The Internet is evolving rapidly. Technologies such as web servers providing dynamic content to web clients by means of web services and server side scripting are widely used. Web-pages often consist of multimedia content, relying on Java and Flash technologies. The creation of new technologies constantly introduce new traffic patterns which cannot be accounted for by **black box** traffic models. A **structural** web traffic model provides the means to accurately generate web traffic which is equivalent to traffic generated by new web technologies. What is more, traffic can be generated to be equivalent to that found on different types of carrier networks.

The model we derived is that of traffic generated by a single web user. It is possible to generate traffic for many web users by aggregating the traffic streams generated by several single users. The contributions of the dissertation are the following:

1. A **survey** of web traffic models used in simulation studies.

2. The **development** of a detailed **structural workload model** of web traffic generated by a single host on a campus network. The model models HTTP messages exchanged between a web client and web servers. The parameters of the model were carefully chosen to be as independent as possible from network conditions such as available bandwidth, network congestion and congestion control mechanisms. The reason why parameters were chosen to be independent from network conditions was to ensure that network conditions during measurement of data did not influence the outcome of analysis of the data. The parameters of a model such as ours can never be completely independent from network conditions. Network conditions can influence a web-user's browsing behaviour indirectly e.g. by causing users to avoid certain slow websites.

3. The **implementation** of a **measurement system** which captured web traffic on a campus network. The system **extracted** selected information from TCP, IP and HTTP headers and wrote it to secondary storage. The information captured was sufficient to reconstruct HTTP dialogues between web clients and servers during the subsequent processing of data. The volume of data written to secondary storage was considerably much less than that which was actually transmitted between web clients and servers. The selective recording of data transmitted between clients and servers allowed us to capture web browsing data for thousands of users over a period of several weeks with small storage requirements (approximately 30GB of disk space). The measurement of web traffic generated by thousands of users would not have been possible if all the data transmitted by each user were recorded.

4. The **implementation** of **processing tools** which extracted datasets for each parameter of the workload model from the data recorded by the measurement system. The tools reconstructed HTTP dialogues between web clients and servers from fragmented, unordered and incomplete information about HTTP dialogues stored in the data recorded by the measurement system. The tools achieved the reconstruction by gracefully dealing with protocol errors and missing information and by using a **heuristic algorithm** to deal with missing information.

5. The **development** of a **statistical methodology** to find best-fit distributions of a specified mathematical family for very large datasets such as the datasets analysed in network traffic studies. We applied the methodology to find distributions for each of the model parameter datasets we extracted from data measured on a campus network. By distribution we mean a distribution family e.g. the normal or exponential families as well as the parameters associated with the family e.g. location, shape and scale.

   The model parameter datasets we analysed typically contained millions of entries. Commercially available statistical analysis software typically do not support the analysis of very large datasets i.e. datasets which have millions of entries. Specialised goodness-of-fit statistics are typically not implemented in commercially available software packages. We used the **Anderson Darling** and λ^2 **discrepancy** goodness-of-fit statistics as they were used in studies similar to ours [Pax94, Fel98, BC98b]. During analysis we found the Anderson Darling Statistic to be imprecise when used on very large datasets. We used the λ^2 discrepancy statistic for analysing very large datasets. The imprecise results obtained from the Anderson Darling Statistic was not investigated further. We intend to follow up on our findings concerning the Anderson Darling Statistic in future work.

   The "R statistical environment" is a freely available environment for the implementation of statistical functions. The environment supports the analysis of very large datasets. We **implemented** the Anderson Darling and the λ^2 discrepancy statistics in the R environment i.e. we wrote the code for these statistics in the R programming language. We also **implemented** several plots used to visually judge goodness-of-fit such as Q-Q and Probability Plots in the environment. We **implemented** Perl scripts to format and manipulate parameter datasets into files suitable for analysis by the R environment.

   Our statistical methodology consists of formatting parameter datasets by using our Perl scripts, and then plotting the data by using the Q-Q and Probability Plots functions we wrote in the R environment. The λ^2 discrepancy measure is then applied to the data. If a suitable fit to a mathematical function is not found for a parameter dataset, we investigate whether it is not possible to split the dataset by using ancillary information about the data e.g. some observations might be affected in a certain way because of certain properties they have. If no split in the data is possible, we investigate the possibility of using a **hybrid** model. A hybrid model is a model which uses different mathematical functions for different ranges of model parameter values. We do this by using **censoring techniques**.

6. We **suggested** a methodology for **validating** our web workload model. We also suggested a methodology for testing the relationship between the existence of heavy-tailed distributions for characteristics of traffic generated by individual users browsing the web and the self-similarity of the inter-arrival times generated by many users browsing the web. The methodology for validating our web work workload model, as well as the methodology for testing the relationship between the self-similarity of traffic generated by many users and the existence of heavy-tailed distributions in traffic generated by a single users, is based on implementing our workload model in the *ns* network simulator. *Ns* is a network simulator with a long history of use in research. It is very well documented and has several traffic generation modules [UC 02]. We provided suggestions for updating existing traffic generator classes in the *ns* simulator in order for the simulator to generate workloads according to our workload model.

It should be mentioned why we chose to characterise model parameters by means of mathematical functions rather than empirical distribution functions i.e. tabulated values for model parameters. By using mathematical functions we were able to generate random numbers for the model parameters by means of their mathematical functions. It was however also possible to generate random numbers by using empirical distribution functions. The advantage of using mathematical functions to describe model parameters was that the functions described valuable information about the distribution of model parameters.

A mathematical description of a parameter conveyed properties of the parameter which were of importance to us. The shape of the distribution of data were indicated by means of the mathematical family which characterised it. It was for instance possible to ascertain which model parameters had **heavy-tailed** distributions by means of considering the mathematical function which characterised it.

We found that the data were more complicated than we initially anticipated. The workload model which we defined oversimplified the complicated nature of modern-day web traffic. The findings of our analysis of the data showed that well-known mathematical functions could not be fitted to datasets of the eleven parameter datasets. Although in some cases the fit between data and mathematical function was good, in most cases the data showed that more complicated functions were necessary to model the data. Or alternatively, the web browsing model should have been broken down into more components to take into account the complexities of the data.

We found that in most cases the fit between the data and mathematical function was close enough to accurately generate random numbers for web traffic generators used in simulation. Compared to the traffic generators currently being used the model we derived was a vast improvement. The implementation of the model in a traffic generator and the validation of traffic generated by such a generator was left for future work.

We created an empirical representation of the eleven parameter datasets. Using the empirical cumulative distribution of each of the eleven parameter datasets web traffic could be generated by applying the inverse transformation method. We included the empirical representation of parameter datasets in the study for completeness sake. Although empirical distributions were not as interesting

as their mathematical counterparts, they could be used to generate traffic which is nearly identical to the original traffic. This is useful for people who are interested in generating web traffic of the exact kind that was found on a typical campus network.

Chapter 2

Web Browsing Workload Model

2.1 Introduction

Developing a web workload model involves five steps: identifying important parameters, obtaining traffic measurements, processing traffic measurements, analysing traffic measurements and validating the resultant workload model.

This chapter deals with the first step, identifying important parameters. A workload model is a collection of parameters that include key features of the workload it models. The purpose of a workload model determines which parameters are included in the model. Once the purpose of a model is identified, the parameters should be chosen to reflect the purpose, keeping in mind issues such as level of detail, underlying system properties, and independence from other parameters.

Section 2.2 discusses the purpose of the workload model we derived. We discuss issues surrounding the choice of workload parameters in Section 2.3. Section 2.4 discusses which parameters we chose to include in our workload model based on a study of measurements of web traffic we conducted on the local area network in our laboratory. We discuss web traffic models used in open source network simulators, paying particular attention to the web traffic model of the *ns* network simulator, in Section 2.5. Section 2.6 defines our web traffic workload model.

2.2 Purpose of Model

We developed a web workload model for use in simulation studies of low bandwidth wireless networks. The model generates traffic for an individual user browsing the web. We realised the need for such a model after reading the simulation studies of Staehle *et al.* [SLT01] and Kalden *et al.* [KMM00]. They composed inaccurate web workload models from separate components taken from different studies as input models for their simulations.

The model we developed characterises web "browsing" traffic. Web traffic which can be categorised as "bulk" traffic such as web-download traffic, and "interactive" traffic such as web-irc traffic

were not included in the model. Traditionally "interactive" traffic such as `TELNET` traffic and "bulk" traffic such as `FTP` traffic had been modelled separately from "request-response" traffic such as web traffic. We observed that characteristics of these types of traffic are different from web browsing traffic. With the web not being used solely for its traditional purpose of browsing web pages but also for downloading multimedia files and interactive discussions on forums, we had to remove "bulk" and "interactive" traffic from the files recorded by our measurement system. We discuss the process of removing the unrepresentative data from measurements in Section 4.7.

We could not obtain traffic measurements of individual users browsing the web on a wireless network. We therefore measured traffic of users browsing the web on a fixed network. A fixed network typically has more bandwidth than a wireless network. The campus network we took measurements on was however mostly congested during the period of measurement. There were approximately 15000 registered students at the university at the time of measurement, most of whom had Internet access rights. The Internet bandwidth at the time of measurement was 6Mbs. The effective bandwidth as measured by the transfer rate of web transfers during the period of measurement was between 3Kbps and 8Kbps on average, which is less than the total available bandwidth on a GPRS channel which is usually between 9.6Kbps and 33.6Kbps. In terms of available bandwidth the campus network at the time of measurement was comparable to that of a wireless network using GPRS.

Although we intended to derive a workload model for a web user on a wireless network, the model we derived is that of web traffic generated by a single host on a campus network. The measurements used to derive the workload model were measurements of traffic generated by web users using hosts on a campus network. By host is meant a machine connected to the campus network. We measured traffic generated by single user hosts i.e. workstations used by a single user at a time. The similarity between the available bandwidth on the campus network we took our measurements on and a GPRS channel provides us with reason to believe that traffic generated by the web workload model we derived is comparable to traffic generated by web users on a wireless GPRS network.

2.3 Parameter Choice

We chose model parameters with the objective of traffic generated by the workload model to have the same properties as observed in measured web traffic. We observed web traffic to have the following properties:

- Web traffic is bidirectional, the study by Caceres *et.al* [CDJ91] showed that it is important to model bidirectional WAN traffic as a large percentage of WAN traffic is strongly bidirectional. The percentage of traffic travelling in one direction differs from the percentage of traffic travelling in the other.

- HTTP is a request-response protocol i.e. for every request by a client the server sends a response. Requests can be pipelined and responses are received in the same order in which

requests were sent.

- Characteristics of traffic travelling from a client to a server in terms of inter-arrival time and size distribution are not the same as characteristics of traffic travelling from a server to a client.
- Web traffic is generated by processes which can be layered into user sessions and dialogue sessions. User sessions describe the behaviour of web users browsing the web. Dialogue sessions describe the interaction between web client and web server software.
- Aggregated web traffic streams generated by web users are bursty on many time scales i.e. self-similar.

It is important that parameters chosen are not influenced by underlying network properties such as latency, throughput or packet loss rate. It is not possible to decouple parameters completely from underlying network properties. A slow connection will cause users to stay away from bandwidth intensive web sites or might cause users to stop browsing altogether. In constructing the model we used parameters which are least affected by network properties. We did not model inter-arrival times of messages which are affected by network properties such as bandwidth and latency i.e. we did not model inter-arrival times of packets which have been transmitted across the network.

To aid in the selection of model parameters we studied the following material:

- HTTP specifications as recorded in RFC's 1945, 2068 and 2616 [BLFF95, BLFea97, BLFea99].
- Web workload models derived by other authors.
- Traffic traces of users browsing the web.
- Web workload model used in the `ns` simulation package.

We discussed workload models by other authors in Section 1.2. We summarised insights gained into web traffic by means of studying web traffic traces in Section 2.4. Our findings of the study on the web workload model used in the `ns` simulator is summarised in Section 2.5.

2.4 Web Traffic Packet Traces

We recorded packet traces of fellow students browsing the web on the LAN in our laboratory, using the `tcpdump` tool. We studied the traces using the `tcpshow` tool to find relationships between characteristics of traffic and the actions, both user and software initiated, which caused them.

2.4.1 Inter-arrival Times

We found that inter-arrival times of HTTP requests can be divided into three broad categories based on the actions which caused them:

- `Inter-arrival times` $>$ 15 minutes

- 1 second < `Inter-arrival times` < 15 minutes
- `Inter-arrival times` < 1 second

We observed that request inter-arrival times greater than 15 minutes are commonly caused by periods during which no traffic is transmitted due to users not browsing any longer. We observed that users seldomly spend more than 15 minutes reading a single web page. An inter-arrival time greater than 15 minutes typically indicates that the user is engaged in an activity other than web browsing.

Inter-arrival times with a size between 1 second and 15 minutes are commonly caused by users browsing the web i.e. users requesting main objects from web servers by clicking on hypertext links. Inter-arrival times with size smaller than 1 second are commonly caused by web clients requesting in-line objects from servers.

These observations do not always hold, it is possible for a user to request two web pages within a second of one another and it is also possible for a web client to place requests for in-line objects with inter-arrival times greater than a second. We however observed that users seldomly manage to click on two successive links within a second of one another, whereas a web client, being a software process, typically places requests for in-line objects in the order of hundreds of microseconds from one another. Figure 3 illustrates a typical scenario during which web user and web client requests are made during a browsing session.

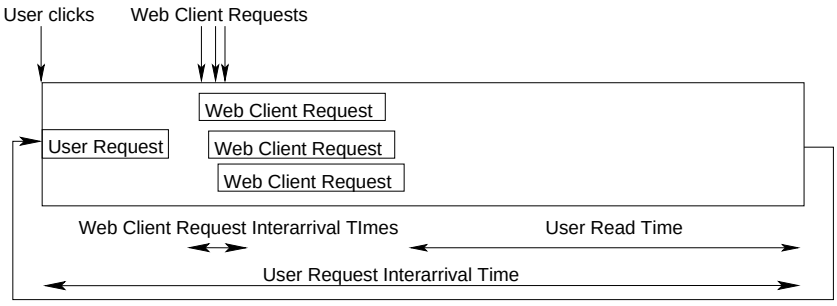


Figure 3: Inter-arrival Time Differences between Web User Requests and Web Client Requests

Figure 3 shows a web user request followed by several web client requests for in-line objects. The process repeats for new web user requests. Typically a web user request inter-arrival time incorporates the time it takes to download and parse the web user response and all the subsequent in-line objects, as well as the time it takes the user to read the web page. This browsing scenario assumes that the user first reads a web page before requesting another which is not always the case. We observed that when a user requests successive pages directly after one another he seldomly manages to click on two successive links within a second of one another.i

As opposed to web user inter-arrival time which is dependent on factors such as the time it takes to download and read a web page, web client request inter-arrival time depends only the speed at

which the web client can parse an HTML file and place requests for embedded in-line objects. We observed the process by the web client of placing requests to typically take in the order of hundreds of microseconds.

Based on the observations of inter-arrival time characteristics we made by studying recorded packet traces, we identified three different inter-arrival time parameters to be used in our model:

- `Browsing Inter-Session Time`
- `Web User Request Inter-arrival Time`
- `Web Client Request Inter-arrival Time`

The `Browsing Inter-Session Time` parameter models the time between sessions during which users browse the web. Our approach of modelling the time between the end of one session and the start of another is different to that of previous work in the area [RLGPC+99, AW95] which model the time between the start of browsing sessions i.e. the inter-arrival time between sessions as opposed to the inter-session time. Modelling the inter-arrival time between browsing sessions results in a model which allows for two browsing sessions by the same user to occur at the same time. The modelling of concurrent browsing sessions by the same user happens when the browsing inter-arrival time between a particular session and the next is shorter than the duration of the session as determined by other model parameters such as the number of requests during the session etc. A new browsing session then starts before the previous one has ended. The occurrence of concurrent browsing sessions by the same user is clearly not possible in practice. We avoid concurrent browsing sessions by the same user from occurring by modelling the inter-session as opposed to inter-arrival time of browsing sessions.

The `Browsing Inter-Session Time`, `Web User Request Inter-arrival Time` and `Web Client Request Inter-arrival Time` are the only inter-arrival time parameters we used in our model. We mentioned in Section 2.2 that it is not advisable to use inter-arrival time values as measured on a network in workload models due to the influences of network characteristics on inter-arrival time parameters. Models based on such data cannot be applicable to other network scenarios, a fact which limits the utility of the models. The inter-arrival time parameters we used are not network related. They model values related to user and web client software behaviour and are therefore not directly affected by network characteristics.

Due to the nature of the environment in which we collected data for the derivation of our workload model, the inter-arrival time parameters were indirectly affected by network characteristics. We will show in Section 3.6 that the influence on the parameters was minimal.

2.4.2 Resource Sizes

We observed that web requests were on average much smaller than web responses. Web requests consist of an HTTP header which contains a few fields used by the HTTP and which includes the URL of the requested object. They do not have a message body and typically have a size of

approximately 500 bytes. Web responses contain a message body which is either a main object in the form of an HTML file or an in-line image.

HTML files were observed to be larger, on average, than image files. The fact that HTML files are larger size than image files is difficult to believe as image files are commonly larger than text files. Web-pages are however usually composed of many small images arranged to create a complete picture. These small images are also typically optimised in size in order to reduce download times and hence page-rendering times. We did however observe a significant number of image files which were much larger than the average HTML file. These files are typically the larger images displayed on some web-pages. We observed HTML files to have an average size of around 8KB and image files to have average size of around 4KB.

It is clear that web requests are much smaller than web responses. It is also clear that web user requests have different size characteristics from web client requests. We could however not determine whether web user requests have different size characteristics from web client requests. Observations based on the packet traces indicated that there is no difference in size between the two quantities. Workload models derived in previous work model the parameters separately [Mah97]. We follow the example of Mah [Mah97] and model the two parameters separately. We identified four resource size parameters to be used in our model:

- `Web User Request Size`
- `Web Client Request Size`
- `Web User Response Size`
- `Web Client Response Size`

2.4.3 Number of Requests

We need two more parameters in order to complete our workload model:

- `Number of Web User Requests per Browsing Session`
- `Number of Web Client Requests per Web User Request`

Workload models derived in previous work do not model web traffic at the level of user browsing sessions [Mah97, CL99]. The `Number of Web User Requests per Browsing Session` parameter was chosen to model the behaviour of users in terms of the duration of browsing sessions. It measures the number of times a user clicks on hypertext links during a browsing session.

The `Number of Web Client Requests per Web User Request` parameter was chosen to model the number of in-line images downloaded after a request for a web page is made.

2.5 Ns Web Workload Model

The purpose of the workload model we derived is to generate web traffic for simulation studies. We constructed a generalised model which can be implemented in any network simulator or used in a simulator developed from scratch. We studied several network simulation packages and libraries to determine how web traffic is generated in these packages. We considered only open source software because of the possibility of extending open source packages with our traffic model. We looked at the OSSCAR, INSANE, REAL, Omnet++ and ns packages. We found all the packages except for ns to have either inadequate or no web traffic models at all. Ns has a long history as a simulator used in research. It is also very well documented and has several traffic generation modules [UC 02].

Network Simulator (ns) was developed as a collaboration between the University of California at Berkeley, Lawrence Berkeley National Laboratory and the VINT project. Ns version 2 is a simulator targeted at networking research and provides substantial support for simulation of TCP/IP, routing, and multi-cast protocols [Com]. It has substantial support for the simulation of wireless networks [UC 02]. We summarise some aspects and shortcomings of the web traffic model used in ns next. We discuss an implementation of our model as an extension to the current web traffic model used in ns in Section 7.2.

Ns is an object oriented application and therefore the web workload model is described using the object oriented paradigm. Traffic generation in ns is based on the following three application classes:

1. `Web client`
2. `Web server`
3. `Web proxy cache`

Web traffic is modelled by connecting web clients to web servers, either directly or via a web proxy cache. Figure 4 shows an arbitrary dumbbell network topology. Using the ns web workload model, one can configure web clients, servers and caches into any arbitrary configuration. For example, we can configure nodes 7-11 to be web clients and nodes 2-6 to be web servers. Nodes 0 and 1 can be configured to be web proxy caches. We can just as easily configure all the nodes to be both web clients and web servers, with no web cache proxies involved.

The ns web workload model can be used as as follows:

A web client generates requests. The inter-arrival times are generated by an `Interval-Generator` object. The `Interval-Generator` uses one of the following random variables to generate inter-arrival times:

1. `Uniform`
2. `Exponential`
3. `Pareto`

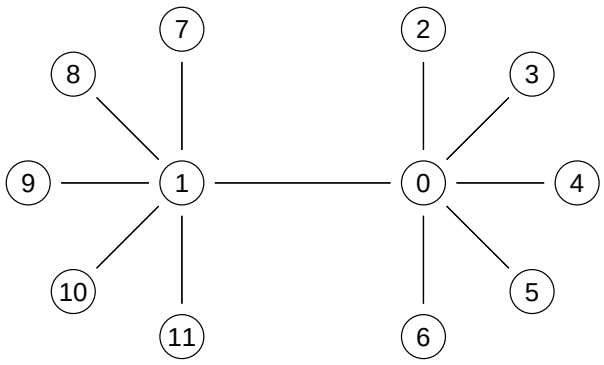


Figure 4: A Dumbbell Network Topology in a ns Simulation Environment

4. `Constant`

5. `Hyper Exponential`

The request sizes are generated by a `Page-Generator` object. The `Page-Generator` uses one of four `PagePool` classes to generate the request sizes. The four `PagePool` classes are:

1. `PagePool/Math` - There is only one type of web object available for request. The size of the object is generated by a random variable chosen from the list of random variables mentioned above.

2. `PagePool/CompMath` - Improves on `PagePool/Math` by using a compound web page model consisting of a main text page object followed by a number of in-line image objects. The sizes of main and in-line objects are fixed. The number of in-line objects is also fixed.

3. `PagePool/ProxyTrace` - Uses an empirical web proxy cache traces to generate request sizes.

4. `PagePool/Client` - Used by caches to keep track of pages resident in cache. Not used for generating request sizes.

Our workload model uses mathematical functions to characterise workload parameters. The `PagePool/ProxyTrace` class is therefore not of interest to us as it uses empirical traces to characterise workload parameters. The `PagePool/Client` does not generate request sizes and is therefore also unsuitable for our purposes. Of the remaining two `Math` classes the `PagePool/CompMath` class allows for main and in-line objects which is consistent with our model. We will therefore concentrate on the `PagePool/CompMath` class.

Traffic generated by ns using the `PagePool/CompMath` class is visualised in Figure 5.

Figure 5 shows a stream of main and in-line objects generated by a web server in response to requests by both a web user and a web client. There are a number of problems with the workload model shown in Figure 5.

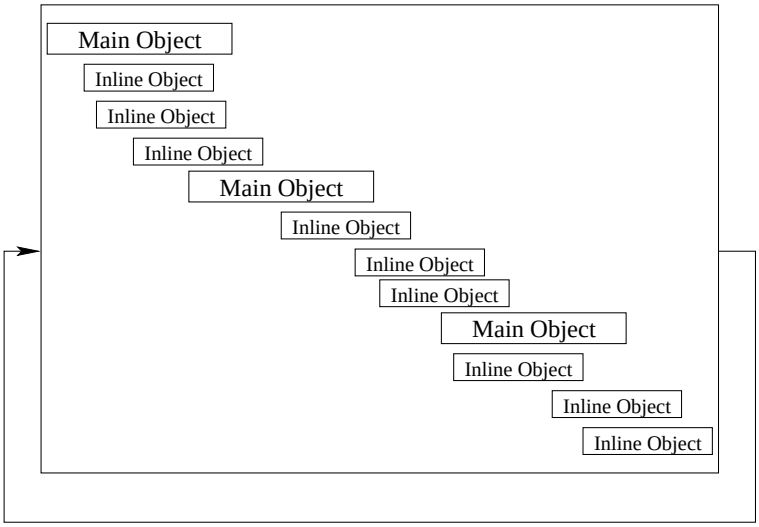


Figure 5: Traffic Generated by Web Workload Model in ns Network Simulator

1. The number of in-line objects is fixed.
2. The sizes of the main and in-line objects respectively are fixed.
3. The inter-arrival times between all objects are drawn from the same distribution.

The ns web workload model is bidirectional and allows for the flexible construction of web client and web server processes. The traffic generated by the model does not reflect some of the characteristics of web traffic which we observed in web traces studied, as discussed in Section 2.4. Traffic generated by the ns web workload model will therefore not accurately characterise real web traffic. The shortcomings of the ns web workload model were considered in the construction of our web workload model.

2.6 Model Definition

The model we developed is layered and models bidirectional traffic from web client to web server and vice versa. We included parameters which had been shown to contribute to the self-similar nature of web traffic inter-arrival times [CB96]. Size distributions with heavy tails and ON/OFF arrival processes with ON and OFF periods drawn from heavy tailed distributions had been shown to contribute to self-similarity in network traffic. We included size and inter-arrival time parameters which our analysis showed to have heavy-tailed distributions.

A `Web User Request` and `Web Client Request` were defined as follows:

A Web User Request is a request made by a web user clicking on a hypertext link or entering an URL in a web client's address text-box.

A Web Client Request is a request made by a web client e.g. Mozilla, Galeon, Konqueror, Netscape, Lynx or MS Explorer.

A `Web User Response` or `Web Client Response` is a response from a web server to either a `Web User Request` or `Web Client Request`.

The model consists of two layers, the `Browsing Session Layer` and the `Client Server Dialogue Layer`. The model is shown in Figure 6. The `Browsing Session Layer` characterises the behaviour of web users by means of modelling the length of browsing sessions and number of requests per browsing session. The `Client Server Dialogue Layer` characterises the interaction between web clients and servers by modelling the number of requests for in-line objects and the size of and inter-arrival time between messages exchanged between web client and server.

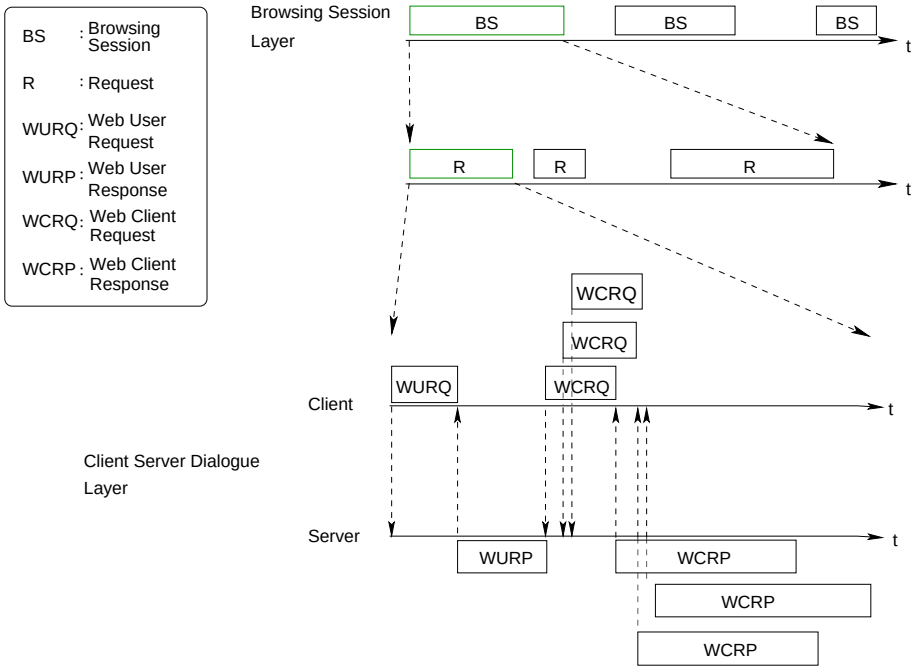


Figure 6: Web Browsing Traffic Generated by the Bidirectional, Layered Workload Model Developed by Us

A short description of each layer follows:

The Browsing Session Layer models the time during which a user browses the web i.e. the time spent requesting and reading documents by using a web browser/ client.

The Client Server Dialogue Layer models interaction between a web client and server which is initiated by a `Web User Request`. The arrows in Figure 6 indicate when a request or response is sent, and the size of a box indicates the size of the file and hence the time it takes the file to reach the other side. A response is sent immediately after a file is received by the web client

or server. A typical scenario is for a user to request an HTML file. On arrival at the server side, the server responds to the request by sending the requested file to the client. The client responds by sending requests for graphical objects that need to be displayed along with text. On arrival at the server side, the server responds to the requests by sending the requested files to the client.

The scenario described above and illustrated in Figure 6 can be modelled by means of eleven parameters. The parameters are shown in Table 3.

No.	Name
1	Browsing Inter-Session Time
2	Number of Web User Requests per Browsing Session
3	Number of Web Client Requests per Web User Request
4	Web User Request Inter-arrival Time
5	Web Client Request Inter-arrival Time
6	Web User Request Size
7	Web Client Request Size
8	Cached Web User Response Size
9	Non-Cached Web User Response Size
10	Cached Web Client Response Size
11	Non-Cached Web Client Response Size

Table 3: Parameters Modelled by Web Browsing Workload Model Developed by Us

Parameter No. 1 models the time between the end of a `Browsing Session` and the start of the next. Parameter No. 2 models the number of web pages requested during a `Browsing Session`. Parameter No. 3 models the number of in-line objects (e.g. image and flash files) contained in a web page. Parameter No. 4 models the time between requests for web pages. Parameter No. 5 models the time between requests for in-line objects. Parameters Nos 6 and 7 model the size of requests generated by web users and web clients respectively. Parameters Nos 8-11 model the size of responses to requests.

2.7 Concluding Remarks

We employed the structural approach to network workload modelling to derive a model of web traffic as generated by a single user browsing the web on a campus network. Due to low Internet bandwidth and severe network congestion on the campus network during the period of measurement the available bandwidth on the network was comparable to that of a low bandwidth wireless network.

We modelled web traffic at the application level. HTTP messages exchanged between a web client and web servers were modelled. We did not model traffic at the level of the TCP and the IP. A network simulation which uses our traffic model along with these protocols would have to implement the appropriate models for TCP and IP. The ns network simulator which we targeted as

the simulation package for implementation of our model has an implementation of TCP and IP.

We attempted to avoid influences of network conditions such as available bandwidth, network congestion and congestion control mechanisms on model parameters such as `message inter-arrival time` and `message size` during data measurement. Influences of network conditions were avoided by using model parameters in our model which were not directly influenced by underlying network conditions. Model parameters which model the behaviour of web-users and web-browsers and the sizes of web-pages were not directly influenced by network conditions. Underlying network factors did however indirectly influence model parameters i.e. a congested network caused users to change their browsing behaviour to adapt to the congestion e.g. users might have visited fewer web-sites during periods of congestion, or might have avoided sites with longer downloading times. It was not possible to avoid influences of underlying network conditions on network parameters completely, but we tried to minimize the impact of these conditions on our model.

We chose the parameters for the model by studying packet traces of web traffic. The choice of parameters was based on the observed characteristics of web traffic. By identifying defining characteristics of web traffic as model parameters we were able to accurately model web traffic generated by a single user. During the analysis of measured parameter datasets we matched mathematical distributions to seven of the eleven model parameters. The reason for not being able to match distributions to all the parameters datasets was that our initial traffic model did not take into account the effects of *local caching*. The original model had only nine parameters. The omission of the effects of *local caching* resulted in us not being able to find matching distributions for two of the then nine model parameters. After investigating the problem we realised that we did not take into account the effects of *local caching* when defining the model. We altered the model by adding two more parameters which remedied the problem. Section 6.11 and 6.12 discuss the changes we made to the model. We found matching mathematical distributions for the eleven parameters in our final web workload model.

Chapter 3

Data Measurement

3.1 Introduction

We extracted **parameter datasets** for the eleven parameters in our workload model from traffic measurements. This chapter deals with the **measurement** of traffic on our campus network, the next chapter deals with the **extraction** of parameter datasets from the measurements taken. The activities of measurement (of traffic) and extraction (of parameter datasets), were closely related. The two activities were initially performed in **real-time**, in a single step. Due to performance reasons we split the measurement and extraction activities into two steps and extracted parameters **off-line**.

We explain why and how measurements were taken in Sections 3.2 and 3.3. We discuss how the data were measured in **real-time** in Section 3.4. The real-time approach failed. We processed the measurements **off-line**. In section 3.5 we discuss which information was recorded to secondary storage. We discuss the format of the measurement file that information was recorded to, and how the information in this file was processed. We conclude the chapter with Section 3.9 by discussing insights gained into the recording of packet traces and the processing of these traces.

3.2 Previous HTTP Measurements

We investigated the suitability of using publicly available network packet traces in our study. We found two repositories of publicly available traffic traces on the Internet:

- Internet Traffic Archive (ITA) :
  http://ita.ee.lbl.gov

- National Laboratory for Applied Network Research (NLANR) :
  http://moat.nlanr.net

An important requirement for traffic measurements we intended to use in our study was that they should include application level (HTTP) information. We required that application level information was included in traffic measurements to enable the accurate extraction of model parameter datasets from traffic measurements. Due to user **privacy concerns** the publicly available packet traces at the ITA and NLANR web sites did not include application level information. The packet traces found at the ITA and NLANR sites did therefore not fulfil the requirements of our study. We took traffic measurements ourselves.

The Information and Technology Services (ITS) department at the university gave us permission to record traces of web user traffic for a period of 30 days. The permission was granted provided that no information which encroached on the rights to privacy of web users would be stored to secondary storage, for any length of time.

Three **measurement techniques** had typically been used in the past to obtain data for the derivation of web workload traffic models. The techniques were *server logs* [AW96], *client logs* [CBC95, CP95], and *packet traces* [Mah97, Den96, RLGPC+99]. We give a short description of each of these methods:

Web server logs were used to record requests for documents on a web server. As the name indicates, data were recorded on the server.

Client logs were recorded by customising web browser software to write measurement files of required data. As the name indicates, data were recorded on the client.

Packet traces were obtained by recording data from a shared medium such as an Ethernet LAN. Data were recorded at a point between the web client and web server.

Web server logs were used to analyse the workload placed on a web server by web clients. Information about the users placing the requests were not recorded. Our study required information about user behaviour, such as when a user clicked on a hypertext link. The *web server log* approach therefore did not provide some information we required for our study.

The *client log* approach provided information about web user behaviour, such as when a user clicked on a hypertext link. This technique recorded all the information necessary for our study. It was however an enormous task to customise the web browsers used on campus to record client logs. The `Microsoft Explorer` browser was widely used around campus. The browser was proprietary and could not be customised. Given our resources and time constraints we could not use this method.

The *packet trace* approach did not record information about user behaviour. **Heuristic methods** could however be used to infer user behaviour from data recorded in packet traces. A very **large sample** of user datasets could easily be recorded by using packet traces. We chose the *packet trace* method to obtain web traffic measurements.

Measuring web traffic by means of packet traces generally consisted of three main steps: recording traffic at some point on a network, filtering and preprocessing the recorded data, and writing selected data in a specified format to secondary storage. We discuss two different approaches to recording packet traces next. We employed both of these approaches in our study.

3.3 Packet Traces

We considered two different approaches to obtaining parameter datasets by means of recording packet traces:

1. Processing data in **real-time** i.e. extracting parameter datasets from measurements at the same time as recording more measurements.

2. Processing data **off-line** i.e. recording selected data to secondary storage and extracting parameter datasets from the recorded data by processing the data off-line.

Approach No. 1 had the advantage of not recording **personal user information** to secondary storage. Parameter datasets were extracted in real-time and written to secondary storage. Parameter datasets did not contain personal user information. The ITS department preferred us to use the real-time method as it guaranteed users' privacy. We therefore implemented the real-time approach.

The real-time approach however failed due to **performance problems** arising from the concurrent processing and capturing of data. The real-time system used two processes, which ran on the same machine, to process and capture data respectively. For the system to succeed in capturing data, the *processing process* had to keep up with the *capturing process*. The *processing process* failed to keep up with the *capturing process*, and the system as a whole therefore failed.

Approach No. 2 did not suffer from the performance problems of Approach No. 1. Measured data were processed off-line. Approach No. 2 however recorded personal user information to secondary storage. This was a breach of the agreement we had with the ITS department. The agreement was that we would not record any personal user information to secondary storage. We reached a compromise with the ITS department which allowed us to process the data off-line provided that this was done on the measurement premises under the supervision of the ITS network administrator. Measurement files containing personal user information were deleted from the measurement machines before removing the equipment from the measurement premises.

We discuss our attempt to extract parameter datasets in **real-time** next, thereafter we discuss how we extracted parameter datasets from recorded traces by processing recorded data **off-line**.

3.4 Real-Time Processing of Packet Traces

We investigated the `tcpdump` and `Ethereal` network monitoring tools for use as real-time measurement tools in our study. The tools were very powerful and could perform specialised network packet measurements and packet analyses. We however needed an application that could process data in real-time by making use of a **heuristic algorithm** we developed. The heuristic algorithm enabled us to extract parameter datasets in real-time from measured data. The `tcpdump` and `Ethereal` tools could not be extended to implement the heuristic algorithm in real-time. We implemented our own network real-time measurement tool.

We implemented a **real-time measurement tool** in C using the Linux Socket Filter (LSF). The LSF is a packet filter which optimises packet filtering by performing filtering in **kernel space**. Figure 7 illustrates how the LSF filters selected packets in kernel space and passes these packets to **user space**, avoiding processing by the TCP/IP stack. The performance gain of using kernel space filtering enabled us to measure traffic on the 6Mbs campus network Internet link. The tool was implemented for both Intel and SPARC architectures i.e. big-endian and small-endian architectures.

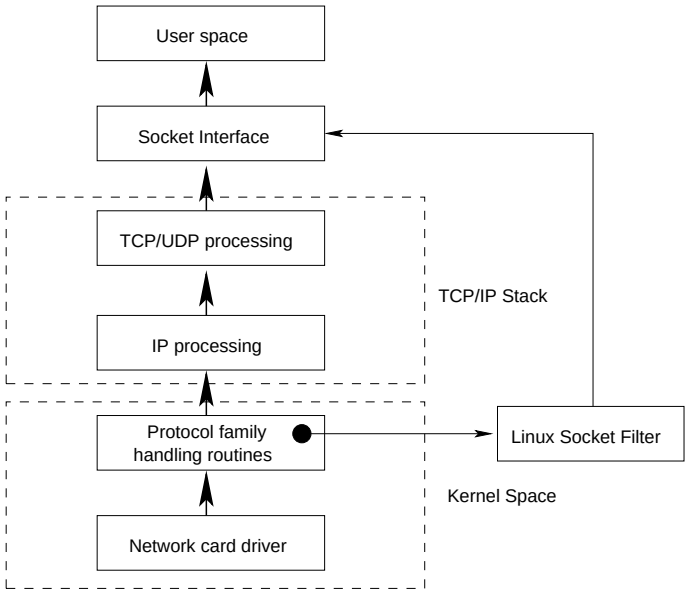


Figure 7: TCP/IP Packet Processing by the Linux Socket Filter

The real-time tool was deployed to monitor traffic between the campus network at university and the Internet. Web traffic between web users on campus and web servers on the Internet was recorded and processed by the tool. Parameter datasets for each host on the campus network i.e. for each machine which was used for browsing the Internet during the measurement period, were extracted from the recorded data.

We used a heuristic algorithm to extract parameter datasets from data generated by each host on campus. A heuristic algorithm was used because information necessary to extract parameter datasets was missing from the recorded data. The heuristic algorithm is discussed in Section 4.5 (page 56) The tool processed data for each host on the campus network in real-time whilst capturing more data. The real-time processing and capturing was achieved by keeping track of TCP/IP connection information for every host on campus browsing the Internet.

The real-time measurement tool is similar to the `Bi-Layer Tracing Tool` [Fel00]. The real-time measurement tool extracted data from HTTP, TCP and IP packet headers, processed the data and wrote parameter datasets to file for each host. Unlike the `Bi-Layer Tracing Tool` our real-time measurement tool did not process every TCP/IP packet monitored. The tool extracted sufficient

information to enable the extraction of parameter datasets from recorded data.

The real-time measurement tool filtered out selected web traffic packets in kernel space, to be used for further processing in user space. The following HTTP messages were filtered out:

- SYN, FIN or RST messages.
- GET requests.
- HTTP responses.

By considering these HTTP messages the tool was able to identify the start and end of TCP connections. The tool also extracted selected information from HTTP requests and responses. Table 4 shows which data were extracted from HTTP messages by the real-time measurement tool, and which headers the data were extracted from.

Data	**Protocol Header**	**Type of HTTP Message**
IP address of browsing host	IP	
TCP port of browsing host	TCP	
Requested URL	HTTP	Request
Referrer URL	HTTP	Request
Content-Length of Request	IP, TCP and HTTP	Request
Content-Length of Response	HTTP	Response
Content-Type field	HTTP	Response
Arrival Time		

Table 4: Data Extracted from IP, TCP and HTTP Headers by Measurement Tool

The `Type of HTTP Message` field in Table 4 shows which type of HTTP message the HTTP header was taken from. There were two types of HTTP messages, requests from client to server and responses from server to client. The `Arrival Time` data were measured by the system clock.

The advantages of extracting **selected information** from traffic transmitted between web clients and web servers were twofold. Firstly, the **complexity of processing** every TCP/IP packet was avoided. To process every TCP/IP packet would have amounted to implementing a system which had approximately the same functionality as the TCP/IP and HTTP stacks combined. Given our time and resource constraints this was not possible. Secondly, the **storage requirements** for storing entire TCP/IP conversations to secondary storage was avoided. We had limited storage capacity available, approximately 30GB. We recorded as little information as possible. Using the approach of extracting selected information from web traffic enabled us to measure and process traffic generated by the entire campus network over an **extended period of time**. We did not lose any data packets due to performance or storage issues.

The disadvantage of extracting selected information from web traffic was that information necessary to extract parameter datasets accurately was lost. In particular information about the size

of web requests and responses and the relationship between requests and responses were lost. We dealt with the lack of information by using a **heuristic algorithm** to extract parameter datasets.

In section 3.4.1 we explain how we extracted parameter datasets from data recorded by the real-time measurement tool.

3.4.1 Parameter Dataset Extraction

The real-time measurement tool used the information reported in Table 4 to extract datasets for the eleven model parameters listed in Table 3 (page 29) for each host on the campus network. As mentioned before Table 4 did not include sufficient information for the real-time measurement tool to extract parameter datasets for each host. We used a **heuristic algorithm** to extract parameter datasets.

The Virtual Memory File

The heuristic algorithm was based on a stored **history of HTTP messages** exchanged between a host on the campus network and web servers outside of the campus network. A history was recorded for every host on the network. The information extracted for each host is reported in Table 4. The histories recorded for hosts were written to a file in **virtual memory**.

The heuristic algorithm used to process the virtual memory file, accessed the data entries in the file in sequential order. The sequential access of data entries ensured that virtual memory management was optimal.

The operating system handled the virtual memory management for maintaining the memory mapped file. The virtual memory was allocated by means of the `mmap()` system call. We recorded the histories of all hosts on campus in a virtual memory area of 2GB.

Parameter datasets were extracted for each host by processing the history of browsing activity for that host as stored in the memory mapped file. The history for each host was kept in a **linked list of data entries**. When inserting a new entry into the mapped file, the previous entry for that host was linked to the new entry by means of a pointer. The result was a linked list of data entries for every host with a recorded history in the mapped file. The history of browsing activity for a particular host was processed by traversing the linked list of data entries.

The memory mapped file was split into **two halves**. Extracted information was written to one half while the other half was being processed by the real-time measurement tool i.e. parameter datasets were extracted from the data by applying the heuristic algorithm to the data. Figure 8 illustrates how the real-time measurement tool processed data recorded in one half of the memory mapped file whilst capturing new information to the other half of the file.

Figure 8 shows how the **parent process** captured, filtered and wrote data to the memory mapped file. A **child process** was spawned by the parent after it had filled one half of the mapped file with data. The child process then proceeded to process the recorded information, writing extracted parameter datasets to secondary storage, and killed itself after processing was finished. The child process processed data recorded in the memory mapped file by iterating through the linked list

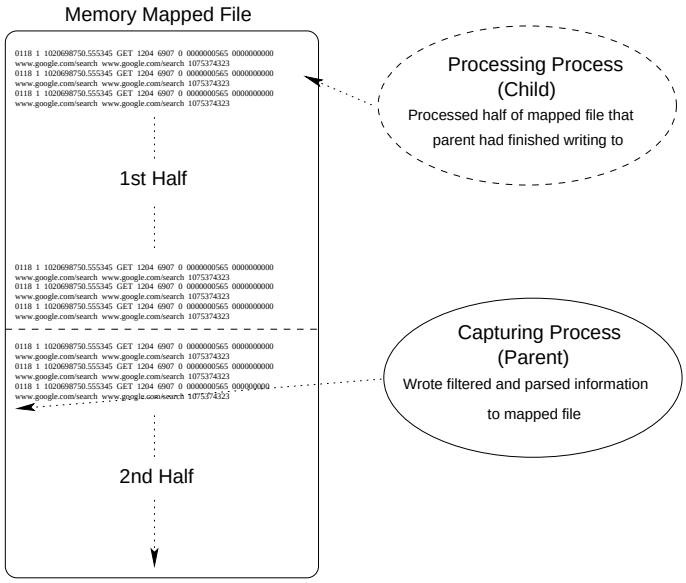


Figure 8: Real Time Processing of Captured Information by Measurement System

of data entries for each host, processing the data for that host, and writing extracted parameter datasets to file.

Architecture of Real-Time Measurement Tool

The real-time measurement tool could run for indefinite periods of time, extracting parameter datasets whilst capturing data in real-time. The processing process had to finish processing its half faster than the capturing process took to fill its half with data. In the case of processing taking longer than capturing data, the program exited with an error status code. Figure 9 illustrates how the real-time measurement tool captured and extracted parameter datasets in real-time.

The lowest layer illustrated in Figure 9 shows that the measurement tool used the LSF to filter out HTTP traffic sent to or received from port 8080 from the traffic stream on the network wire. The reason for extracting traffic sent to or received from port 8080 was that the two web cache proxy servers on the campus network used port 8080 to connect to hosts on the campus network. All web traffic between hosts on the campus network and web servers outside of the campus network went through these two web cache proxy servers. The LSF therefore extracted all traffic between hosts on campus and web servers on the Internet by filtering out traffic sent to or received from port 8080. The extracted traffic was passed to user space by the LSF. Processing time was saved by avoiding the processing of irrelevant packets by the TCP/IP stack.

Data passed to `Layer 1` by the LSF was processed at `Layer 1`. Selected information was written to memory mapped file. Selected data for a host were added to the linked list of data entries for that host in the memory mapped file. A new data entry for a host was added to the end of the linked

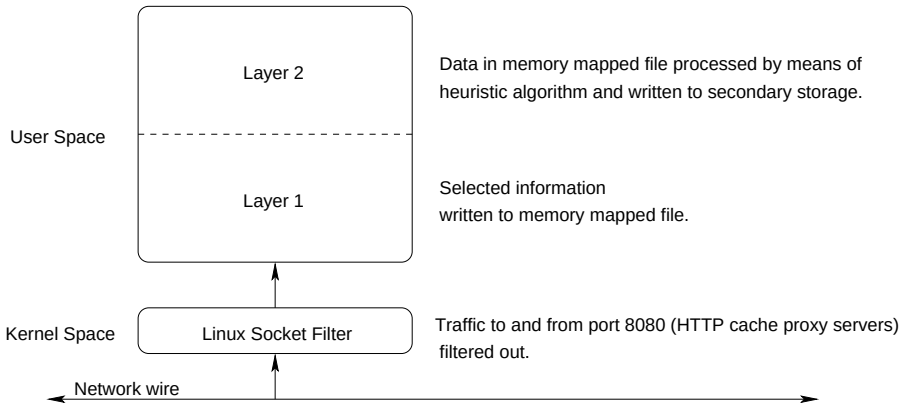


Figure 9: Overview of the Filtering, Parsing and Processing of HTTP Messages by the Measurement Tool in Real-Time

list for that host. Pointers to the last node for the linked list of every host were stored in a table indexed by the IP address of a host. The array had 65 536 entries as the university had a Class B IP address space.

Data were processed at `Layer 2` and parameter datasets extracted for each host. There were several problems which had to be solved in order to extract parameter datasets for each host. The most difficult problem we faced was to **differentiate** between web client and web user requests i.e. requests that were generated by web users as opposed to requests that were generated by web client software e.g. Microsoft Explorer. We used a heuristic algorithm to solve this problem. This problem and our solution to it is discussed next. Another problem that had to be solved was that of **matching** pairs of HTTP request and response messages to one another. The information contained in data captured from the network wire did not include information which matched an HTTP response message to its relevant request message. The problem of matching HTTP request and response messages, and our solution to it, is discussed after the next section.

The Web User vs. Web Client Request Differentiation Problem

The web user vs. web client request differentiation problem can be stated as follows:

Was a request a web user or web client request? I.e. did a web user or web client generate a request?

The answer to this question was very important. The successful extraction of parameter datasets from the measured data depended on finding as accurate an answer as possible. It was not possible to obtain a definite answer to the question. In order to answer the question definitively information about a web user's behaviour was required. Data obtained by monitoring a network link did not contain information about a user's behaviour.

We had to **infer user behaviour** by using the information at our disposal. By using the information at our disposal we compiled a **list of web client request characteristics** e.g. we

observed that a web client request was usually a request for an image file.

We used the list of characteristics we compiled to categorise a new request as being either a web client, or a web user request. We checked whether a new request's characteristics matched any of the characteristics in the list we compiled. If a match was found, the request was categorised as a web client request, and if a match was not found it was categorised as a web user request.

We discuss this algorithm and its implementation in detail in Section 4.5 (page 56). We also used the algorithm during the off-line processing of packet traces which is discussed in Section 4.5.

The HTTP Request/Response Matching Problem

The real-time measurement tool did not not record every TCP/IP packet transmitted between clients and servers. The data captured by the tool did not contain information which associated HTTP requests with their corresponding HTTP response messages.

The HTTP request/response matching problem can be stated as follows:

How did we match HTTP response messages to the HTTP request message which caused them?

We used our knowledge about how the HTTP placed requests on TCP connections to solve the problem. HTTP requests were **pipelined** on a TCP connection i.e. multiple requests were made without waiting for responses to return. Responses returned in the same order as requests were made.

We were able to match responses to requests by keeping track of all requests made on all TCP connections. Responses were matched to requests by matching returning responses to requests stored in a **queue** for a specific TCP connections. A queue data structure was maintained for every TCP connection. New requests were added to the tail of the queue, and requests were removed from the head of the queue when an incoming response was received.

Queues storing requests could be implemented by using either static or dynamic memory allocation. Static memory allocation required more space but was faster. The processing process had to **perform optimally**. The processing of one half of the mapped file had to be completed faster than the time it took to capture data to the other half of the mapped file. We used **static memory allocation** in order to speed up the processing of the memory mapped file.

In order to statically allocate memory, we calculated the number of queues necessary to keep track of information transmitted on TCP connections. In calculating the number of queues we assumed the worst case scenario of a host having 10 browsers open each using 5 TCP connections at once. The worst case scenario was very unlikely to occur in practice. Using the worst case scenario formula we allocated memory for 50 queues. The maximum length of a queue was calculated as the maximum number of consecutive requests that could be made without any responses returning. We again used the worst case scenario of 50 requests being made without responses returning. The worst case scenario was very unlikely to occur in practice.

We implemented a two-dimensional array which stored 50 queues each of length 50. Figure 10 illustrates the data structure.

Each queue started at position 0 in the array. Elements were added to the tail of the queue,

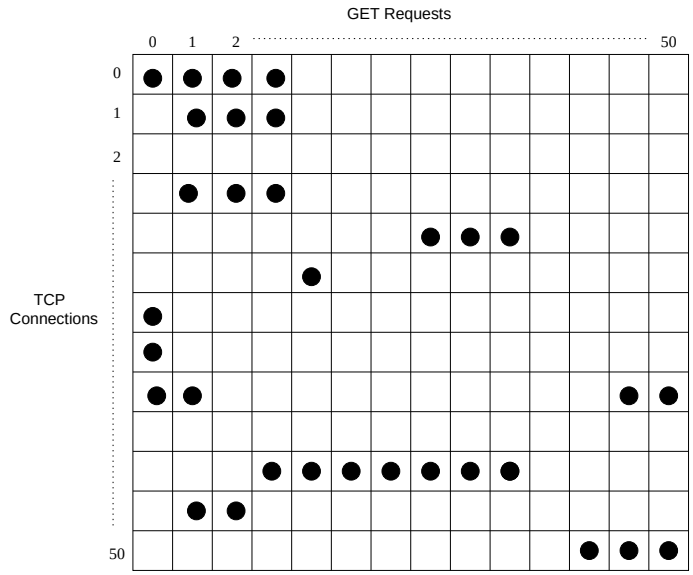


Figure 10: Multi-Dimensional Array used by Measurement Tool for Keeping Track of GET requests on TCP ports

and removed from the head of the queue. Data stored in the queue moved from left to right across the data structure illustrated in Figure 10 eventually rolling over to the start of the structure again. The structure had been implemented to roll over to **optimise performance**. The more intuitive implementation of copying the whole queue one space across every-time an element was removed would have imposed an unnecessary performance penalty on the tool.

We created a mapping from a TCP connection (i.e. TCP port number) to the index in the array for that connection. The mapping was stored in a two-dimensional array. Figure 11 illustrates the mapping stored in the two-dimensional array.

TCP port numbers were inserted in order into the mapping array. A binary search was performed on the mapping array to find the index in the GET request array for a specific TCP port number. The GET request array therefore needed to be sorted. The GET request array was compacted when it became full. The compaction method removed all entries which did not have any pending GET requests.

The queue data structures we implemented kept track of all GET requests placed on all TCP connections used by a host. The information about GET requests stored in the queue data structures enabled us to match HTTP requests to responses.

The real-time measurement tool succeeded in extracting parameter datasets in real-time for hosts during off-peak traffic times i.e. during the evening. The tool however **failed** to extract parameter datasets in real-time during the day. The process processing data could not keep up with the process capturing data during peak traffic times. We therefore abandoned the real-time approach and implemented programs to process recorded information off-line. We discuss the off-line

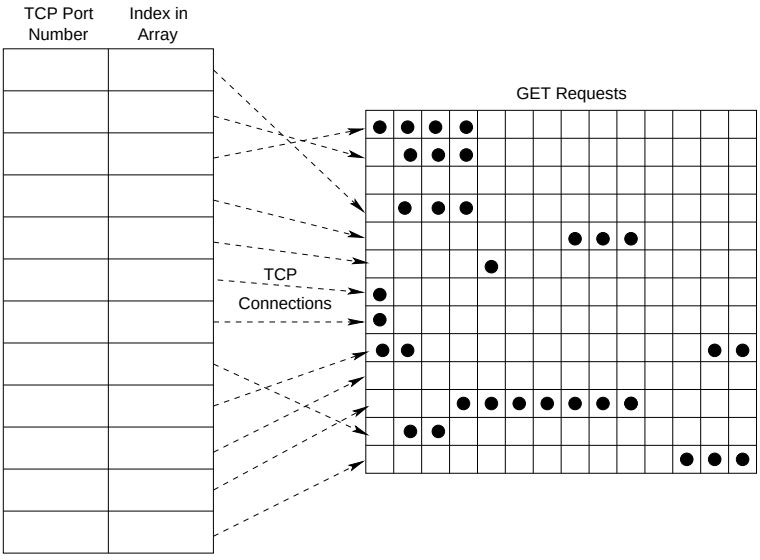


Figure 11: Mapping used by Measurement Tool to Map a TCP Port to an Array Index

processing programs next.

3.5 Off-line Processing of Packet Traces

We failed to extract parameter datasets in real-time. The **hard time constraints** on the real-time processing of measured data were not met when the tool was used during peak traffic hours. Off-line extraction of parameter datasets did however not impose any hard time constraints on the processing of measured data. When extracting parameter datasets off-line, the extraction process did not have to keep up with the flow of data on the network. The performance of the measurement tool was no longer of crucial importance. We decided to extract parameter datasets off-line. In order to achieve off-line processing of data, we had to record information to secondary storage. Personal user information were recorded to secondary storage in the process.

Recording personal user information to secondary storage was a breach of our agreement with the ITS department. After deliberation with the ITS department, they granted us permission to take the measurements under the condition that parameter datasets were extracted from measured data under the supervision of an ITS network administrator. The measured data containing personal information were deleted from the measurement system before the system was removed from the ITS department's premises.

The off-line measurement tool was very similar to the real-time measurement tool. The difference between the off-line measurement tool and the real-time measurement tool was that the real-time measurement tool processed measured traffic, whereas the off-line measurement tool did not. The off-line measurement tool wrote measured network data to a measurement file on secondary storage.

The measurement file created by the off-line measurement tool was processed off-line by another tool we will discuss in Section 4.3.

The data measured by the real-time measurement tool were the same as data measured by the off-line measurement tool. The functionality of the two tools in terms of data measurement were very similar. The difference between the tools was that the real-time tool wrote data to a memory mapped file whereas the off-line tool wrote data to secondary storage.

The off-line measurement tool captured TCP/IP packets transmitted on the Ethernet segment to which the university campus' web cache proxy servers were attached. It intercepted traffic sent to or sent by the proxy machines. As we previously discussed, the first level of data extraction was performed by the LSF. The LSF extracted web traffic between the campus network and web servers on the Internet. Extracted TCP/IP packets were passed to user space as illustrated in Figure 7 (page 34). We next discuss how the off-line measurement tool was deployed on the campus network in order to capture web traffic between hosts on campus and web servers on the Internet in Section 3.6. We then discuss which information was extracted by the off-line measurement tool and how the extracted information was written to measurement file in Section 3.7.

3.6 Measurement Strategy

We instrumented two machines with the data off-line measurement tool and placed them at the **trace collection points** on the campus network as illustrated in Figure 12. The position of the machines on the network ensured that all HTTP requests from and responses to university students and staff browsing the web were captured. We captured traffic generated by 6 689 hosts over a 1 month long period.

The measurement machines were synchronised by means of the Network Time Protocol (NTP) [Mil92]. Synchronisation was necessary as data captured by the machines were merged into one dataset. The resultant dataset was used for analysis. The analysis required accuracy of a hundred milliseconds. The machines' clocks were synchronised to within ten milliseconds of each another.

The `Web Client Request Inter-arrival Time` parameter as shown in Table 3 (page 29) was measured in microseconds which would have been problematic during the merging of datasets as data were not measured at microsecond accuracy. Fortunately `Web Client Requests` generated by the same `Web User Request` were all sent to the same web proxy machine. The merging of datasets was therefore not a problem as accuracy was not affected by merging datasets.

As mentioned in Section 2.6 (page 27), the `Web User Request Inter-arrival Time` and `Web Client Request Inter-arrival Time` were the only inter-arrival time parameters we modelled. These parameters model the time between web user clicks and web client initiated requests for graphics files respectively. They were not intrinsically linked to network behaviour, and if in fact we had been able to instrument all the web browsers at the university, we would have been able to measure these parameters without any influence by network conditions. As Figure 12 however shows, our measurement machines were placed on the network. This meant that the time it took for a

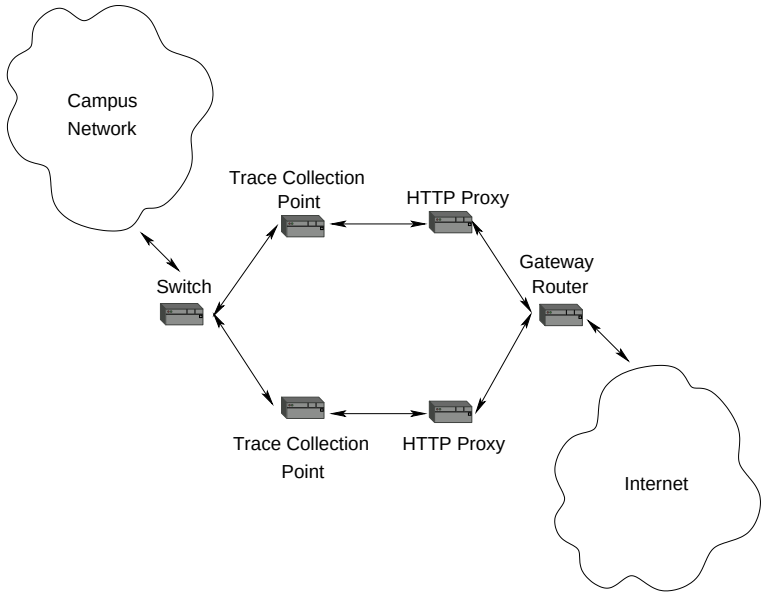


Figure 12: Deployment of Measurement Tool on Campus Network

packet to travel from a web user's host on the university campus network to the measurement machine would have influenced the measurement of these parameters. The university campus network had a fibre optic backbone network with capacity of 100Mbs. The utilisation on this network was very low, between 15% and 20% on average, and very seldomly higher than 25%, which resulted in low latency and variability of latency between users on the campus network and the measurement machines. We therefore measured data for these parameters at the **trace collection points**.

3.7 Information Extracted from HTTP, TCP and IP Packets

In order to solve the HTTP request/response matching problem we kept track of the start and end of **all TCP connections** for each host. We accomplished this by recording information contained in TCP/IP packets with their SYN, FIN or RST flags set. The fields shown in Table 7 (page 48) were extracted from these packets and written to measurement file.

We captured all SYN packets sent from a host on campus to a web server and all FIN packets sent from a web server to a host on campus. By capturing these packets the start and end of a TCP connection as indicated by means of SYN and FIN flags were recorded. We were certain of capturing the start of a TCP connection by recording SYN packets travelling in one direction only as TCP's three way handshake mechanism ensured that the SYN flag is set for the first two packets transmitted in opposite directions when a new TCP connection is opened. Capturing either of the packets was sufficient for us. When closing a TCP connection by setting the FIN flag of a TCP/IP packet the last two packets transmitted in opposite directions on the TCP connection have their

FIN flags set. We captured one of these.

We recorded RST packets sent by a host on campus to a web server as well as RST packets sent by a web server to a host on campus. A single packet with its RST flag set can be sent to abort a connection in either direction on a TCP connection. Recording all RST packets ensured that the end of a TCP connection as indicated by an RST flag was recorded.

All other packets captured by the off-line measurement tool contained HTTP messages. The LSF extracted HTTP messages and passed them to the off-line measurement tool in user space. We inspected the headers of these HTTP messages and extracted the information in Tables 5 and 6 (page 46).

We used **regular expressions** to parse and extract selected information from HTTP header fields. The C `regex` library, which is compliant with the `POSIX.2 interface` for regular expressions, was used to match HTTP header fields to regular expressions. Figure 13 shows the regular expressions we used.

```
Request Field:
^GET http://([^/]+)/([^[:space:]]*) HTTP/[0-9]+.([0-9]+)[[:space:]]*$
Response Field:
^HTTP/[0-9]+.([0-9]+) ([0-9]*) .*$
If Modified Since Field:
^If-Modified-Since:.*length=([0-9]+)[[:space:]]*$
Content Length Field:
^Content-Length:  ([0-9]+)[[:space:]]*$
Referrer Field:
^Referrer:  http://([^/]+)/([^[:space:]]*)[[:space:]]*$
Content Type Field:
^Content-Type:  ([^[:space:]]+)[[:space:]]*$
Transfer Encoding Field:
^Transfer-Encoding:  ([^[:space:]]+)[[:space:]]*$
```

Figure 13: Regular Expressions used by Measurement Tool to Parse HTTP Message Headers and to Extract Selected Information

The first two regular expressions shown in Figure 13 were used to parse `request` and `response fields` contained in HTTP request and response messages respectively (request and response fields are defined in RFC 2616 [BLFea99]). Round braces in the regular expressions indicate which parts of a field were extracted. The `host part of request URL`, `path part of request URL` and `HTTP version` fields were extracted from the request field and are shown in Figure 5 (page 46). The `HTTP version` and `status code` fields were extracted from the response field and are shown in Figure 6 (page 47).

We measured the sizes of HTTP request and response messages in bytes. The size of an HTTP message was taken to be the size of the HTTP header added to the size of the body of the HTTP message. The sizes of messages were not always easy to calculate from information contained in HTTP request, HTTP response, TCP and IP header fields. We calculated the size of HTTP request and response messages as follows.

We assumed that HTTP request messages did not span more than one TCP/IP packet. Observation of HTTP traffic traces showed this to be the case for nearly all HTTP request messages we captured. Under this assumption an HTTP request message was transmitted in a single TCP/IP packet. The size of an HTTP request message was calculated as the size of the payload of a TCP/IP packet that contained an HTTP request message. The size of the payload of a TCP/IP packet was calculated by subtracting the size of the combined Ethernet, TCP and IP headers from the size of the data message that was passed to the measurement tool by the LSF i.e. the size of the whole packet as read from the network link. The sizes of the TCP and IP headers were extracted from the relevant fields within TCP and IP headers.

The size of an HTTP response message was more complicated to calculate because response messages typically spanned several packets. We calculated the size of an HTTP response message by calculating the size of the message header and body separately and then adding the sizes together.

The control sequence `"CRLFCRLF"` designated the end of an HTTP message header contained in a TCP packet's payload [BLFF95, BLFea97, BLFea99]. We calculated the size of the HTTP response message header by finding the control sequence in the payload data stream. The amount of bytes contained in the data stream before the appearance of the control sequence was recorded as the size of the HTTP response message header.

The size of the HTTP response message body was contained in the `content-length` field of the HTTP message header. Some HTTP message headers did not contain the `content-length` field. We had no means other than the `content-length` field to obtain the size of an HTTP message body. We therefore could not record a size value for HTTP response messages which did not have a `content-length` field, even though we knew what the size of the HTTP headers for these messages were.

Some HTTP response messages did not have a message body, these were response messages with the following `status codes`: 1xx (informational), 204 (no content) and 304 (not modified) [BLFF95, BLFea97, BLFea99]. These messages were recorded to have message bodies of size zero bytes. The size of a HTTP response message was calculated by adding the size of the HTTP header to the size of the HTTP message body.

We next discuss how information extracted from HTTP, TCP and IP packets was recorded to measurement file.

3.7.1 Measurement File Format

Information extracted from HTTP, TCP and IP packets was recorded to a measurement file. The off-line measurement tool recorded data in space delimited text format. Three different types of entries were used to record data.

1. HTTP request message entry.
2. HTTP response message entry.
3. SYN, FIN and RST message entry.

Figure 14 shows examples of the three different types of measurement file entries. We show separate examples for `SYN`, `FIN` and `RST` messages even though they belong to the same entry type.

```
SYN Message Entry:
1030643953.416477 SYN d56d 5806
HTTP Request Message Entry:
1030643954.003347 GET d56d 5806 0 0000000428 0000000000 www.mweb.co.za
  home/images/welcome1.gif www.mweb.co.za home/default.asp
HTTP Response Message Entry:
1030643956.020927 200 d56d 5806 0 0000001296 image/gif
FIN Message Entry:
1030643957.630441 FIN d565 5806
RST Message Entry:
1030643957.630441 RST d565 5906
```

Figure 14: Examples of the Three Different Types of Measurement File Entries Recorded in Space Delimited Text Format

The processing routines and heuristic algorithm described in Chapter 4 used the data contained in the measurement file to extract datasets for the eleven model parameters.

We next discuss each of the three different types of measurement file entries in turn. We give a short description for each field contained in a measurement file entry.

Table 5 shows the fields contained in an `HTTP request message entry`.

Field No.	**Name**	**Sample Entry Field**
1	Arrival Time	1030643954.003347
2	Request Type	GET
3	Host Number (hexadecimal)	d56d
4	Port Number (hexadecimal)	5806
5	HTTP Version	0
6	Message Size	0000000428
7	If-Modified-Since Size	0000000000
8	Host Part of Request URL	www.mweb.co.za
9	Path Part of Request URL	home/images/welcome1.gif
10	Host Part of Referrer URL	www.mweb.co.za
11	Path Part of Referrer URL	home/default.asp

Table 5: Fields Contained in an HTTP Request Message Entry

A short description for each of the `HTTP request message entry` fields follows:

No. 1: `Arrival Time`, was measured as the number of seconds since the beginning of the epoch (1970), with microsecond accuracy.

No. 2: `Request Type`, could be one of the following: OPTIONS, GET, HEAD, POST, PUT,

DELETE, TRACE, CONNECT. The GET request type accounted for by far the most traffic. We only considered GET request entries.

No. 3: `Host Number`, is a number which uniquely identifies the host which generated the request. The number is the last 16 bits of the 32 bit IP address of a host, in hexadecimal. The university had a Class B IP address space i.e. 16 bits was used to represent the host part of an IP address.

No. 4: `Port Number`, is the number of the communication port the request was generated from, in hexadecimal.

No. 5: `HTTP Version`, is the version of HTTP used by the web client which placed the request. It had a value of 0 or 1 representing either HTTP Version 1.0 or 1.1.

No. 6: `Message Size`, is the size of the request message in bytes.

No. 7: `If-Modified-Since Size`, was used to determine whether a file had been changed since the last time the file was accessed. If a file had been changed, it was returned by the server to the client, otherwise not.

No. 8: `Host Part of Request URL`, is the dot-delimited host-name part of the requested URL.

No. 9: `Path Part of Request URL`, is the path and filename part of the requested URL.

No. 10: `Host Part of Referrer URL`, is the dot-delimited host-name part of the `referrer` URL. The `referrer` URL indicated which URL was responsible for the current request i.e. which URL the current request was placed from.

No. 11: `Path Part of Referrer URL`, is the path and filename part of the referrer URL.

Table 6 shows the fields contained in a `HTTP response message entry`.

Field No.	**Name**	**Sample Entry Field**
1	Arrival Time	1030643954.020927
2	Status Code	200
3	Host Number (hexadecimal)	d56d
4	Port Number (hexadecimal)	5806
5	HTTP Version	0
6	Message Size	0000001296
7	Content Type	image/gif

Table 6: Fields Contained in an HTTP Response Message Entry

A short description for each of the `HTTP response message entry` fields follows:

No. 1: `Arrival Time`, is the time of arrival of a response.

No. 2: `Status Code`, is a 3-digit integer code indicating the status of the response, in terms of whether it had satisfied the request associated with it.

No. 3: `Host Number`, is a number which uniquely identified the host to which the response was sent.

No. 4: `Port Number`, is the number of the communication port to which the response was sent.

No. 5: `HTTP Version`, indicates the version of HTTP used by the web client to which the response was sent.

No. 6: `Message Size`, is the size of the response message in bytes.

No. 7: `Content Type`, indicates the type of media contained in the response.

Table 7 shows the fields contained in a `SYN, FIN and RST message entry`.

Field No.	**Name**	**Sample Entry Field**
1	Arrival Time	1030643953.416477
2	Message Type	SYN
3	Host Number (hexadecimal)	d56d
4	Port Number (hexadecimal)	5806

Table 7: Fields Contained in a SYN, FIN and RST Message Entry

A short description for each of the `SYN, FIN and RST message entry` fields follow:

No. 1: `Arrival Time`, is the time of arrival of a SYN, FIN or RST message.

No. 2: `Message Type`, had a value of either SYN, FIN or RST. We defined a message to be a SYN, FIN or RST message type if the respective flag was set in the TCP header. SYN messages indicated that a TCP connection was being opened, FIN messages that a TCP connection was being closed, and RST messages that a TCP connection was being closed abnormally.

No. 3: `Host Number`, had different meanings depending on the `Message Type`:

- The host which sent a SYN message.
- The host which was to receive a FIN message.
- The host which either sent or received a RST message, depending on which one the web client was.

No. 4: `Port Number`, is the number of the communication port to which the message was sent.

3.8 Data Measurement

We used the off-line measurement tool to capture data on the campus network for a period of 30 days. We captured web traffic generated by 6 692 hosts during the measurement period. The data were captured to measurement file in the format described in the previous section. Table 8 summarises information about the data measurement.

Description	Information
Measurement start date	2002-08-20 11:58:54 +0200
Measurement finish date	2002-09-19 21:01:18 +0200
Measurement duration	30 days
No. of hosts recorded	6 692

Table 8: Details of Data Measurement on Campus Network

The data were recorded in two measurement files because the measurement machines were placed in two locations on the campus network as illustrated in Figure 12 (page 43). The two files had a combined size of 25GB.

3.9 Concluding Remarks

The information obtained through the packet trace of web traffic was not sufficient for extracting data about **user behaviour**. We used a complicated **heuristic algorithm** to **infer user behaviour** from information recorded in the packet trace. The heuristic algorithm used information such as the **type of data requested** by a user, or the **time between successive requests** to infer user behaviour.

The heuristic algorithm was implemented in a measurement tool. The algorithm used a **recorded history** of data transmitted between a web client and web servers. The measurement system failed to extract data in **real-time** due to the complexity of the heuristic algorithm. Although the system was able to extract data in real-time during off-peak web browsing time, it could not extract data during peak web browsing time.

The **real-time measurement tool** consisted of two processes which were synchronised. The tool was therefore well suited to being **distributed** over two or more processors. We believe that the real-time measurement tool would succeed in extracting data during peak time if optimised by distributing it over more than one processor. The distribution of processing over more than one processor was left for future work.

We implemented an off-line measurement tool which extracted **selected information** from data transferred between web clients and servers. The data recorded by the off-line measurement tool contained sufficient information to extract datasets for the parameters of our web browsing traffic model. We extracted user behaviour data off-line from the web traffic packet traces.

The off-line measurement tool was **novel** as it did not record all data transferred during a TCP/IP connection between two hosts. Similar studies to ours used measurement tools which recorded all TCP/IP data transferred between hosts [Fel00]. Recording whole TCP/IP connections required **enormous storage space and/or processing power**. We did not have either of these resources and therefore implemented an off-line measurement tool which extracted only essential information from data transmitted between web clients and servers. The data recorded by the off-line measurement tool were used to extract datasets for the eleven parameter datasets of our web workload model. We were able to record data for a 30 day period, capturing web traffic generated by 6 692 hosts on a campus network. The resultant measurement file had a size of 25GB. A storage requirement of 960GB of storage space would have been required if entire TCP/IP conversations were recorded.

Chapter 4

Data Processing

4.1 Introduction

As discussed previously, we captured traffic to measurement file. The data captured was processed off-line. The data measurement resulted in two measurement files with a combined size of 25GB. We **merged** the two measurement files into a single file. The merged file was **split** into 6692 files, one for each host whose traffic was recorded during measurement. We processed the host datafiles and extracted **parameter datasets** from the host datafiles. Eleven parameter datasets were extracted from each of the host datafiles. The data processing resulted in 73612 parameter datasets which we analysed.

Our web workload model characterised user behaviour. Information contained in host datafiles did not contain information about user behaviour. Extracting parameter datasets from host datafiles was therefore a complicated process. We inferred user behaviour from information contained in host datafiles. We differentiated between requests for web pages which were generated by web users, and requests which were generated by web clients i.e. we differentiated between when users clicked on hypertext links, and when web clients requested inline objects.

Differentiating between web user and web client requests was the main problem we solved during the processing of data. The solution to the **web user vs. web client differentiation problem** enabled us to extract parameter datasets from host datafiles. We used a **heuristic algorithm** to differentiate between web user and web client requests.

The web workload model we defined in Chapter 2 was based on the **traditional web browsing model**. The traditional web browsing model can be explained as follows: A web page is requested by a web client. An HTML file is returned by a web server in reply to the request. The HTML file is received and parsed by the web client. After parsing the HTML file the web client places requests for inline objects which are returned to the web client by the web server.

The measurement file contained data which were not generated according to the traditional web browsing model. For example, the file contained data generated by applications such as web-spiders

and the Microsoft Windows web update tool. Web-spiders and the Microsoft Windows update tool generated traffic which had different characteristics to that of traffic generated according to the traditional web browsing model. We **removed** data that were not generated according to the traditional web browsing model.

We discuss the problems we solved in Section 4.3.1, and the heuristic algorithm in Section 4.5. We discuss the removal of this data in Section 4.7.

4.2 Splitting of Measurement File

As discussed previously, the measurement of web traffic resulted in two datafiles with a combined size of 25GB. We combined the two datafiles by using the Linux `cat` utility. We sorted the file with the Linux `sort` tool written by Mike Haertel and Paul Eggert. The tool used a combination of the Quicksort and Mergesort algorithms. The datafile was sorted on the `Host Number` and `Arrival time` fields as shown in Tables 5 - 7 (page 46). The command we used for sorting the data is shown in Figure 15. The sort was performed in place i.e. the resultant sorted file was copied over the original, and required an extra 25GB of secondary storage for swap space. The total amount of storage space required by the sorting process was therefore 50GB. The sort took 9 hours to complete.

```
nohup sort -k 3,3d -k 1,1n -T /mnt/swapfile -o sorted_file
```

Figure 15: Command Used to Sort Measurement Datafile

The sorted datafile was split into a separate datafile for each host. We implemented a tool which split the sorted datafile into host datafiles. The tool processed datafile entries in `O(N)` time.

The largest addressable file size for a 32 bit file-pointer was approximately 2.1GB. The sorted datafile had a size of 25GB. 32 bit file-pointers were not large enough to access the whole of the sorted datafile. We enabled 64 bit file-pointers during the compilation of the tool to overcome this problem. The special `gcc` command line parameters shown in Figure 16 was used to enable 64 bit file-pointers.

```
-D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE
```

Figure 16: Command line Parameters to Enable 64 bit File-pointers for `gcc`

We extracted nine parameter datasets from each of the 6 692 host datafiles. We discuss this process next.

4.3 Extraction of Parameter Datasets

The methods used by the real-time measurement tool to extract parameter datasets, as discussed in the previous chapter, was adapted to extract parameter datasets off-line. We discuss problems we solved in order to extract parameter datasets off-line next.

4.3.1 Extraction Problems

As mentioned before, the main problem with extracting parameter datasets from data measured by packet traces was that information related to user behaviour was not recorded. For example, information about when a user clicked on a hypertext-link or entered an URL in the address text-box of their web-browser were not recorded in the packet trace.

Figure 17 illustrates the measurement setup and resultant shortcomings of data captured to packet traces.

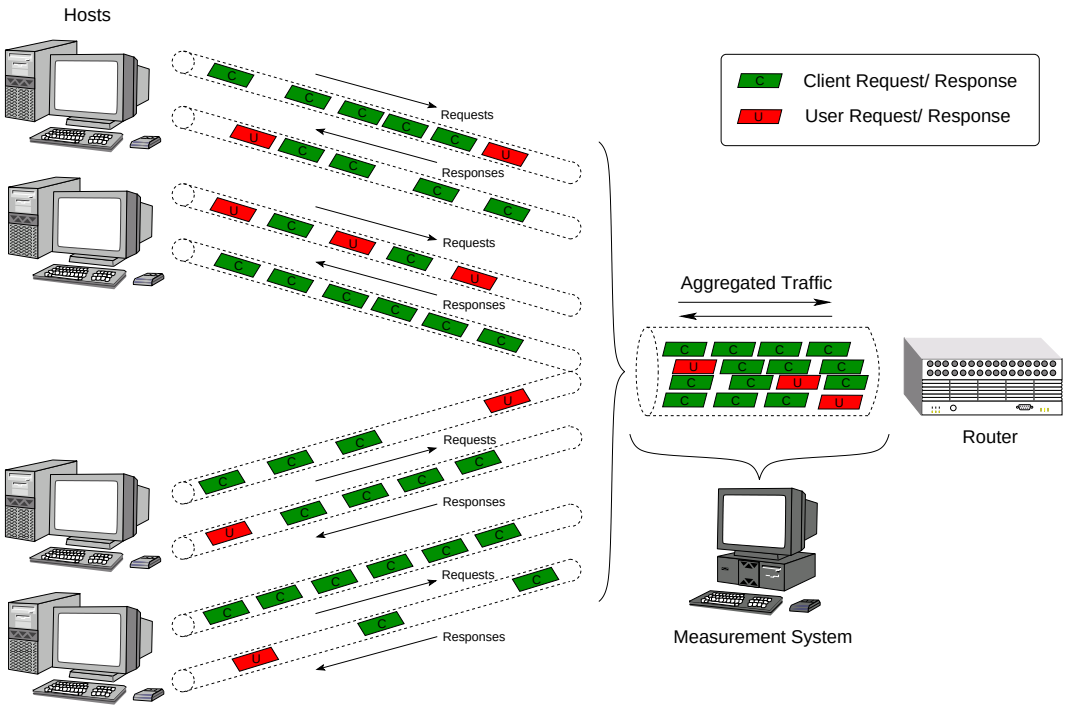


Figure 17: Measurement Setup and Resultant Shortcomings of Measured Data

Data extracted by the measurement system as shown in Figure 17 are shown in Tables 5 - 7 (page 46).

In Figure 17 web user requests are indicated by red boxes and web client requests by green boxes. The two types of requests were indistinguishable to the measurement system. The first problem we solved was:

The Web User vs. Web Client Request Differentiation Problem: Was a request a web user or a web client request? I.e. did a person or a web client generate the request? This was the most important and difficult problem we solved. The answer to this problem was used to extract data for parameter datasets. For instance, in order to determine when a new web browsing session started, it was necessary to know whether a request was generated by a web user or a web client.

In Figure 17 a series of web client requests (green) always follow after a particular web user request (red). A web user request is always responsible for a series of web client requests to be placed by a web client. The measurement system did not record information which relates web user requests to web client requests. The second problem we solved was:

The Web Client Request Matching Problem: Which web user request was responsible for a series of web client requests to be placed? The extraction of the `Web Client Request Inter arrival Time` and `Number of Web Client Requests per Web User Request` parameter datasets depended on a solution to the web client request matching problem.

In Figure 17, every HTTP request message as indicated by a message on the `request pipe` of a host, has a corresponding HTTP response message as indicated by a message on the `response pipe` of that host. The measurement system did not record information which associated HTTP request messages with their corresponding response messages. The reason for the measurement system not recording this information was that whole TCP/IP conversations were not recorded, as we previously discussed. The third problem we solved was:

The HTTP Request/Response Matching Problem: Which HTTP response message belonged to which HTTP request message? We used the solution to this problem to extract parameter datasets from host datafiles. The information also enabled us to dispose of data associated with TCP connections which terminated abnormally.

We discuss our solution to the HTTP request/response matching problem in Section 4.4 and our solution to the web user vs. web client request differentiation problem in Section 4.5.

4.4 The HTTP Request/Response Matching Problem

We discussed our solution to the HTTP request/response matching problem for the real-time measurement tool in Section 3.4.1 (page 39). Our solution to the problem for the off-line processing tool was very similar to the solution of the problem for the real-time measurement tool. The difference between the two solutions was that the off-line processing tool was not optimised for performance, whereas the real-time processing tool was optimised for performance. Because performance was not a crucial issue, it was not necessary to use complex data structures for statically allocated queues to store HTTP requests in the off-line processing tool.

Figure 18 illustrates how we kept track of HTTP requests and responses on a single TCP connection by using a queue data structure. We matched HTTP responses to HTTP requests by keeping track of all HTTP requests made on all TCP connections, as we did for the real-time measurement tool.

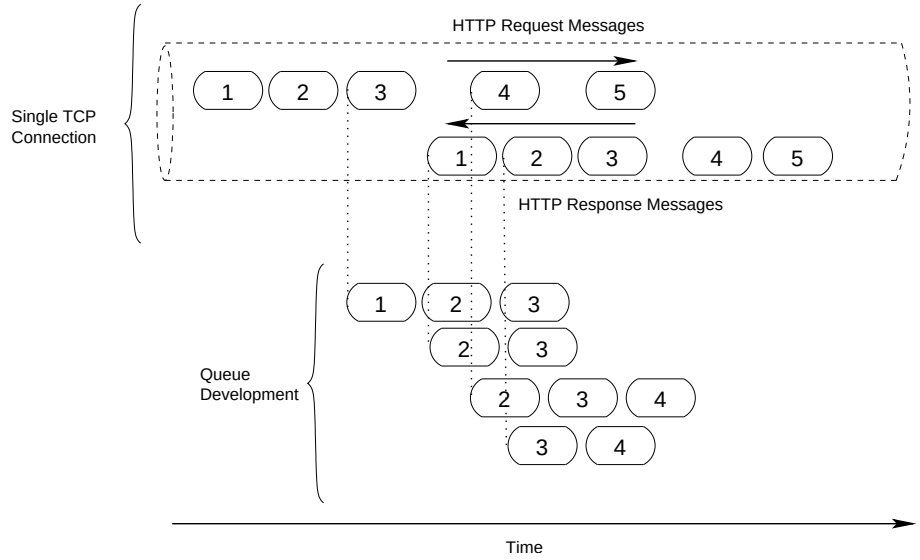


Figure 18: HTTP Request and Response Matching on a TCP Connection by Using a Queue Data Structure

As Figure 18 shows, new requests were added to the tail of the queue, and a request was removed from the head of the queue for every incoming response. We maintained a queue for every TCP connection opened by a host.

The SYN, FIN and RST flags of a TCP packet indicated when a TCP connection was opened, closed or terminated abnormally respectively. We opened a queue for a TCP connection when a TCP packet's SYN flag was set and closed the queue when the FIN or RST flags were set. Whilst a TCP connection queue was open we added requests to it as we encountered the requests in the host datafile. When we encountered a response in the host datafile, we matched it to the request at the head of the queue, and removed the request from the queue. We implemented the queue data structure using dynamic memory allocation. We used static memory allocation for the real-time measurement system. Using dynamic memory allocation obviated the need to estimate queue sizes before runtime. The use of dynamically sized queues resulted in a more flexible and accurate processing tool.

Packets were sometimes lost during transmission. If a request or response message was lost, subsequent responses were wrongly matched to requests. In order to correctly address this problem, every TCP sequence number had to be checked to ensure that packets were not lost. We did not capture the level of detail necessary to solve this problem.

We approximated a solution to the problem by detecting when a request or response was lost on a TCP connection. The behaviour of the queue associated with the connection was used as an indicator of packet loss. During error-less transmissions, a TCP connection queue was opened, messages were added to and removed from the queue, and the queue was closed when a FIN packet was received. The queue was always empty at the time a FIN packet was received. If the queue was not empty, we knew that a message had been lost. In this case we invalidated all requests and responses on the connection. We found that the invalidation of TCP queues did not happen very often.

4.5 Web User vs. Web Client Request Differentiation Problem

We discussed the web user vs. web client request differentiation problem in Section 3.4.1 (page 38) when describing the real-time measurement system. We solved the problem by compiling a list of characteristics for a web client request. We differentiated between web user and client requests by testing whether a new request had any of the characteristics in the list. If a request did have a characteristic that matched one of the characteristics in the list, it was categorised as a web client request, and if not, it was categorised as a web user request.

We compiled the list of characteristics by studying measurement file data. The data fields available in the measurement file data are shown in Tables 5 - 7 (page 46). We used some of the fields shown in Tables 5 - 7 to compile the list of characteristics. The fields we used were the following:

1. `Host Part of Request URL`
2. `Path Part of Request URL`
3. `Host Part of Referrer URL`
4. `Path Part of Referrer URL`

We used two more pieces for information which were not recorded in Tables 5 - 7. They were the following:

1. `Inter arrival Time Between Requests`
2. `Type of Request`

The `Inter arrival Time Between Requests` was calculated from data in Tables 5 - 7, and the `Type of Request` was calculated by studying the file extensions of HTTP request URLs.

4.5.1 Categorisation of HTTP Requests

We categorised requests according to the type of file requested. We distinguished between the following categories:

HTML: Requests for files with HTML content.

GRAPHICS: Requests for image files.

OTHER: Requests for all other files.

We observed that HTTP requests had file extensions matching the extension categories shown in Figure 19.

```
HTML: .html, .js, .cgi, .php, .asp, .pl, .cfm, .vbs, .css

GRAPHICS: .gif, .jpg, .png, .jpeg
```

Figure 19: HTTP Request File Extensions for HTML and GRAPHICS Categories

We wrote a function which categorised requests by comparing the various extensions listed in Figure 19 to the string contained in `Path Part of Request URL` field as recorded in Table 5 (page 46). If a match was found the request was categorised as belonging to the category to which the extension belonged.

The `Path Part of Request URL` string was typically very long, and we observed that the extension often occurred somewhere within a string as opposed to occurring at the end of a string. The comparison between the extension and the `Path Part of Request URL` was done for every single file extension in the list of extensions shown in Figure 19, until a match (or none) was found. As many comparisons were performed, we needed a fast string comparison function.

We implemented the suffix array search method as described by Jon Bentley in *Programming Pearls* in our program [Ben86]. The search function found matching substrings within a string. The function did string comparisons in **O(nlogn)** time.

4.5.2 Web Client Request Characteristic List

The web user vs. web client request differentiation problem was solved by the use of a **list of web client request characteristics**. The list was used to identify web client requests. Requests which were not identified as web client requests according to the **list of web client request characteristics** were classified as being web user requests.

The list of web client request characteristics was compiled by studying traces of web traffic. A web traffic trace is shown in Figures 44-46 in Appendix A. The figures in Appendix A show selected fields taken from the measurement file for a web user request and subsequent web client requests. By

studying traces of web traffic such as the one in Appendix A, we were able to identify characteristics of web client requests. The **web client request characteristics list** is reported in Tables 9 - 12.

The requests shown in the figures of Appendix A were generated by a single web user request for the `www.cnn.com` URL. The web user request was followed by 82 web client requests for inline images. It is interesting to note that one web user request generated a large number of web client requests. The fact that there were many more web client requests than web user requests prompted us to classify requests according to web client request characteristics.

We next discuss the **web client request characteristics list** as reported in Tables 9 - 12. The characteristics are shown in descending order of importance i.e. tests based on characteristics appearing earlier form a stronger basis for correct classification than tests based on characteristics which appear later in the tables. The characteristics in Tables 9 - 12 are divided into four groups. We indicate the group that characteristics belong to in the third column of each table. The web client characteristic groups are discussed one by one.

4.5.3 Characteristic Group No. 1

Table 9 shows web client request characteristics belonging to Group No. 1.

No.	Characteristic	Characteristic Group
1	A GRAPHICS request with a `Referrer URL` matching that of any other request preceding it by less than 10 seconds is likely to be a web client request	**Group 1:** Web client request following another web client request
2	An HTML or OTHER request with a `Referrer URL` matching that of any other request preceding it by less than 2 seconds is likely to be a web client request	
3	A GRAPHICS request with a `Host Part of Request URL` matching that of any other request preceding it by less than 5 seconds is likely to be a web client request	
4	An HTML or OTHER request with a `Host Part of Request URL` matching that of any other request preceding it by less than 2 seconds is likely to be a web client request	

Table 9: Web Client Request Characteristics - 1

Group No. 1 characteristics were based on observations of a series of web client requests following after one another. We observed that a sequence of web client requests caused by the same web user request usually had the same `Referrer URL`. We also observed that the time between these successive requests was usually very small. Characteristic No. 1 in Table 9 categorises a GRAPHICS request following shortly after any other type of request with the same `Referrer URL` as being a web client request. Characteristic No. 2 categorises HTML and OTHER requests in the same way, but requests have to follow one another within a smaller time frame. The smaller time frame was due to the fact that HTML or OTHER requests were likely not to be web client requests if they followed longer

than 2 seconds after other requests with the same `Referrer URL` i.e. they were then more likely to be web user requests.

Characteristics No.s 3 and 4 state that the `Host Part of Request URL` was usually the same for a sequence of web client requests caused by the same web user request. These characteristics were less certain than the previously discussed `Referrer URL` characteristics, but were included in Table 9 to categorise requests which did not have a `Referrer URL` recorded in the measurement file. The `Referrer URL` was often missing from measurement file entries as some browsers did not use the `Referrer URL` field, and sometimes software filters were used on networks to filter out the `Referrer URL` for security reasons. The time frame for categorisation based on the comparison of GRAPHICS requests to preceding requests was reduced to 5 seconds (as opposed to 10 seconds for the `Referrer URL`) due to categorisation based on the `Host Part of Request URL` being less certain than that based on the `Referrer URL`.

Group No. 1 characteristics were implemented in the parameter extraction tool by means of comparing a request's characteristics to previous requests' characteristics. Previous requests were stored in the `all_request_queue`. The `all_request_queue` stored previous requests for all types of files i.e. `HTML`, `GRAPHICS` and `OTHER` files, as opposed to the `html_request_queue` which only stored previous `HTML` requests. The `html_request_queue` is used for testing Group No. 2 characteristics and is discussed in Section 4.5.4.

The implementation of Group No. 1 characteristics is shown in Appendix B.

4.5.4 Characteristic Group No. 2

Table 10 shows Group No. 2 web client request characteristics.

No.	Characteristic	Characteristic Group
5	A GRAPHICS request with a `Referrer URL` matching that of any HTML request whose response precedes it by less than 50 seconds is likely to be a web client request	**Group 2:** Web client request following an HTML request
6	A GRAPHICS request with a `Referrer URL` matching the `Request URL` of any HTML request whose response precedes it by less than 50 seconds is likely to be a web client request	
7	An HTML or OTHER request with a `Referrer URL` matching that of any HTML request whose response precedes it by less than 2 seconds is likely to be a web client request	
8	An HTML or OTHER request with a `Referrer URL` matching the `Request URL` of any HTML request whose response precedes it by less than 10 seconds is likely to be a web client request	

Table 10: Web Client Request Characteristics - 2

Group No. 2 characteristics were based on observations of web client requests following after

HTML requests. Characteristic No. 5 categorises GRAPHICS requests which follow within 50 seconds of the response to an HTML request, and have the same `Referrer URL` as the HTML request, as web client requests.

We observed that HTML requests were often web client requests for content embedded in a web page by means of frames or pop-up windows. These HTML requests generated more requests after they had been parsed. The resultant requests had the same `Referrer URL` as the HTML request which caused them. If the requests were GRAPHICS requests and occurred within 50 seconds of the response to the relevant HTML request, there was a high likelihood for them being web client requests. Characteristic No. 7 states that HTML and OTHER requests have to follow within 2 seconds of the response to the HTML request to be categorised as web client requests, as there is a large possibility of these requests being new web user requests.

We observed that GRAPHICS requests with a `Referrer URL` matching the `Request URL` of the response to an HTML request which preceded it by less than 50 seconds were likely to be web client requests. This situation arose when the `Referrer URL` referred to the web user request which caused the series of web client requests for GRAPHICS files. Characteristic No. 6 characterises web client requests in this way. Characteristic No. 8 states the same for HTML and OTHER requests, but with a time limit of 10 seconds between the response to an HTML request and the request in question, because of the possibility of the HTML request being a new web user request.

The `html_request_queue` stored previous HTML requests, and was used for testing Group No. 2 characteristics.

The html_request_queue

The `html_request_queue` stored the arrival time of an HTML request as well as the arrival time of the response to an HTML request. In comparison, the `all_request_queue` stored the arrival time of requests only.

Observations No.s 5-8 in Table 10 were based on scenarios where an HTML response generates more requests itself, after it had been parsed by a web client. The quantity of importance in these scenarios was the amount of time between the arrival of an HTML response and the arrival of requests generated by the HTML response after it had been parsed. In order to calculate this quantity we needed a record of HTML response arrival times.

On arrival of an HTTP response we matched it to its respective HTTP request as discussed in Section 4.4. After matching the HTTP response, we tested whether the response was an HTML response by inspecting the `Content-Type` field. Figure 20 shows the different types of `Content-Type` fields for HTML responses. If the response was an HTML response we inserted the relevant HTML request and response information (including HTTP response arrival time) as a new element into the `html_request_queue`.

The implementation of Group No. 2 characteristics is shown in Appendix B.

```
text/html, application/x-javascript, text/javascript, text/xml, text/css, text/plain
```

Figure 20: Content-Type field of an HTML response

4.5.5 Characteristic Group No. 3

Table 11 shows Group No. 3 web client request characteristics.

No.	Characteristic	Characteristic Group
9	A GRAPHICS request with a `Referrer URL` matching that of any other request preceding it by less than 60 seconds is likely to be a web client request	**Group 3:** Web client request following another web client request
10	A GRAPHICS request with a `Host Part of Request URL` matching that of any other request preceding it by less than 60 seconds is likely to be a web client request	

Table 11: Web Client Request Characteristics - 3

Group No. 3 characteristics were based on observations of web client requests following web client requests. These characteristics were similar to Group No. 1 characteristics, but for relaxed inter-arrival time constraints and the fact that we only considered GRAPHICS requests in Group No. 3 characteristics.

We observed that GRAPHICS requests with inter-arrival time constraints larger than those specified for Group 1 characteristics were often web client requests. We therefore relaxed the inter-arrival time constraints of characteristics No.s 1 and 2, ensuring that a web client request which was generated later than usual, was correctly categorised.

The reason for not using the relaxed inter-arrival time constraints of observations No.s 9 and 10 from the outset, was that given the relaxed inter-arrival time constraints of characteristics No.s 9 and 10, characteristics No.s 3-8 were more likely to categorise the request correctly than characteristics No.s 9 and 10.

The implementation of Group No. 3 characteristics is shown in Appendix B.

4.5.6 Characteristic Group No. 4

Table 12 shows Group No. 4 web client request characteristics.

Group No. 4 characteristics were based on the observation that requests for advertising content were very likely to be generated by the last HTML request.

We observed that requests for advertising content usually had URLs containing at least one of strings shown in Figure 21.

We used the string comparison routines discussed in Section 4.5.1 to test whether a request was a request for advertising content. If the `Host Part of Request URL` or `Path Part of Request URL`

No.	Characteristic	Characteristic Group
11	A request for advertising content following an HTML Request is likely to be a web client request generated by the HTML Request	**Group 4:** Advertisement requests and requests following web user requests
12	A request following within 2 seconds of a web user request is likely to be a web client request	

Table 12: Web Client Request Characteristics - 4

```
doubleclick.com, doubleclick.net, akamai, atwola, realmedia.com, zedo.com,
advertising.com, fastclick.net, imrworldwide.com, peel.com, ads, img.co.za
```

Figure 21: Advertising Content URLs or part thereof

or a combination of both matched at least one of the strings in Figure 21, the request was classified as a request for advertising content.

We observed that web user requests very seldomly followed directly upon one another within a 2 second period. This type of traffic was generated by a user clicking twice within a 2 second period, while no other traffic i.e. inline objects, were generated by the web browser. We categorised a request following within 2 seconds of a web user request as being a web client request.

4.6 The Web Client Request Matching Problem

The problem was solved by the implementation of the solution to the `Web User vs. Web Client Request Differentiation Problem`. By differentiating between web user and web client requests we were able to also match web client requests to the correct web user requests. When a request was categorised as being either a web user or a web client request, an index number was assigned to the request. The index number assigned to a web client request associated that request to the relevant web user request.

We achieved the correct matching by recording index numbers of requests which had already been categorised in the `all_request_queue` and `html_request_queue`. Subsequent tests made use of the information in the `all_request_queue` and `html_request_queue` to match web client requests to web user requests.

4.7 Removal of Unrepresentative Data

We used the packet trace method to obtain data measurements. Due to the nature of the packet trace method, some of the data we recorded were not relevant to our study. We identified and removed data which were not relevant to our study. A considerable amount of data were removed.

The model we defined in Chapter 2 characterised web-browsing traffic generated by a single host on a campus network. We have previously defined traffic generated by our model as **traditional web traffic**. Any data which did not correspond to traditional web traffic were removed from the study. TCP/IP and HTTP errors, HTTP data generated by applications other than web-browsers such as web-download and web-irc traffic were removed. Data were removed from host datasets during the processing of datasets, Section 4.7.1 documents the process of removing unrepresentative data. We also removed complete host datafiles which had parameter datasets with too few entries to be analysed, Section 4.7.2 documents the process of removing unrepresentative datasets.

4.7.1 Removal of Unrepresentative Data from Host Datasets

Data generated by TCP/IP and HTTP errors and data generated by applications other than web-browsers such as web-download, web-irc traffic and Windows Messenger data were removed from host datasets.

We removed all Windows Messenger traffic from measurements. The Windows Messenger application used the HTTP. Windows Messenger HTTP traffic was asynchronous i.e. several messages were sent in one direction without responses returning in the same order on a TCP connection. Web HTTP traffic on the other hand was synchronous. Our tools processed HTTP data assuming that the traffic was synchronous. Asynchronous traffic transmitted on the same TCP connection as web HTTP traffic caused errors during the processing of data.

We used the string comparison routines discussed in Section 4.5.1 to identify Windows Messenger HTTP response messages. The `Content Type` field of Windows Messenger response messages matched the string: “messenger”. We discarded Messenger responses during processing. We discarded all pending HTTP requests on the TCP connection queue associated with the TCP port on which a Messenger response was found (the TCP connection queue was discussed in Section 4.4). We discarded pending HTTP requests on the same TCP connection because the asynchronous nature of Messenger messages could not be processed by our application.

We removed web-download requests from host datasets. We used the previously discussed string comparison routine to identify web-download requests. We observed that the `Path Part of Request URL` field of web-download requests typically matched the file extensions shown in Figure 22. We identified a request as being a web-download request by matching it to one of the strings in Figure 22. We categorised requests for files larger than 1Mb as being web-download requests.

We created a **blacklist** category for all traffic other than traditional web traffic and web-download traffic. We compiled a list of URLs which were associated with this traffic. Figure 23 shows URLs or parts of URLs that were identified as being blacklisted. This was time consuming task, but was

```
.cab, .zip, .exe, .dll, .gz, .gzip, .wmz, .gbdzip, .scr,
.jar, .pfr, .class, .mp3, .ra, .avi, .au, .mpeg, .mpg, .mid,
.mov, .pdf, .ps, .ppt, .doc, .bmp, .art, .ico, .psp, .tif,
.wmv, download, .rm and .iso
```

Figure 22: File Extensions Associated with Web Download Requests

the only way to identify and remove unrepresentative traffic from the datasets. We categorised an HTTP request as being a blacklisted request if the request was a request for a blacklisted URL. Blacklisted traffic was generated by applications such as: web-irc, streaming audio, streaming video, peer-to-peer file transfer, automated photo-gallery applications etc.

```
icq.com, 205.188.250.25, mirabilis.com, africam.com, ads.win4win.com, pgq.yahoo.com,
toolbar.mweb.co.za, windowsupdate, toolbar.google.com, liveupdate.symantec,
catchthegroove.com, smgradio.com, gator.com, mail.com, outblaze.com, realmedia,
svcs.microsoft.com, WindowsMessenger, gnucleus.gnutelliums.com, update, flipside.com,
www.xingtong.dns2go.com, download, tiscali.co.za, wustat.windows.com, defsoul.com,
spencer.idjmg.com, calico.ac.za, kazaa.com, www.jade.za.net, cricket.org,
mailbits.com, ict.sportingodds.com, www.page3.com, hotmail, hotmail, ticker,
update.companion.yahoo, www.pku.edu.cn, defjam.com, link_to_database
```

Figure 23: Blacklisted URLs or Parts of URLs

A request of which either the `Host Part of Request URL` or `Path Part of Request URL` or a combination of both matched any of the strings contained in Figure 23 was categorised as a blacklisted request.

We observed that web client requests often fell in the blacklisted category, but passed the test for blacklisting due to being redirected and hence having URLs different to the ones shown in Figure 23. We implemented a blacklist queue which stored a list of blacklisted web user requests. Web client requests were compared to the web user requests in the blacklist queue, and removed from the relevant host dataset if they matched the blacklisted web user request.

We removed 10076332 requests and responses from the host datasets. We were left with 32575718 requests and responses in the host datasets.

4.7.2 Removal of Unrepresentative Host Datasets

We removed host datasets which were not generated by web users from the the study. We removed 2 host datasets that were generated by web cache proxy software and 1 host dataset that was generated by web spider software.

We removed parameter datasets with less than 30 entries from the study. We removed 4519 host datasets because the `Browsing Inter-Session Time` parameter datasets had less than 30 entries.

We removed host datasets which were unlikely to have been generated by users browsing the web.

It was unlikely for a user to request more than 500 web pages per session. Host datasets with `Number of Web User Requests per Browsing Session` parameter datasets which contained values larger than 500 were unlikely to have been generated by web users. We removed 20 host datasets with values larger than 500 for the `Number of Web User Requests per Browsing Session` parameter.

We removed 323 host datasets because they had entries larger than 1000 for the `Number of Web Client Requests per Web User Request` parameter datasets. It was unlikely for a web page to have more than a 1000 inline images.

A total of 4865 datasets were removed from the 6692 host datasets. We were left with 1827 datasets for analysis. We used 715 of the datasets for analysis and 1112 for validation purposes (randomly split datasets into two groups). The reason for the uneven split in numbers between analysis and validation datasets was the removal of datasets after we divided the datasets into two groups. We removed datasets for reasons previously discussed. As previously discussed the validation of our traffic model was left for future work.

4.8 Concluding Remarks

We developed a **heuristic algorithm** which **differentiated** between web user and web client requests. The heuristic algorithm was based on a **list of web client characteristics**. We identified web client characteristics by studying web traffic traces.

We used the heuristic algorithm to **extract parameter datasets** from captured data. The data we extracted parameter datasets from were captured by taking traffic measurements on a campus network. Using the heuristic algorithm, we were able to draw inferences about user behaviour based on the limited information at our disposal. We had limited information at our disposal due to the nature of the packet trace measurements technique. No information about user behaviour was captured in the data.

We **validated** the heuristic algorithm by performing an experiment in our laboratory. We generated a series of web user and web client requests and applied the data processing application which implemented the heuristic algorithm to the data. Requests were categorised **very accurately** by the algorithm. The **quantification** of the algorithm's accuracy was left for future work.

The accuracy of the heuristic algorithm can be attributed to the use of **detailed information** to categorise requests. We used information extracted from TCP, IP and HTTP packet headers to categorise requests. Previous work used only TCP and IP information to categorise web requests [SCJO01]. The use of HTTP information resulted in a more accurate approach to the categorisation of web requests.

There was a **tradeoff in performance** for the accuracy which the heuristic algorithm had. Algorithms which did not make use of HTTP information in categorising requests, were computationally more efficient than the heuristic algorithm. These algorithms were used to process data in real-time. It was not possible for us to capture and process data in real-time. Captured data had to be processed off-line by the heuristic algorithm. The off-line processing of data was not a problem

as we had sufficient storage space to process captured data off-line.

We **removed data** which were not relevant to the study from the captured data. Data which did not correspond to data generated according to the **traditional web browsing model** were not relevant to the study. The traditional web browsing model was previously discussed. In essence it was a model which modelled the common use of the web as a conveyer of information by means of web pages. Web pages constituted text and images.

A large amount of measured data were removed. The reason we removed a large amount of data was that a large amount of data were generated by web-irc or web-download applications. Web-irc and web-download traffic did not correspond to the traditional web browsing model. We decided to strictly define our workload model as a model of traditional web browsing traffic only. The reason for this strict definition was that the model was defined for the purpose of simulating traffic generated on wireless networks. We did not expect users to download large amounts of data or use web-irc applications extensively on wireless networks. The quantification of the amount of traffic generated by traffic other than traditional web browsing traffic on a campus network was left for future work.

The off-line processing tool was developed to be **accurate** rather than fast. We made the decision to concentrate on accuracy rather than performance, as the Internet link which we measured was relatively small (6Mbps). It was possible for us to capture 30 days worth of web traffic on 25GB of secondary storage. The approach of accuracy rather than performance would however not have worked on network links with much larger bandwidth. Many universities in the United States have Internet bandwidth in the order of 100Gbps. The storage requirement for capturing traffic on these networks with our measurement tool would have been very large. It would not have been possible to use our measurement tool to capture data on networks with bandwidth in the order of 100Gbps. Our measurement and processing tools could however be used to capture and process a subset of the data transmitted on networks with very large bandwidth.

Chapter 5

Statistical Methodology

5.1 Introduction

We analysed the datasets for each of the eleven model parameters. We intended to find a function of a **specified mathematical family** which fit each of the eleven workload model parameters well. As previously discussed, the data were more complicated than we anticipated. We did not find a perfect fit to a mathematical function for any of the model parameters. We however managed to find mathematical functions which fit the data reasonably well.

By function of a specified mathematical family we mean a distribution family e.g. the normal or exponential distribution families, as well as the model constants associated with the family e.g. location, shape and scale. We started by considering four well known mathematical functions as models for the data namely the exponential, Weibull, lognormal and Pareto distributions. We implemented analysis routines for these distributions in the **R statistical analysis environment**. We found that these distributions did not provide enough flexibility in terms of shape and size to model the eleven model parameters. We added routines for the normal, beta, gamma and extreme value distributions to our toolset of R programs. We found **reasonable matches** for the eleven model parameter datasets by using these eight statistical distributions.

We used visual techniques to explore the data. We applied goodness-of-fit statistics to the data to determine which of the eight distributions fit the data best. We used the histogram, Q-Q plot, probability plot and empirical cumulative distribution function to visually explore the data. We used the **method of maximum likelihood estimation** to obtain maximum likelihood estimates for model constants. We used two goodness-of-fit statistics, the **Anderson Darling statistic** and the λ^2 **discrepancy measure**.

The λ^2 discrepancy measure is a statistic which relied on the **binning of data**. The size chosen for bin width affected the accuracy of the statistic. We used well known techniques to accurately calculate bin widths. Bin widths for datasets with a skewed distribution had to be calculated differently to those for datasets with symmetrical distributions. Most of the datasets we analysed

were positively skewed. We used a technique suggested by Scott to calculate optimally sized bins for these datasets [Sco92].

The **very large size** of most of the datasets we analysed was a major obstacle. Most of the datasets had millions of entries. Statistical analysis packages typically were not able to analyse datasets with more than a million entries. We had to implement our own analysis routines in the R statistical analysis environment. Another problem we had was that statistical theory often did not provide the necessary tools to analyse datasets with more than a million entries. The Anderson Darling statistic in particular could not be used for the analysis of datasets with more than 200 entries.

Many of the datasets we analysed showed signs of **heavy-tailed distributions**. There were many techniques available to analyse the tail of a distribution. We used the well known technique of exploring the tail-end of a distribution's complementary cumulative distribution function on a log-log scale. We also used the well known Hill plot technique.

We describe the eight distributions we tested for, and some of their properties in Section 5.2. Section 5.4 describes the visual techniques we used for data exploration. Section 5.5 describes the goodness-of-fit metrics we used i.e. the Anderson Darling and λ^2 statistics. Section 5.6 discusses the techniques we used to identify heavy-tailed distributions.

5.2 Analytic Distributions

We tested eight well-known probability distribution families for **goodness-of-fit** against the eleven parameter datasets. The distribution families we tested were the exponential, Weibull, lognormal, normal, beta, gamma, extreme value, and Pareto probability distribution families. The eight distribution families we tested have often been used in simulation studies as workload models. Together, the chosen distribution families represent a wide variety of possible shapes.

We were specifically interested in distribution families which were able to match heavy tailed empirical distributions. Heavy tailed distributions were commonly encountered in network traffic modelling studies. Amongst the distribution families we selected there were several which could model heavy tailed empirical distributions. Previous studies by Paxson, Feldmann, Barford and Crovella [Pax94, Fel98, BC98b] used these eight distribution families to characterise Internet workload.

Tables 13 and 14 summarise functions associated with the probability distributions.

Table 15 lists the maximum likelihood estimators used to estimate model constants for the different distributions.

[1] $B(\alpha, \beta) = \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} dt$
[2] $\Gamma(\gamma) = \int_0^\infty t^{\gamma-1} e^{-1} dt$
[3] Solve for γ and α
[4] Solution exists only when a and b are known
[5] $\Psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$

Distribution Family	Distribution Parameters	General Density Function
Exponential	$\mu,\ \alpha$	$\frac{1}{\alpha}e^{-\frac{(x-\mu)}{\alpha}} \quad x > \mu; \quad \alpha > 0$
Weibull	$\mu,\ \gamma,\ \alpha$	$\frac{\gamma}{\alpha}\left(\frac{x-\mu}{\alpha}\right)^{\gamma-1} e^{\left(\frac{-(x-\mu)}{\alpha}\right)^{\gamma}} \quad x > \mu; \quad \gamma > 0; \quad \alpha > 0$
Lognormal	$\theta,\ \zeta,\ \sigma$	$\frac{e^{-\left(\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right)}}{(x-\theta)\sigma\sqrt{2\pi}} \quad x \geq \theta; \quad \zeta > 0; \quad \sigma > 0$
Normal	$\mu,\ \sigma$	$\frac{e^{-\left(\frac{(x-\mu)^2}{2\sigma^2}\right)}}{\sigma\sqrt{2\pi}} \quad \sigma > 0$
Beta	$\alpha,\ \beta, a,\ b$	$^1\frac{(x-a)^{\alpha-1}(b-x)^{\beta-1}}{B(\alpha,\beta)(b-a)^{\alpha+\beta-1}} \quad a < x < b; \quad \alpha > 0; \quad \beta > 0$
Gamma	$\mu,\ \gamma,\ \alpha$	$^2\frac{1}{\alpha\Gamma(\gamma)}\left(\frac{x-\mu}{\alpha}\right)^{\gamma-1} e^{\left(\frac{x-\mu}{\alpha}\right)} \quad x > \mu; \quad \alpha > 0; \quad \gamma > 0$
Extreme Value Type 1	$\alpha,\ \beta$	$\frac{1}{\beta}e^{-\frac{(x-\alpha)}{\beta}} e^{\left(-e^{-\frac{(x-\alpha)}{\beta}}\right)} \quad -\infty < x < \infty$
Pareto	$\alpha,\ \beta$	$\beta\alpha^{\beta}x^{-\beta-1} \quad \beta > 0 \quad x \geq \alpha \quad \alpha > 0$

Table 13: Distribution Parameters and General Density Function for Distribution Families Used in the Study

Distribution Family	Cumulative Distribution Function	Mean
Exponential	$1 - e^{-\frac{(x-\mu)}{\alpha}} \quad x > \mu; \quad \alpha > 0$	α
Weibull	$1 - e^{\left(\frac{-(x-\mu)}{\alpha}\right)^{\gamma}} \quad x > \mu; \quad \gamma > 0; \quad \alpha > 0$	$\mu + \alpha\Gamma\left(\frac{\gamma+1}{\gamma}\right)$
Lognormal	No closed form	$e^{(\zeta+0.5\sigma^2)}$
Normal	No closed form	μ
Beta	$\int_a^x \frac{(x-a)^{\alpha-1}(b-x)^{\beta-1}}{B(\alpha,\beta)(b-a)^{\alpha+\beta-1}} \quad a < x < b; \quad \alpha > 0; \quad \beta > 0$	$a + b\left(\frac{\alpha}{\alpha+\beta}\right)$
Gamma	$\frac{\Gamma_x(\gamma)}{\Gamma(\gamma)} \quad x > 0; \quad \gamma > 0; \quad \mu = 0$	$\alpha\lambda + \mu$
Extreme Value Type 1	$e^{-x}e^{\left(-e^{-\frac{(x-\alpha)}{\beta}}\right)} \quad -\infty < x < \infty$	$\alpha + 0.5772\beta$
Pareto	$1 - (\alpha/x)^{\beta} \quad x \geq \alpha$	$\beta > 1: \quad \frac{\beta\alpha}{\beta-1}$ $\beta \leq 1: \quad \infty$

Table 14: Cumulative Distribution Function and Mean for Distribution Families Used in the Study

Distribution Family	**Maximum Likelihood Estimator**
Exponential	$\hat{\alpha} = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}$
Weibull[3]	$\hat{\gamma} = \left[\left(\sum_{i=1}^{n} x_i^{\hat{\gamma}} log x_i\right)\left(\sum_{i=1}^{n} x_i^{\hat{\gamma}}\right)^{-1} - n^{-1}\sum_{i=1}^{n} log x_i\right]^{-1}$ $\hat{\alpha} = \left[n^{-1}\sum_{i=1}^{n} x_i^{\hat{\gamma}}\right]^{1/\hat{\gamma}}$ for fixed $\mu = 0$
Lognormal	$For\ z_i = log(x_i - \theta)$: $\hat{\zeta} = \frac{1}{n}\sum_{i=1}^{n} z_i = \bar{z}$ $\hat{\sigma} = \left[(n-1)^{-1}\sum_{j=1}^{n}(z_j - \bar{z})^2\right]^{\frac{1}{2}}$
Normal	$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}$ $\hat{\sigma} = \left[(n-1)^{-1}\sum_{j=1}^{n}(x_j - \bar{x})^2\right]^{\frac{1}{2}}$
Beta [4]	[5]$\Psi(\hat{\alpha}) - \Psi(\hat{\alpha} + \hat{\beta}) = \frac{1}{n}\sum_{i=1}^{n} log\left(\frac{Y_i - a}{b-a}\right)$ $\Psi(\hat{\beta}) - \Psi(\hat{\alpha} + \hat{\beta}) = \frac{1}{n}\sum_{i=1}^{n} log\left(\frac{b - Y_i}{b-a}\right)$
Gamma	$\frac{1}{n}\sum_{i=1}^{n} x_i = \hat{\alpha}\hat{\lambda}$ $\frac{1}{n}\sum_{i=1}^{n} lnx_i = ln\hat{\alpha} + \frac{\Gamma'(x)}{\Gamma(x)}$ for fixed $\mu = 0$
Extreme Value Type 1	$\sum_{i=1}^{n} e^{-\left(\frac{x_i - \hat{\alpha}}{\hat{\beta}}\right)} = n$ $\sum_{i=1}^{n}(x_i - \hat{\alpha})\left(1 - e^{-\left(\frac{x_i - \hat{\alpha}}{\hat{\beta}}\right)}\right) = n\hat{\beta}$
Pareto	$\hat{\alpha} = min(x_i)$ $\hat{\beta} = n\left[\sum_{i=1}^{n} log(x_i/\hat{\alpha})\right]^{-1}$

Table 15: Maximum Likelihood Estimator for Distribution Families Used in the Study

We used general-purpose optimisation based on Nelder-Mead, quasi-Newton and conjugate-gradient algorithms using numerical derivatives to optimise the log-likelihood of maximum likelihood estimators. The functions for general-purpose optimisation were available in the R statistical analysis environment.

5.3 Correlation and Autocorrelation

In order to associate parameter datasets with mathematical functions based on the frequency of occurence of observations the observations have to be independent of one another.

The **autocorrelation function** can be used to test for independence between time instances of a stochastic process. We used the autocorrelation function to test for independence between observations in each the 11 parameter datasets. We explain the autocorrelation function and related concepts next.

Correlation is a measure of linear dependence between jointly distributed random variables X and Y. The **correlation coefficient** $\rho(X, Y)$ measures the degree of linear correlation between the two random variables X and Y. We define

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X)Var(Y)}} \tag{1}$$

where both the variances and covariances of both X and Y exist and the variances are nonzero. The covariance of X and Y is

$$Cov(X, Y) = E(XY) - E(X)E(Y) \tag{2}$$

The following relationship holds

$$-1 \leq \rho(X, Y) \leq 1$$

It is said that X and Y are independent or uncorrelated if $\rho(X, Y) = 0$. It is said that X and Y are negatively correlated if $\rho(X, Y) < 0$ i.e. $X = -aY + b$ for $a > 0$ and all b. It is said that X and Y are positively correlated if $\rho(X, Y) > 0$ i.e. $X = aY + b$ for $a > 0$ and all b.

Correlation between instances of the same random variable is measured by the **autocorrelation function**. The autocorrelation function is denoted by $\gamma_X(\ell)$ where ℓ indicates the lag. The lag is the distance between values tested for correlation.

The **autocorrelation function** γ of a stochastic process is the joint moment of random variables $X(t_1)$ and $X(t_2)$. The autocorrelation function is

$$\gamma(t_1, t_2) = E[X(t_1)X(t_2)] \tag{3}$$

As with the correlation coefficient for two random variables discussed earlier, the autocorrelation function is a measure of the relationship between two time instances of a stochastic process. A related quantity is the **autocovariance**

$$C[t_1, t_2] = \gamma(t_1, t_2) - E[X(t_1)]E[X(t_2)]. \quad (4)$$

For many random variables the value $\gamma(1)$ is particularly significant. If a random variable is correlated the correlation is often greatest at a lag of 1. We decided whether a parameter dataset was correlated by considering the value of $\gamma(1)$ for the parameter dataset. If $|\gamma(1)| < 2/\sqrt{n}$ (where n is the number of observations in the dataset) then the dataset was considered to be uncorellated. This follows from the fact that $|\gamma(1)| < 2/\sqrt{n}$ is true for uncorrelated datasets 5% of the time. We therefore constructed a 95% confidence interval for $\gamma(1)$.

5.4 Visual Techniques

We used the following exploratory data analysis techniques to analyse the data:

1. The Histogram.
2. The Empirical Cumulative Distribution Function.
3. The Log Empirical Complementary Cumulative Distribution Function.
4. The P-P Plot.
5. The Q-Q Plot.

Histograms were used to identify the shape, location and scale of the distribution of data. They showed the presence of symmetry, peakedness, outliers and heavy tails. Histograms required us to choose a size for bin width. A poor choice of size resulted in loss of information or over-sensitivity to small changes in data distribution. There were three commonly used rules for bin width calculation: the Sturges rule, the Friedman/ Diaconis rule and the Scott rule. The rules for bin width calculation relied on the assumption that the data were normal. Non-normal data required more bins than normal data. The data we analysed were mostly non-normal. We addressed the problem by implementing an adaptation of the Scott rule which results in optimally sized bins for non-normal data [Sco92].

The empirical cumulative distribution function had the following advantages over the histogram:

1. It did not involve “binning”.
2. Its complexity was independent of the number of observations.
3. It supplied direct information about the shape of the underlying distribution.
4. It supplied robust information on location and dispersion.

The log empirical complementary cumulative distribution function was used to analyse the right hand tail of a distribution. It was also used to indicate the goodness-of-fit of the exponential distribution to the data.

P-P and Q-Q plots were used to judge the goodness-of-fit of a particular mathematical function to a dataset. We estimated the model constants for a particular parameter dataset by using the formulae reported in Table 15. The plots formed a straight line when the hypothesised distribution was the true underlying distribution of the data. It afforded the opportunity to judge goodness-of-fit by judging the linearity of the plot. We tested the goodness-of-fit of the Q-Q plot to the straight line by using the measure

$$\frac{1 - \sum_{i=1}^{n}(x_i - y_i)^2}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}} \tag{5}$$

where x_i are instances of the explanatory variable and y_i are instances of the response variable.

The R code for the log empirical complementary cumulative distribution function plot, and P-P and Q-Q plots for selected mathematical functions are given in Appendix C.

Measures of location and spread such as the quartiles of the data, the median, mean, variance, skewness and kurtosis gave an indication of the distribution of data.

5.5 Statistical Techniques

Goodness-of-fit statistics quantified evidence suggested by visual analysis. We used the Anderson Darling and λ^2 statistics.

5.5.1 Anderson Darling Statistic

The Anderson Darling statistic was based on the empirical distribution function (EDF). We implemented the statistic in R according to the description in the chapter "Tests Based on EDF Statistics" in the monograph "Goodness-of-Fit Techniques" by R.B. D' Agostino and M.A. Stephens [Ste86]. The R code for the test is listed in Appendix C.

EDF statistics were more accurate than statistics based on binning techniques such as the χ^2 goodness-of-fit statistic. EDF statistics did not involve the binning of data, which would have resulted in loss of information. The Anderson Darling statistic was particularly well suited for detecting departures from the true distribution in the tail of the distribution. This property was of particular significance to us as it had been shown that data collected from data network traces were often heavy tailed [CTB98, Res97].

We used percentage point tables given by Stephens to calculate p-values [Ste86]. Model constants were estimated from the sample data by means of optimising the log-likelihood of the estimators given in Table 15 (page 70).

A considerable drawback of the Anderson Darling statistic was that it could not be used for tests on large datasets [Pax94, BC98b]. The statistic had published tables for p-values for datasets with

sizes up to 200 entries. The datasets we analysed usually had more than one million entries. The Anderson Darling statistic could therefore not be used to analyse the datasets we had to analyse. We investigated the lambda discrepancy statistic as an alternative test.

5.5.2 Lambda Discrepancy Statistic

The λ^2 discrepancy statistic was defined by Pederson and Johnson in the paper "Estimating Model Discrepancy"[PJ90] as

$$\hat{\lambda}^2 = \frac{X^2 - K - df}{N - 1} \tag{6}$$

where N is the sample size and df is defined as n-r-1, where n is the number of bins and r the number of model constants estimated from data. Assume that $Y = (Y_1, Y_2, \ldots, Y_n)$ is a multinomial random variable with $p = (p_1, p_2, \ldots, p_n)$ denoting a hypothesized probability vector for Y, then

$$K = \sum_{i=1}^{n} \frac{(Y_i - Np_i)}{Np_i} \tag{7}$$

and

$$X^2 = \sum_{i=1}^{n} \frac{(Y_i - Np_i)^2}{Np_i} \tag{8}$$

In order to construct confidence intervals we used a consistent estimator of the variance of λ^2

$$\hat{v}(\hat{\lambda}^2) = [2df + 4N\hat{\lambda}^2 + 4N\hat{\lambda}^4 + 4T]/N^2 \tag{9}$$

where

$$T = \sum[(Y_i - N\hat{p}_i)^3 - 2(Y_i - N\hat{p}_i)(N\hat{p}_i) + (5/2)(Y_i - N\hat{p}_i)^2 + (3/2)Y_i]/(N\hat{p}_i)^2 \tag{10}$$

As can be seen from Equation 6, the λ^2 statistic was based on the χ^2 goodness-of-fit statistic. The λ^2 and χ^2 statistics were based on binning techniques, and measured the magnitude of departure of empirical data from a distributional model. It had been found that for smaller datasets the λ^2 statistic was less biased and had smaller variance than the χ^2 statistic [PJ90]. Of even greater importance to us was the fact that unlike the Anderson Darling statistic, the λ^2 statistic could be used on large datasets.

Another advantage of the λ^2 statistic was that it could be used to compare the goodness-of-fit of tests performed on datasets with different sample sizes. It is not possible to compare tests performed on datasets with different sizes when using Pearson's χ^2 or the Anderson Darling statistic. The λ^2 statistic could be used to compare results from tests performed on datasets of different sizes because the sample size and number of bins were taken into account in the calculation of the statistic.

Binning Methods

The λ^2 statistic was based on a binning technique. The statistic suffered from the same weakness as the histogram i.e. the user had to choose a bin size. A poor choice resulted in imprecision in the statistic. A bin size chosen to be larger than necessary resulted in the statistic measuring differences on a gross scale. Measuring differences on a gross scale missed small but important deviations in the data. A bin size chosen to be smaller than necessary exaggerated small discrepancies between dataset and model. Exaggerating small discrepancies resulted in falsely rejecting functions which fit the data well. Studies on data network traffic by Paxson and Feldmann [Pax94, Fel98] approached the problem of choosing bin sizes by using the strategy suggested by Scott [Sco79]. Bin sizes were calculated by using the following formula

$$w = 3.49\hat{\sigma}_x n^{-1/3} \tag{11}$$

where w was the bin width, σ_x was the estimated standard deviation and n the number of elements in the dataset.

We observed that Equation 11 resulted in oversized bin sizes when small datasets, or datasets with a skewed distribution were analysed. The oversized bins obscured features of the analysed dataset which would have been visible if smaller bins were used. The reason for the miscalculation in bin size was that Equation 11 assumed normally distributed data. The data we analysed were mostly skewed and therefore using Equation 11 resulted in the calculation of inaccurate bin sizes.

In order to take into account the skewness of data Scott showed that for non-normal data a larger number of bins and hence smaller bin size was necessary [Sco92]. Scott showed that Equation 11 should be multiplied by a "skewness factor" when bin width was calculated for data with a possible log-normal distribution. A random variable Y with a possible log-normal distribution should be multiplied by

$$\frac{2^{1/3}\sigma}{e^{5\sigma^2/4}\left(\sigma^2+2\right)^{1/3}\left(e^{\sigma^2}-1\right)^{1/2}} \tag{12}$$

where σ^2 was the parameter of the normally distributed random variable $X = logY$. The strategy of multiplying Equation 11 by the "skewness factor" in Equation 12 could be applied to distributions which are skewed to the right, as the log-normal distribution is skewed to the right. We deem this a plausible leap of faith.

We implemented both binning strategies, and used the adapted strategy for skewed distributions. Most of the data we analysed had skewed distributions.

We followed Moore's guidelines when calculating the number of observations per bin [Moo86]. We added adjacent bin counts for bins with less than 5 observations. Datasets which resulted in less than 4 bins were removed from the analysis. The R code for our implementation of the λ^2 statistic is listed in Appendix C.

Domain of Statistic

As previously discussed, we failed to find perfect matches between the mathematical functions and data we were analysing. We did however manage to find reasonable matches for most of the data. During the initial analysis of the data we however failed to find even reasonable matches for any of the datasets. The problem we discovered was the implementation of the λ^2 statistic that we were using.

Our implementation of the λ^2 statistic calculated the statistic over the entire domain which the mathematical function tested for was defined. The empirical data however fell within a well defined range of values, with set minimum and maximum values. The domain of the mathematical function tested for extended beyond the minimum and maximum values of the empirical data. The fact that the mathematical functions extended beyond the set minimum and maximum values of the empirical data caused the λ^2 statistic to falsely reject mathematical functions which actually fit the data well over the range of values between the minimum and maximum values.

The set maximum and minimum values found in empirical data were the result of the heuristic algorithm we used to process empirical data. The heuristic algorithm was based on observations of maximum and minimum values for packet sizes and inter-arrival times in packet traces. The heuristic algorithm synthetically introduced rigid maximum and minimum values in empirical data because it used set minimum and maximum values when processing captured data.

We changed our implementation of the λ^2 statistic to solve the problem of falsely rejecting mathematical functions which fit the empirical data. Instead of calculating the value of the statistic over the domain of the mathematical function tested for, we calculated the statistic over the range of values contained in the empirical data. We found reasonably good fits to mathematical functions for the empirical data by using this implementation of the statistic.

We were able to adopt the approach of calculating the value of the statistic over the range of values in the empirical data because of the goal of our study. The goal of our study was to find a mathematical representation which would allow us to generate random numbers for simulation purposes. By fitting a subset of the domain of a mathematical function to empirical data we were able to attain this goal. The mathematical functions found to fit the data could be used to generate random numbers for simulation purposes over the range of values specified by the maximum and minimum values of the empirical data. When generating random numbers for simulation, values smaller than the minimum and larger than the maximum value were discarded.

We aligned empirical data to analytic distributions with origin at zero by subtracting a fixed value from all empirical data values. Most of the analytic distributions we tested for had an origin at zero. The value subtracted should be added back to random values generated for simulation. We listed the value subtracted along with the maximum and minimum values for each mathematical function we found to match the empirical data in Tables 48 and 49 (page 124).

Implementation of Statistic

The code for our implementation of the λ^2 statistic is listed in Appendix C. We implemented the statistic by calculating expected values for the number of elements per bin by using the cumulative distribution function of the distribution being tested for. Another approach was to calculate expected values for the number of elements per bin by generating random variates from the distribution being tested for, and to compare them to the empirical data. Results of the alternative approach to implementing the test however vary between different runs of the test on the same data, because the test is based on random variates.

The advantage of the approach we followed was that it produced consistent results between different tests on the same data. Another advantage of the approach we followed to implement the statistic was that we did not have to solve for the roots of a mathematical function with specified model constants. We simply used the minimum and maximum values contained in the empirical data. We would have had to solve for the roots of a mathematical function if we implemented the test without using minimum and maximum values contained in the empirical data.

The λ^2 statistic could take on any integer value. Positive numbers indicated that the data did not fit the mathematical function tested for, and negative numbers indicated that the data did fit the function. The smaller the result was, the better was the fit of the data to the function tested for.

5.6 Heavy Tailed Distributions

A heavy tailed distribution is a distribution for which the upper part or "tail" of the distribution declines according to a power rate rather than an exponential rate. Distributions commonly used to model network characteristics such as the exponential and normal distributions have tails which decline exponentially or faster. Heavy tailed distributions have tails that decline slower than these distributions which result in a greater degree of variability in the size of observations. The probability of larger observations occurring is not negligible. We say that a random variable X follows a heavy-tailed distribution with tail index α if

$$P[X > x] \sim cx^{-\alpha}, \;\; as\; x \to \infty, \;\; 0 < \alpha < 2, \tag{13}$$

where c is a positive constant, and where $\sim$ means the ratio of the two sides tends to 1 as $x \to \infty$ [CTB98]. This distribution has infinite variance, and if $\alpha \leq 1$ it has infinite mean. Heavy tailed behaviour can be detected in a dataset by inspecting the data set's complementary cumulative distribution function (ccdf) which is defined as

$$P[X > x] = \bar{F}(x) = 1 - F(x),$$

where F(x) is the cumulative distribution function $F(x) = P[X \leq x]$ [CTB98]. Datasets which exhibit heavy tailed behaviour have the property that the tail-end of their ccdf plotted on a log-log scale is linear. This property follows from Equation 13 from which we can derive

$$\lim_{x\to\infty} \frac{d\ \log \bar{F}(x)}{d\ \log x} = -\alpha$$

so that for large x the ccdf should appear to be a straight line on log-log axes with slope $-\alpha$.

The Hill Plot [DdHR00] is another method of estimating α. The Hill plot is a plot of the Hill estimator for a range of values of k, which is defined as the number of upper order statistics used in the calculation of the Hill estimator. The Hill estimator is defined as

$$H_{k,n} = \frac{1}{k}\sum_{i=1}^{k} \log \frac{X_{(i)}}{X_{(k+1)}} \tag{14}$$

where X is the order statistics $X_1, \ldots, X_n$ of the observations in the dataset being tested, such that

$$X_{(1)} > X_{(2)} > \ldots > X_{(n)}$$

and k is the number of these upper order statistics used in the estimation as mentioned before [DdHR00]. The Hill Plot is therefore the plot of

$$((k, H_{k,n}^{-1}), 1 \leq k < n)$$

The plot is interpreted by finding the value of the ordinate where the plot starts to "stabilise" by turning into a line parallel to the x-axis. The value is used as an estimate for α.

5.7 Concluding Remarks

It was difficult to analyse the very large datasets we extracted from the measurement file. Most of the parameter datasets had more than one million entries, some had as many as 15 million entries. It was not possible to use many of the statistical techniques we intented to use to analyse the data because of the large size of the datasets. For example Q-Q plots could not be used to plot datasets with 15 million entries. It was not practical to plot datasets which contained 15 million points as it took too long to plot these datasets. We used the **R statistical analysis environment** to overcome problems with analysing very large datasets. The R statistical analysis environment was fully extensible and allowed us to implement functions which allowed us to analyse very large datasets. We implemented random sub-sampling and selectively removed data from datasets.

The Anderson Darling and λ^2 goodness-of-fit statistics were used in several previous studies involving network traffic data [Fel98, BC98b, Pax94, PF95, Pax93]. We decided at the outset of the study to use these two statistics. We implemented them in the R statistical environment. During the study we became aware of other goodness-of-fit metrics which apparently had some advantages over the techniques we used e.g. the "Akaike information criterion" (AIC) and the "generalised lambda distribution with the method of moments" method. We decided not to pursue these statistics as

we already implemented the Anderson Darling and λ^2 goodness-of-fit statistics and had a lot of experience in using these metrics.

We found that the Anderson Darling statistic was not suitable for our purposes. The statistic could not be used for analysing large datasets. We used the p-value tables published by D'Agostino and Stephens [Ste86] in our tests. For most of the distributions we were testing for the tables typically had p-values for up to 200 observations. The tables also included values for infinitely many observations. The values for infinitely many observations were inaccurate. The values for infinitely many observations were typically not much larger than the p-values for 200 observations. The statistic however grew rapidly for datasets with more than 200 observations.

We performed a test to prove that the values for infinitely many observations were inaccurate. We synthetically generated a large dataset of exponentially distributed values. We applied the Anderson Darling test for exponentially distributed data to the very large dataset of exponentially distributed values. The dataset of exponentially distributed values failed the test. The p-value for infinitely many observations for the exponential distribution as given in the tables was much smaller than the test statistic. We concluded that the Anderson Darling test should only be used for datasets of sizes which have tabulated p-values (values other than the value for infinitely many observations).

In order to overcome the shortcoming of the Anderson Darling statistic, we attempted to use the technique of applying the Anderson Darling test to random subsamples of 200 observations taken from very large datasets. This technique would have enabled us to make use of tabulated p-values. The technique worked for datasets with a few hundred or a few thousand observations, but was not effective for datasets with hundreds of thousands or millions of observations. The subsamples were not representative of the very large datasets and resulted in widely varying results between tests performed on different subsamples taken from the same dataset.

We used the Anderson Darling statistic for analysing datasets with less than 200 observations. We also used it to analyse censored data. Tables for p-values existed for Type I and II censoring for most of the distributions we tested for [Ste86]. We used the same test for censored data that was used by Barford and Crovella to locate the changeover point, in a hybrid distribution for file sizes on a web-server, between a lognormal and Pareto distribution [BC98a].

We used the λ^2 statistic to analyse very large datasets. It produced consistent results across datasets with widely varying sizes. The statistic had the drawback of losing accuracy if bin sizes were badly chosen. We used the guidelines given by Scott [Sco79, Sco92] to calculate optimal bin sizes for tests. We used the "skewness factor" given by Scott [Sco92] to calculate bin sizes for tests on datasets with skewed distributions.

We implemented the λ^2 statistic by calculating the statistic over the range of values contained in the empirical data. This approach is different from the typically used approach of calculating the statistic over the domain of the mathematical function tested for. Our approach had the advantage of disregarding values in the domain of the mathematical function which do not feature in empirical datasets. Using our approach it was possible to find mathematical representations for empirical data which could be used to generate random values for simulation purposes.

Chapter 6

Workload Model Parameters

6.1 Introduction

We found reasonable matches to mathematical functions for the eleven workload model parameters we defined in Section 2.6 (page 27). As previously discussed, during the analysis of parameter datasets we found that two of the 9 original parameter datasets had to be split into two separate datasets each. We therefore ended up with 11 instead of 9 parameter datasets. We discuss how we changed our workload model to take the extra two parameters into account in Sections 6.11 and 6.12 (pages 113 and 118).

In this chapter we discuss how the statistical methodologics of the previous chapter were applied to each parameter dataset in order to obtain best-fit mathematical functions for each parameter dataset. We start by discussing the issue of independence of observations contained in the parameter datasets. We then discuss statistical distributions which were found to match workload model parameters in previous work, and how this background related to our work. The rest of the chapter is an analysis of the 11 parameter datasets of our web workload model.

6.2 Independence of Observations

As previously discussed in Section 5.3 (page 71), collections of elements can be fitted to a statistical distribution only when the individual elements are independent from one another. We made the assumption that the elements in the 11 parameter datasets were independent from one another.

We were confident in our assumption of independence, as we attempted to minimize interdependence between elements in the same parameter dataset by carefully choosing parameters for our workload model. We tried to choose parameters according to functional groupings which would break up the interdependence between elements within the aggregate data.

We tested the assumption of independence between elements by calculating the autocorrelation functions for the 11 parameter datasets. Table 16 lists the results of the test for independence.

Model Parameter	**Autocorrelation with Predecessor**	**95% Probability Interval**
Browsing Inter-Session Time	0.0721	(-0.0096, 0.0096)
Number of Web User Requests per Browsing Session	0.0832	(-0.0086, 0.0086)
Number of Web Client Requests per Web User Request	0.1473	(-0.0024, 0.0024)
Web User Request Inter-arrival Time	0.1429	(-0.0024, 0.0024)
Web Client Request Inter-arrival Time	0.1674	(-0.0005, 0.0005)
Web User Request Size	0.401	(-0.0024, 0.0024)
Web Client Request Size	0.735	(-0.0005, 0.0005)
Cached Web User Response Size	0.3774	(-0.0063, 0.0063)
Non-Cached Web User Response Size	0.2285	(-0.0035, 0.0035)
Cached Web Client Response Size	0.7614	(-0.0009, 0.0009)
Non-Cached Web Client Response Size	0.254	(-0.0007, 0.0007)

Table 16: Autocorrelation function at lag 1 ($\gamma(1)$) for the 11 parameter datasets

We calculated the autocorrelation at lag 1 to lag 40. The autocorrelation of a random variable is typically greatest at lag 1. All the parameters showed correlation across the range of different lags. Table 16 reports the autocorrelation at lag 1 for all the parameter datasets. The autocorrelation at lag 1 was weak for most of the parameters. The `Web User` and `Web Client Request Size` parameters, and the `Cached Web User` and `Cached Web Client Response Size` parameters showed strong correlation. The strong autocorrelation of the `Request Size` parameters could be ascribed to the fact that request message sizes did not vary much. A request message consisted of a packet header without a message body. Request headers had a fairly standard size. The strong autocorrelation of the `Cached Response Size` parameters can also be ascribed to the fact that cached response messages had a standard size. Cached response messages consisted of a packet header containing a `304 Not Modified` response code indicating that the object in the cache should be used. Replies of this type consisted of an HTTP header with no message body. Response messages were small and of standard size.

None of the parameters had autocorrelation function values at lag 1 within the 95% confidence interval which signified no correlation in the data. Apart from the parameters discussed in the previous paragraph, the parameters were all weakly correlated. We concluded that more investigation should be done to reduce interdependence between elements in the same datasets. We attempted to reduce the interdependence between parameters by choosing parameters carefully. We did not manage reduce interdependence enough. We found that automated web scripts (client and server side) were the principal source for the interdependence in the data. A refinement of our model is necessary to take these factors into account. Our model is sufficiently descriptive to ensure that correlations are weak for most of the model parameter.

Parameters which displayed very high correlation consisted of datasets which had values of similar

sizes and could be modelled by constant values.

6.3 Previous Work

We gained insight into what we might expect to find in our analysis by studying results of previous work done in the field. Work done by other authors provided us with an invaluable source of information. We used their results as a "sanity check" for the results we found. Our results were different to those of previous studies in many cases. Prompted by these differences we investigated data further, and often found interesting reasons for the differences in results.

Several authors used the **structural approach** to network traffic modelling to find distributions for web workload model parameters. In order to find distributions for their model parameters they used statistical techniques similar to the ones we used. Choi et. al. [CL99] created a "behavioural model" of web traffic by finding mathematical functions for their model parameters. Mah [Mah97] created an "empirical model" of web traffic by finding empirical distribution functions for his model parameters. Barford et. al. [BC98b] created a web workload generator which exercised web servers and test-bed networks by finding mathematical functions for their model parameters. Reyes-Lecuona et. al. [RLGPC+99] and Arlitt et. al. [AW95] created web traffic models for simulation purposes by finding mathematical functions for their model parameters.

Choi et. al.'s [CL99] web traffic model comprised seven parameters which corresponded to parameters in our model. Apart from the similarities, there were several differences between their model and ours. For instance, their model did not have `Browsing Inter-Session Times` or `Web User Request Inter-arrival Times` parameters, which our model did. Their model did not differentiate between `Web User` and `Web Client Request Size` parameters, which our model did. Their work however made an important differentiation between cached and non-cached `Web User Requests` which our model did not make. We updated our model after realising that it was important to take caching into account. We discuss how we updated our model in Sections 6.11 and 6.12 (pages 113 and 118).

The model by Mah [Mah97] also comprised several parameters which corresponded to parameters in our workload model. Mah's model was however empirical and we could therefore not compare his model to ours as our model was based on mathematical functions. The advantage of the empirical approach that Mah adopted was that traffic could be reproduced accurately, closely matching original traffic. The empirical method however required more storage and was slower at generating random values than using analytic distributions were. We decided to use the approach of using mathematical functions to represent parameters. Mathematical functions provided a precise way of describing traffic characteristics, and could be used to compare our results to results found in other studies. The mathematical modelling of parameters also enabled us to investigate important mathematical properties of parameters, such as the occurence of heavy-tailed distributions.

The traffic model by Barford et. al. [BC98b] had a different purpose to our model. The model by Barford et. al. was used to exercise web servers and test-bed networks by generating web requests. Our model had only three parameters in common with the model of Barford et. al.

The models by Reyes-Lecuona et. al. [RLGPC+99] and Arlitt et. al. [AW95] did not have much in common with our model. The models by Reyes-Lecuona et. al. and Arlitt et. al. modelled traffic as **traffic flows** at the TCP/IP packet level. Parameters derived at the network level were influenced by underlying network conditions such as bandwidth and latency. We therefore did not model traffic at the network level.

Our model was very different to all the models created in previous work. The models by Choi et. al. and Barford et. al. were the closest to ours in terms of the number of similar workload model parameters. The model by Choi et. al. had seven parameters which matched parameters of our workload model. We next discuss the mathematical functions which were found to match model parameters of the workload models created in previously discussed studies.

6.3.1 Distributions of Model Parameters

Tables 17-19 list the mathematical functions and model constants found for workload model parameters in previous studies. We listed only the parameters that corresponded to parameters in our workload model in the tables.

Author	**Browsing Inter-Session Time**	**Number of Web User Requests per Browsing Session**	**Number of Web Client Requests per Web User Request**
Choi et. al. [CL99]	—	**Non-cached Lognormal:** $mean = 12.6$ $std.dev. = 21.6$ $median = 5$ **Cached Gamma:** $mean = 1.7$ $std.dev. = 1.7$ $median = 1$	**Gamma:** $mean = 5.55$ $std.dev. = 11.4$ $median = 2$
Barford et. al. [BC98b]	—	—	**Pareto:** $\alpha = 1$ $\beta = 2.43$
Reyes-Lecuona et. al. [RLGPC+99]	—	**Lognormal:** $mean = 22.975$ $std.dev. = 166.16$	—
Arlitt et. al. [AW95]	—	**Geometric:** $mean = 50$	—

Table 17: Mathematical Functions Found for Model Parameters in Previous Studies - 1

Tables 17-19 show that few of the parameters used in previous studies corresponded to parameters in our workload model. The work of Choi et. al. contained more parameters, that were the same as our workload model parameters, than any of the other previous studies. The parameters of Mah's

Author	**Web User Request Inter-arrival Time**	**Web Client Request Inter-arrival Time**	**Web User Request Size**
Choi et. al. [CL99]	—	**Gamma:** (*measured in seconds*) *mean* = 0.86 *std.dev.* = 2.15 *median* = 0.17	**Lognormal:** (*measured in bytes*) *mean* = 360.4 *std.dev.* = 106.5 *median* = 344
Arlitt et. al. [AW95]	**Exponential:** *no parameters measured*	—	—

Table 18: Mathematical Functions Found for Model Parameters in Previous Studies - 2

Author	**Web Client Request Size**	**Web User Response Size**	**Web Client Response Size**
Choi et. al. [CL99]	**Lognormal:** (*measured in bytes*) *mean* = 360.4 *std.dev.* = 106.5 *median* = 344	**Lognormal:** (*measured in bytes*) *mean* = 10 710 *std.dev.* = 25 032 *median* = 6 094	**Lognormal:** (*measured in bytes*) *mean* = 7 758 *std.dev.* = 126 168 *median* = 1 931
Barford et. al. [BC98b]	—	**Pareto:** (*measured in bytes*) α = 1 β = 1 000	**Pareto:** (*measured in bytes*) α = 1 β = 1 000

Table 19: Mathematical Functions Found for Model Parameters in Previous Studies - 3

workload model were not reported in the tables as Mah's model used empirical distribution functions instead of mathematical functions.

The `Browsing Inter-Session Time` parameter was not modelled in previous work. Choi et. al. and Reyes-Lecuona et. al. modelled the `Number of Web User Requests per Browsing Session` parameter. Choi et. al. found that the lognormal distribution with a standard deviation of 22 fitted the parameter best. Choi et. al. also found that on average, approximately 13 web user requests were placed per browsing session. Reyes-Lecuona et. al. found that the lognormal distribution with a standard deviation of 165 fitted the parameter best. Reyes-Lecuona et. al. found that on average, approximately 23 web user requests were placed per browsing session. The mean value of 23 found by Reyes-Lecuona et. al. was much larger than the average value of 13 found by Choi et. al. The larger value found by Reyes-Lecuona et. al. seemed to confirm our initial observation that people typically browsed the web for extended periods of time. We however found this not to be the case. The average number of requests per browsing session turned out to be 12 according to our study. The value found by Choi et. al. reflected current browsing behaviour of users better than that of Reyes-Lecuona et. al.

Choi et. al. and Barford et. al. modelled the `Number of Web Client Requests per Web User Request` parameter. Choi et. al. found that the gamma distribution with a standard deviation of 11 fitted the parameter best. Choi et. al. found that on average, 6 web client requests were placed for every web user request. Web-pages typically contained a great deal of graphical content. It was therefore surprising that Choi et. al. found such a small number of in-line objects per web page. We found the parameter to have an average value of 20 web client requests per web user request. 20 Web client requests per web user request seems to be in keeping with the large numbers of graphical images per web-page we observed during the study. Barford et. al. found the parameter to have the Pareto distribution.

Arlitt et. al. were the only ones to model the `Web User Request Inter-arrival Time` parameter. They found the parameter to have an exponential distribution, but did not find any values for the model constants.

Choi et. al. modelled the `Web Client Request Inter-arrival Time` parameter. They found the inter-arrival times between web-client requests to have an average value of 0.86 seconds with standard deviation 2.15 seconds and a gamma distribution. The small average value of 0.86 seconds was attributed to a web client typically placing consecutive requests very rapidly. The average value of the parameter was quite surprising, as we expected the value to be smaller. Observations we made of inter-arrival times suggested that the average value was in the order of microseconds, rather than milliseconds. The final results of our study however surprised us. The average value of the parameter as found in our study was 1.5 seconds. Further investigation into the large average value showed that the reason for value being large was the severe skew in the distribution of inter-arrival time values. Our study showed that the `Web Client Request Inter-arrival Time` parameter was severely skewed. It was categorised as a heavy-tailed distribution.

Choi et. al. modelled the `Web User Request Size` and `Web Client Request Size` parameters. They found that the two parameters had the same characteristics, and therefore modelled both parameters with the lognormal distribution. Choi et. al. found that both web user and web client request sizes had an average value of 360 bytes with standard deviation of 107 bytes. We also found that the two parameters were very similar. We however modelled the two parameters separately, as we found model constants for the two parameters to be different.

The reason why web user and web client request sizes were small relative to response sizes, was that requests did not contain message bodies, whereas responses did. Requests consisted out of an HTTP header which contained the requested URL.

Choi et. al. found web user response sizes to have an average size of 10710 bytes with a standard deviation of 25032 bytes. Web user responses were usually HTML text files.

Choi et. al. found web client response sizes to be smaller than web user response sizes. Web client response sizes were found to have an average size of 7758 bytes. Web client response sizes however had a larger standard deviation than web user response sizes - 126168 bytes. Web client responses were usually image files. Choi et. al. found both the `Web User Response Size` and `Web Client Response Size` parameters to have a lognormal distribution.

It was interesting to note that Choi et. al. found request sizes to be larger on average than response sizes. They however found response sizes to sometimes be much larger than requests sizes, as indicated by the much larger standard deviation of response sizes.

We next discuss the analysis of the `Browsing Inter-Session Time` parameter. Thereafter we discuss the analyses of the remaining 10 parameters.

6.4 Browsing Inter-Session Time

Section 2.6 (page 27) defined a browsing session as the period during which a user browsed the web. The `Browsing Inter-Session Time` parameter modelled the time between browsing sessions.

We measured the parameter at minute resolution. We found that second resolution was too small and hour resolution too large. The parameter had values larger than 15 minutes as a browsing session was terminated when a 15 minute period of inactivity i.e. no user clicks, was detected. The parameter had a maximum value of 480 minutes because we modelled traffic during office hours only.

We aggregated the 715 parameter datasets, under the assumption that user behaviour in terms of the time between browsing sessions was the same for all hosts.

Table 20 shows summary statistics for the aggregated parameter dataset.

Sample Size	43336
Five Number Summary	(15, 24.7, 45.89, 100.5, 479.7)
Sample Mean	81.04
Sample Variance	7586.723
Standard Deviation	87.1018
Coefficient of Variation	1.075
Skewness	2.16
Kurtosis	4.766
Upper Outlier	373

Table 20: Summary Statistics for `Browsing Inter-Session Time` Parameter Dataset

The five number summary shows values for the smallest number, the first quartile, the median, the upper quartile and the largest number. The range between the smallest value and the median was 30.89 minutes while the range between the median and the largest value was 433.81 minutes. The difference in size between these ranges indicated a skew to the right in the data. The skewness and kurtosis values confirmed this. The skewness was 2.16 (positive values indicated skew to the right) and the kurtosis was 4.766 which indicated a skew to the right in the data as well as indicating that the data were centred closely around the mean. The coefficient of variation was the ratio between the standard deviation and the mean and was a metric of variability in the data. The value of the coefficient of variation was 1.075 which indicated that there was a great deal of variability in the data e.g. the exponential distribution had a coefficient of variation of 1.

The summary statistics showed that the distribution was skewed to the right and that the

exponential distribution might have been a good fit to the data. The statistics also showed that users working at hosts took significant breaks between browsing sessions. The average time between two sessions was nearly one and a half hours. It was more likely that a break would be smaller than one and half hours, but larger breaks did occur regularly. Approximately 25% of breaks were smaller than half an hour and approximately 25% were in the interval of 2-8 hours, indicating that there was a significant chance of users taking breaks much larger than the average sized break of one and a half hours.

We aligned the data to zero by subtracting 14 minutes from all observations as explained in Section 5.5.2 (page 74).

Table 21 shows the results of applying the λ^2 discrepancy measure to the data. We only show results for distributions which passed the λ^2 statistic test i.e. tests which resulted in negative values, as explained in Section 5.5.2. All distributions not shown in the table failed the test and therefore did not provide a good fit to the data.

Distribution	**Parameters**	λ^2	**Number of Bins**
Lognormal	$\zeta = 3.242876$ $\sigma = 1.637514$	0.042 $(0.038, 0.046)$	459
Exponential	$\alpha = 66.041138$	0.295 $(0.277, 0.314)$	366
Weibull	$\gamma = 0.7437$ $\alpha = 54.902209$	0.026 $(0.022, 0.03)$	425
Gamma	$\gamma = 0.645296$ $\alpha = 102.342467$	0.05 $(0.044, 0.055)$	414
Pareto	$\alpha = 9.6e-05$ $\beta = 0.08005$	5.054 $(4.929, 5.178)$	330
Beta	$\alpha = 0.51966$ $\beta = 2.762956$	0.284 $(0.244, 0.323)$	401
Extreme Value	$\alpha = 32.933427$ $\beta = 46.746046$	1.579 $(1.491, 1.666)$	312
Normal	$\mu = 66.041137$ $\sigma = 87.101794$	3.581 $(3.384, 3.778)$	324

Table 21: Lambda Discrepancy Test Results for Web Browsing Inter-Session Data over the Interval of $(1, 465)$ Minutes

We fitted the data to analytic distributions over the interval of $(0,\ 465)$ minutes. We obtained maximum likelihood estimates of distribution parameters by maximising the likelihood function (on logarithmic scale) of the data using techniques explained in Section 5.2.

The gamma, Weibull and lognormal distributions provided the best fit to the data in that order.

The most bins were used in the calculation of the lognormal, Weibull and gamma λ^2 values. A smaller number of bins used in the calculation of λ^2 was attributed to adjacent bins being grouped together for bins containing less than 5 entries, which in our case happened when the analytic

distribution did not model the upper tail of a positively skewed distribution well. The lognormal, Weibull and gamma distributions modelled the upper tail of the empirical distribution better than the other distributions.

Figure 24 is a plot of the gamma, Weibull and lognormal distributions fitted to a histogram of the data.

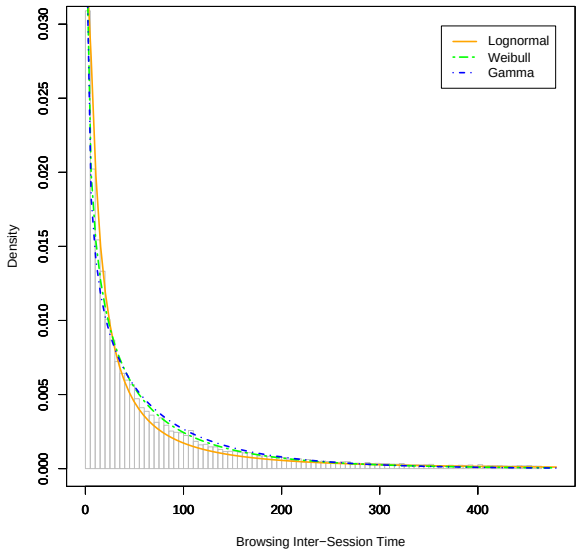


Figure 24: Best-Fit Distributions Plotted against a Histogram of `Browsing Inter-Session Time` Data

The distributions provided a very good fit to the data. The Weibull and gamma distributions modelled the body of the data very well. The Weibull distribution underestimated the upper tail of the distribution i.e. values between three and six hours. The lognormal distribution modelled the upper tail of the empirical distribution containing values larger three hours well.

Figure 25 shows Q-Q and P-P plots for the Weibull and lognormal distributions. Straight lines were fitted to the plots. The regression statistics are shown in Table 22.

The straight lines fit the empirical data very well. The regression statistics shown in Table 22 showed a very good fit to the data (a value close to zero indicated a good fit). 75% Of the data fell in the interval $(0, 260)$. The lower end of the straight line fitted the data very well for both the Weibull and lognormal distributions. The Weibull distribution underestimated the upper tail of the distribution according to the Q-Q plot.

The Weibull and lognormal distributions were good candidates for random number generation for the `Web Browsing Inter-Session Time` parameter.

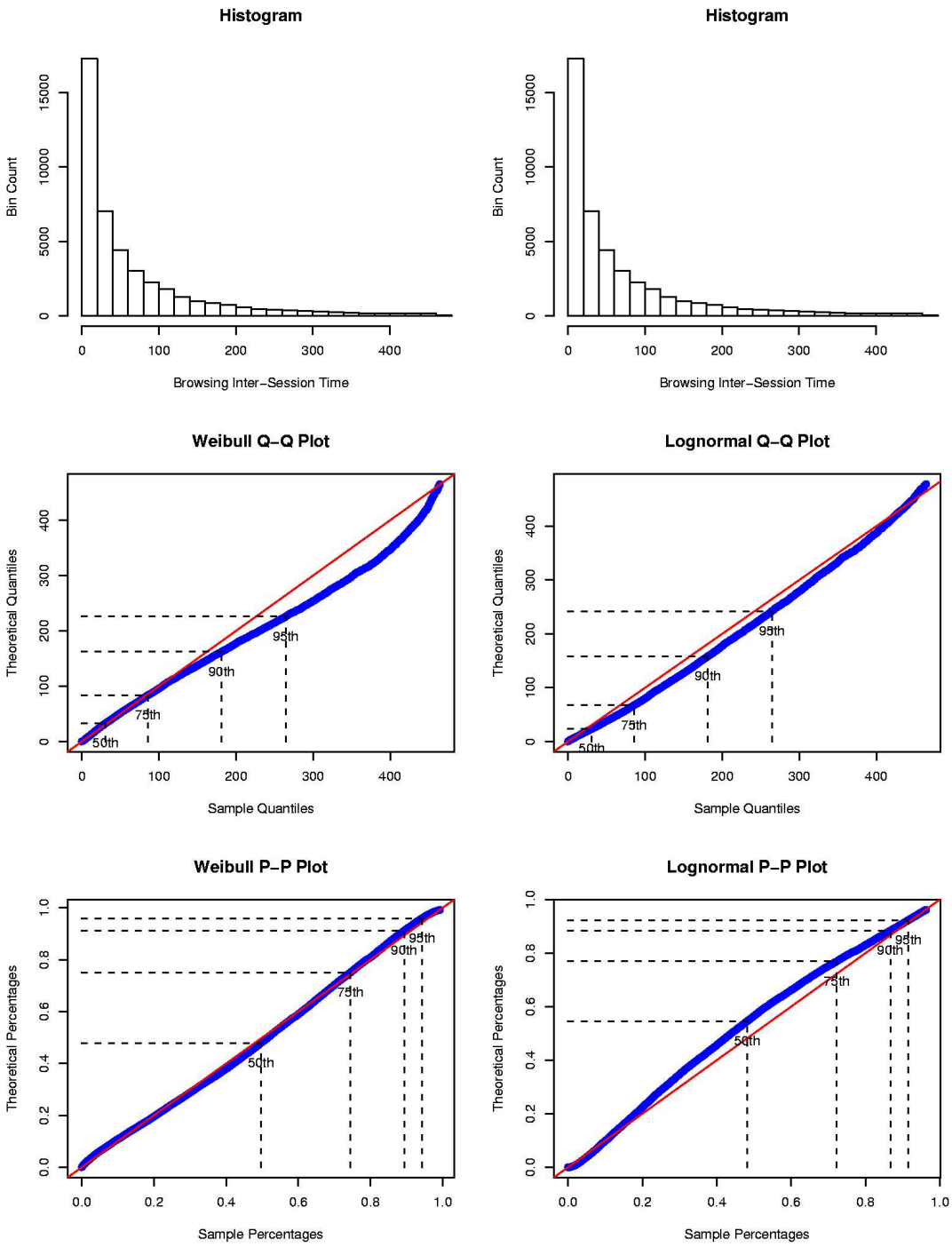


Figure 25: Weibull and Lognormal Plots for `Browsing Inter-Session Time` Parameter

Distribution	Regression Statistics
Weibull	-0.0148
Lognormal	-0.014

Table 22: Regression Statistics for Weibull and Gamma Q-Q Plots

6.5 Number of Web User Requests per Browsing Session

Section 2.6 defined the `Number of Web Requests per Session` as the number of times a user clicked on a hypertext link or entered an URL in a web browser's address text box during a Browsing Session.

The minimum and maximum values the parameter could have were 2 and 100 requests. We found that 1 request per session usually meant that the request was generated by an automatic updating mechanism such as a news-ticker or web photo-gallery update. We found that more than a 100 requests per session usually indicated that the requests were automatically generated by a web-download. These limits were implemented as such in the heuristic algorithm used to categorise data during the processing stage.

Table 23 shows summary statistics for the aggregated parameter dataset.

Sample Size	53 621
Five Number Summary	(2, 3, 7, 15, 100)
Sample Mean	12.22
Sample Variance	192.4179
Standard Deviation	13.8715
Coefficient of Variation	1.132
Skewness	2.542
Kurtosis	7.973
Upper Outlier	55

Table 23: Summary Statistics for `Number of Web User Requests per Browsing Session` Parameter Dataset

The five number summary again indicated a skew to the right in the data. The range between the smallest value and the median was 5 requests and the range between the median and the largest value was 73 requests. The values for skewness and kurtosis again confirmed the skew to the right in the data. The value of the kurtosis was larger than that for the `Browsing Inter-Session Time` parameter, indicating that the data were distributed closer around the mean. The coefficient of variation again indicated a great deal of variability in the data.

75% of browsing sessions consisted out of less than 15 requests. 25% of browsing sessions consisted out of less than 3 requests. The average user on campus usually read a small number of web pages. (25%) of users visited between 15 and 100 web pages. A small proportion of users however read a large number of pages.

Our findings corroborated the findings of Choi et. al. [CL99] whom also found that the average user visited a small number of web pages during a browsing session. They found the mean number of requests per session to be approximately 13 which was similar to the value we found.

The high amount of variability in the data and severe skew to the right suggested that the data were heavy tailed.

We aligned the data to zero by subtracting 1 request from all observations. Table 24 shows the results of applying the λ^2 discrepancy measure test to the data.

Distribution	**Parameters**	λ^2	**Number of Bins**
Lognormal	$\zeta = 1.792736$ $\sigma = 1.147842$	0.209 $(0.199, 0.219)$	98
Exponential	$\alpha = 11.222189$	0.48 $(0.461, 0.499)$	83
Weibull	$\gamma = 0.912485$ $\alpha = 10.681936$	0.36 $(0.345, 0.375)$	90
Gamma	$\gamma = 0.931245$ $\alpha = 12.050697$	0.428 $(0.411, 0.446)$	86
Pareto	$\alpha = 1$ $\beta = 0.557807$	0.204 $(0.196, 0.213)$	98
Beta	$\alpha = 0.777341$ $\beta = 5.164118$	0.445 $(0.411, 0.479)$	77
Extreme Value	$\alpha = 6.087854$ $\beta = 7.282397$	2.109 $(1.999, 2.219)$	64
Normal	$\mu = 11.222189$ $\sigma = 13.871479$	4.898 $(4.637, 5.16)$	60

Table 24: Lambda Discrepancy Test Results for Number for Web User Requests per Browsing Session Data over the Interval of $(1, 99)$ Minutes

The data were fitted to mathematical functions over the interval of $(1, 99)$ minutes.

The Extreme Value and Normal distributions did not fit the data well. All other distributions fit the empirical data reasonably well. The Pareto, lognormal, Weibull and beta distributions provided the best fit in that order. The Pareto, lognormal and Weibull distributions provided the best fit to the tail of the distribution according to the number of bins used in calculating λ^2.

Figure 26 is a plot of the Pareto, beta and Weibull distributions fitted to a histogram of the data.

Small numbers of requests, less than 3, were modelled well by the Pareto distribution. The beta distribution also modelled these values well, but the Weibull distribution underestimated them. The Weibull and gamma distribution modelled the body of the data in the interval $(2, 20)$ requests well. The Pareto distribution underestimated these values. The upper tail of the data was modelled well by the Pareto, lognormal and Weibull distributions. Figure 26 suggested that the upper tail of the distribution decays at a sub-exponential rate.

Figure 27 shows Q-Q and P-P Plots for the Pareto and lognormal distributions.

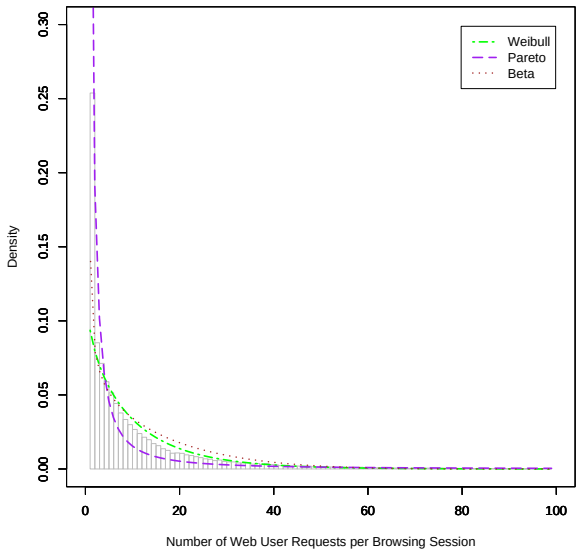


Figure 26: Best-Fit Distributions Plotted against a Histogram of `Number of Web User Requests per Browsing Session` Data

The data were heavily concentrated around smaller values of the parameter. The lower 15% of the range of the data contained 75% of the observations. The interval (0, 40) contained 95% of the data. The lognormal distribution modelled the data very well. The Pareto distribution underestimated the empirical data over the interval (5, 22), and overestimated the data over the interval (45, 100).

The regression statistics in Table 25 indicated that the mathematical functions model the data reasonably well. The `S` trend in the Pareto distribution plots were caused by the large number of small values combined with the heavy upper tail of the empirical distribution.

Distribution	Regression Statistics
Pareto	-0.066
Lognormal	-0.006

Table 25: Regression Statistics for Pareto and Lognormal Q-Q Plots

The Pareto and lognormal distributions were good candidates for random number generation for the `Number of Web User Requests per Browsing Session` parameter.

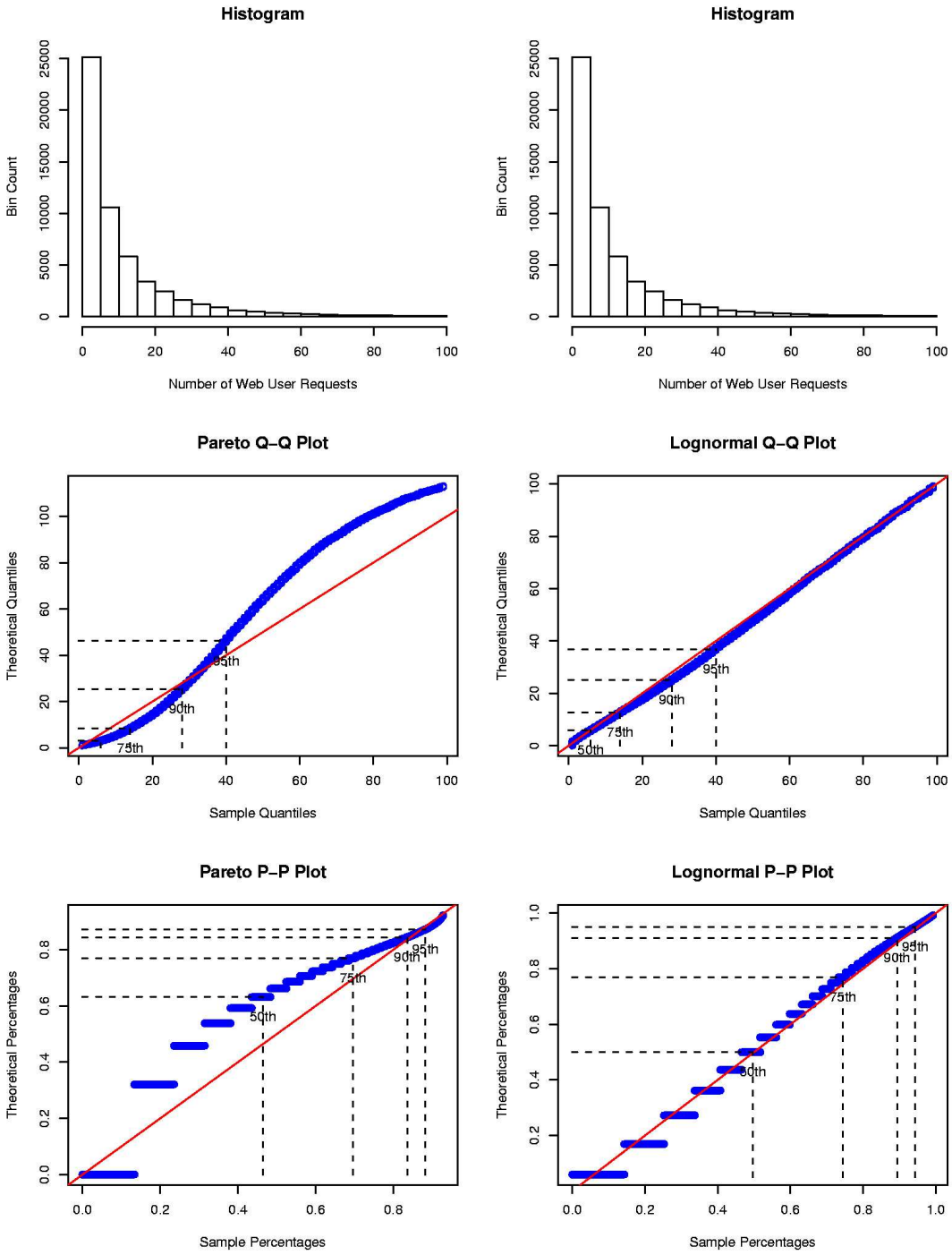


Figure 27: Pareto and Lognormal Plots for `Number of Web User Requests per Browsing Session` Parameter

6.6 Number of Web Client Requests per Web User Request

Section 2.6 defined the `Number of Web Client Requests per Web User Request` as the number of inline objects contained in a web-page.

We observed that web pages seldomly had more than 200 inline objects. A web page which generated requests for more than 200 inline objects was likely to be a web-download. The heuristic algorithm categorised web pages with more than 200 inline objects as being web downloads and discarded these requests along with their responses.

We did not align the data with zero as the smallest value in the aggregated dataset was one. Table 26 shows summary statistics for the aggregated parameter dataset.

Sample Size	703084
Five Number Summary	(1, 3, 9, 26, 200)
Sample Mean	19.687
Sample Variance	704.936
Standard Deviation	26.551
Coefficient of Variation	1.349
Skewness	2.68
Kurtosis	9.286
Upper Outlier	111

Table 26: Summary Statistics for `Number of Web Client Requests per Web User Request` Parameter Dataset

The five number summary again indicated a skew to the right. The range between the smallest value and the median was 8 requests and the range between the median and the maximum value was 191 requests. The values for skewness and kurtosis confirmed the skew to the right in the data. The very high value for kurtosis indicated that the data were distributed closely around the mean. The coefficient of variation indicated high variability in the data.

We observed many web pages with more than 50 inline objects during the processing of the data. The web request shown in Appendix A shows a typical request for a web page (`www.cnn.com`), it had 82 inline objects. The value of the median and mean were unexpectedly low.

A large percentage of web pages had a small number of inline objects. 50% of web pages had between 1 and 9 inline objects. There were however a significant number of web pages that had a much larger number of inline objects. 50% of web pages had between 9 and 200 inline objects. The skew to the right in the distribution of data suggested that the distribution had a heavy tail.

Our findings were different to those of Choi et. al. [CL99]. They found the mean number of inline objects per web page to be approximately 6. A possible reason for the different findings might have been the simple heuristic they used to distinguish between web user and web client requests. The heuristic they used did not take into account the fact that a request for a file with HTML content might have been a web client request. Consequently web client requests might have been wrongly categorised as being web user requests. If this error occurred, subsequent web client

requests for inline objects would also have been wrongly associated with the wrongly classified web user request. The number of inline objects per web page would therefore have been underestimated as a consequence.

Table 27 shows the results of applying the λ^2 discrepancy measure test to the data.

Distribution	**Parameters**	λ^2	**Number of Bins**
Lognormal	$\zeta = 2.154982$ $\sigma = 1.377062$	0.294 $(0.29, 0.297)$	199
Exponential	$\alpha = 19.686967$	0.893 $(0.883, 0.903)$	184
Weibull	$\gamma = 0.793412$ $\alpha = 17.067434$	0.417 $(0.413, 0.422)$	199
Gamma	$\gamma = 0.728499$ $\alpha = 27.024945$	0.5 $(0.494, 0.505)$	199
Pareto	$\alpha = 1$ $\beta = 0.464041$	0.325 $(0.322, 0.329)$	199
Beta	$\alpha = 0.68239$ $\beta = 5.113663$	0.901 $(0.854, 0.948)$	171
Extreme Value	$\alpha = 9.937799$ $\beta = 13.784351$	3.83 $(3.757, 3.903)$	146
Normal	$\mu = 19.686938$ $\sigma = 26.569469$	12.816 $(12.507, 13.126)$	127

Table 27: Lambda Discrepancy Test Results for `Number of Web Client Requests per Web User Request` Data over the interval $(1, 200)$

The data were fitted to analytic distributions over the interval $(1, 200)$.

The λ^2 discrepancy test combined bins with less than 5 entries as discussed in Section 5.5.2. Too many combined bins indicated a poor fit to the analytic distribution. We did not include the results of the extreme value distribution in Table 27 as too many bins were combined in the test.

All the distributions except for the normal and extreme value distributions fitted the data. The lognormal, Pareto and Weibull distributions modelled the empirical data best. All three distributions modelled the tail of the distribution well according to the number of bins used to calculate λ^2.

Figure 28 is a plot of the lognormal, Pareto and Weibull distributions fitted to a histogram of the data.

Small numbers of requests for inline objects, less than 3 inline objects, were modelled well by the Pareto distribution. Neither the Weibull nor the lognormal distribution modelled the small values of the parameter well. The Pareto distribution underestimated the value in the body of the distribution, between 3 and 40 inline objects. The lognormal and Weibull distributions modelled these values well. All three distributions modelled the upper tail, 40 inline objects and more, well. The heavy upper tail contributed to the high degree of variation in the data, as indicated by the coefficient of variation.

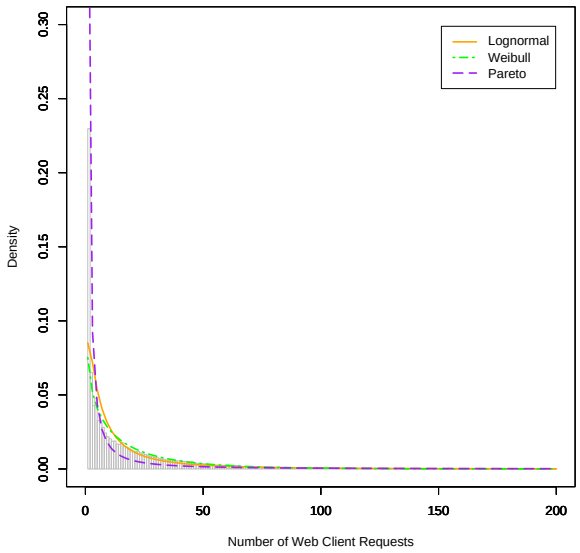


Figure 28: Best-Fit Distributions Plotted against a Histogram of `Number of Web Client Requests per Web User Request` Data

Figure 29 shows a Q-Q plot for the lognormal and a Probability Plot for the Pareto distribution.

The straight line fitted the lognormal Q-Q plot very well over the whole range of data. The straight line for the Pareto Q-Q Plot showed deviation from the empirical data for values in the body of the data i.e. values in the interval $(4,\ 40)$. The Pareto distribution underestimated values in the interval $(4,\ 40)$ and overestimated values in the interval $(75, 200)$. The Pareto distribution however modelled the large number of very small values well.

The regression statistics are shown in Table 28. The lognormal distribution provided a very good fit to the data. The Pareto distribution modelled the data reasonably well.

Distribution	Regression Statistics
lognormal	-0.008
Pareto	-0.107

Table 28: Regression Statistics for Lognormal and Pareto Q-Q Plots

The lognormal and Pareto distributions were good candidates for random number generation for the `Number of Web Client Requests per Web User Request` parameter.

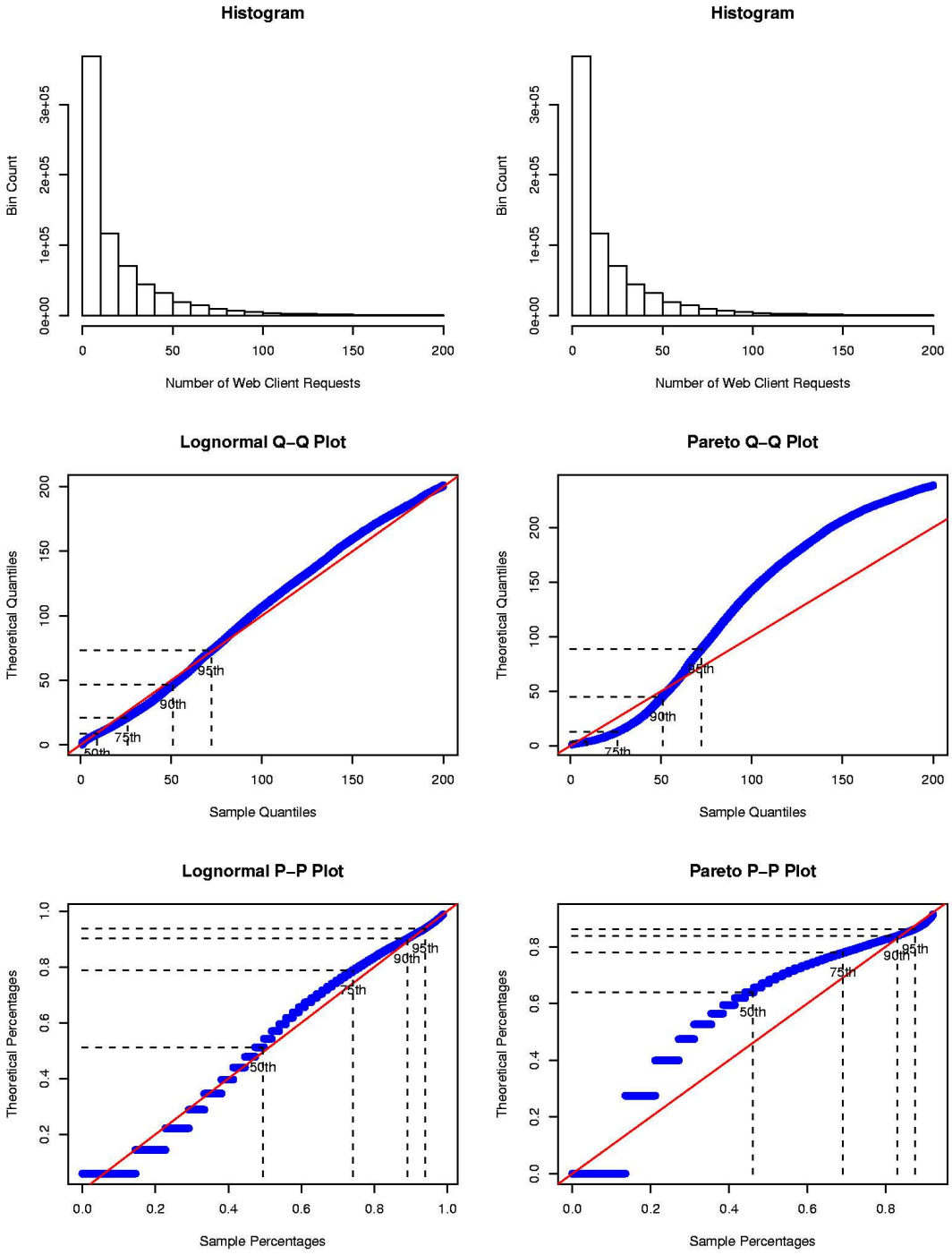


Figure 29: Pareto and lognormal Plots for `Number of Web Client Requests per Web User Request` Parameter

6.7 Web User Request Inter-arrival Time

Section 2.6 defined web user request inter-arrival time as being the time between requests for web-pages. The `Web User Request Inter-arrival Time` parameter models this quantity.

The model parameter was measured in seconds, and could take on a value in the interval $(1, 900)$ seconds. We observed that inter-arrival times greater than 15 minutes (900 seconds) were usually caused by the start of a new browsing session. The heuristic algorithm categorised inter-arrival times greater than 15 minutes as such.

We did not align the data with zero as the smallest value in the aggregated dataset was 1. Table 29 shows summary statistics for the aggregated parameter dataset.

Sample Size	653466
Five Number Summary	(1, 26, 73, 141, 900)
Sample Mean	115.427
Sample Variance	19780.099
Standard Deviation	140.642
Coefficient of Variation	1.218
Skewness	2.571
Kurtosis	7.875
Upper Outlier	484

Table 29: Summary Statistics for `Web User Request Inter-arrival Time` Parameter Dataset

The five number summary indicated a skew to the right. The range between the smallest value and the median was 72 seconds, and range between the median and the maximum value was 827. The values for skewness and kurtosis confirmed the skew to the right in the data.

The mean value indicated the value for the average time between two user requests to be a little less than 2 minutes. This meant that users spent on average less than 2 minutes reading a web page before moving on to another web page. We observed that users usually spent little time on a single web-page. Web-pages were designed to keep the user's attention and usually did not contain a great deal of information on a single page. They presented the user with small amounts of information at a time, with hyper-links to more information.

There were a significant number of inter-arrival times which were much larger than 2 minutes. 25% of inter-arrival times fell in the interval (141, 900) seconds, indicating a large likelihood of an inter-arrival time having a value of up to 15 minutes. The large value for the coefficient of variation confirmed the large likelihood of an inter-arrival time varying substantially from the mean.

Table 30 shows the results of applying the λ^2 discrepancy measure test to the data.

The Pareto, beta, extreme and normal distributions did not model the data well. All the other mathematical functions modelled the data well over the interval $(1, 900)$ seconds. The Weibull, gamma and exponential distributions modelled the data best. The Weibull, gamma and exponential distributions modelled the tail of the distribution well according to the number of bins used to calculate λ^2.

Distribution	Parameters	λ^2	Number of Bins
Lognormal	$\zeta = 4.001863$ $\sigma = 1.478699$	0.155 $(0.152, 0.157)$	900
Exponential	$\alpha = 115.426519$	0.153 $(0.15, 0.157)$	854
Weibull	$\gamma = 0.85689$ $\alpha = 106.420313$	0.065 $(0.063, 0.066)$	900
Gamma	$\gamma = 0.795353$ $\alpha = 145.127336$	0.078 $(0.076, 0.08)$	900
Pareto	$\alpha = 0.000174$ $\beta = 0.078999$	6.98 $(6.934, 7.026)$	900
Beta	$\alpha = 0.644358$ $\beta = 3.95013$	1.382 $(1.298, 1.466)$	789
Extreme Value	$\alpha = 63.486574$ $\beta = 75.045753$	1.775 $(1.724, 1.826)$	680
Normal	$\mu = 115.426513$ $\sigma = 140.645402$	11.676 $(11.3, 12.052)$	628

Table 30: Lambda Discrepancy Test Results for `Web User Request Inter-arrival Time` Data over the interval $(1, 900)$

Several studies had shown that the exponential distribution was a poor model for inter-arrival times of TCP connections and HTTP requests. The exponential distribution however often provided a good model for parameters involving human behaviour. The `Web User Request Inter-arrival Time` parameter modelled the behaviour web users by measuring the time between users clicking on hyper-links. The exponential distribution provided a good fit to the `Web User Request Inter-arrival Time` parameter data. This was consistent with past results.

Figure 30 is a plot of the Weibull, gamma and exponential distributions fitted to a histogram of the data.

The three mathematical functions modelled the data well according to the histogram. The Weibull and gamma distributions modelled the large number of inter-arrival times with value less than 3 seconds more accurately than the exponential distribution.

Figure 31 shows Q-Q plots for the Weibull and gamma distributions.

The Q-Q plots looked similar to the ones for the `Browsing Inter-session Time` parameter, the lower parts of the plots fitted the straight lines very well and the upper parts deviated from the lines. The Weibull and gamma distributions underestimate the upper tail of the empirical distribution. 95% Of the data is very well modelled by the two mathematical functions. The straight lines lines fitted the Q-Q plots well for inter-arrival time values in the interval $(0, 400)$ seconds which accounted for approximately 95% of the data. The regression statistics in Table 31 confirmed that both mathematical functions modelled the data well.

The Weibull and gamma distributions were good candidates for random number generation for

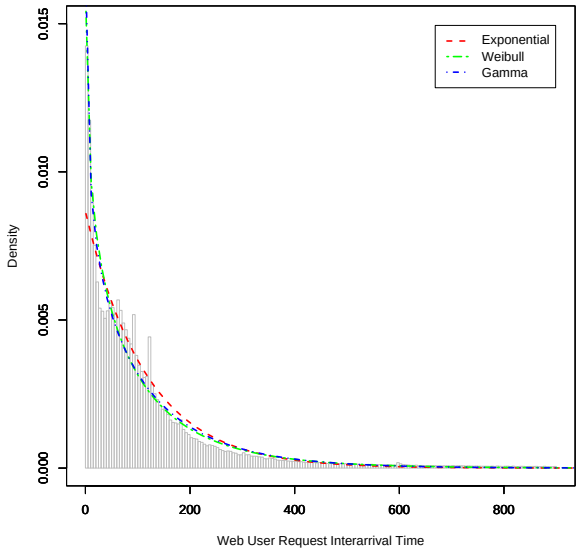


Figure 30: Best-Fit Distributions Plotted against a Histogram of `Web User Request Inter-arrival Time` Data

Distribution	Regression Statistics
Weibull	-0.014
Gamma	-0.02

Table 31: Regression Statistics for Weibull and Gamma Q-Q Plots

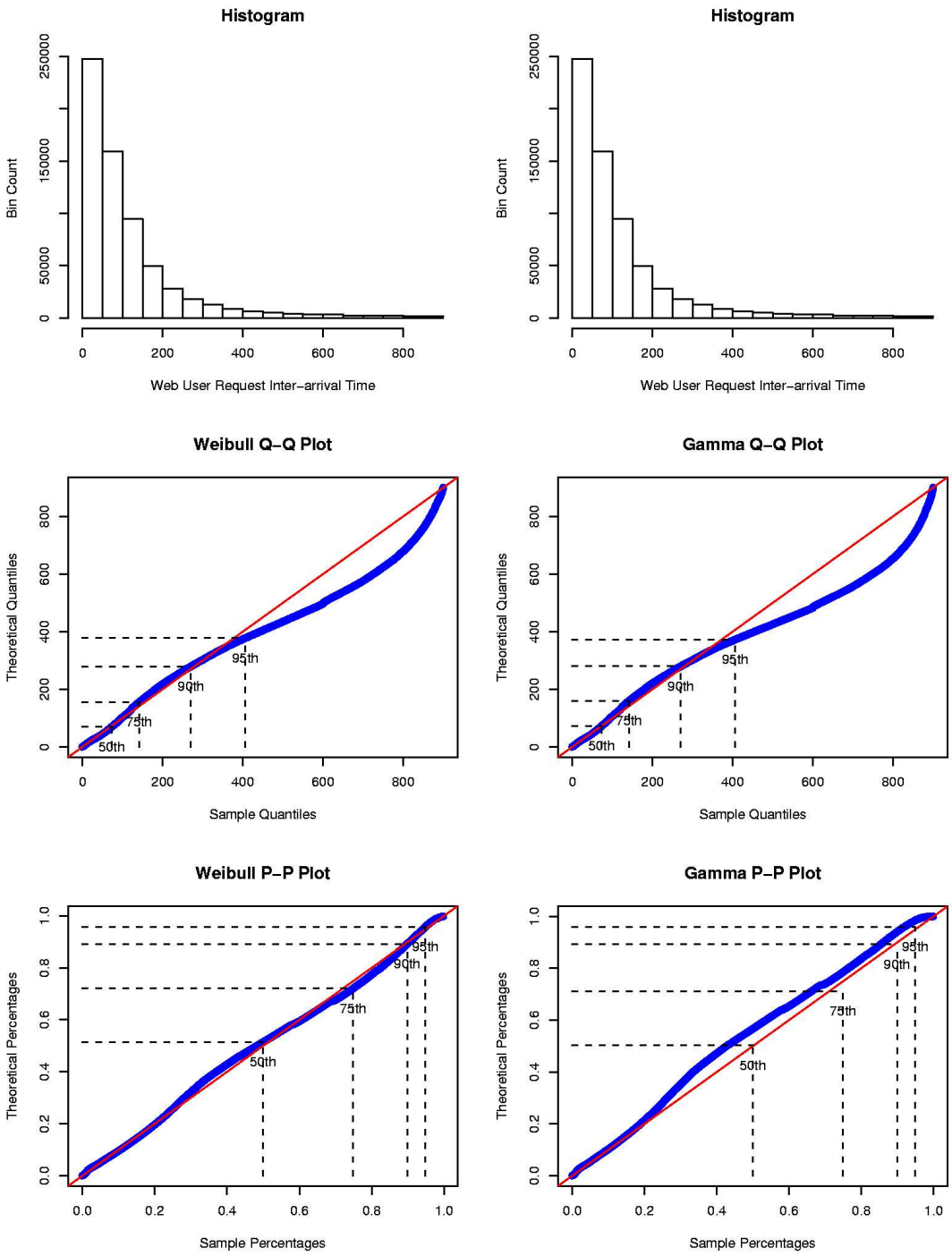


Figure 31: Weibull and Gamma Plots for `Web User Request Inter-arrival Time` Parameter

the `Web User Request Inter-arrival Time` parameter.

6.8 Web Client Request Inter-arrival Time

Section 2.6 defines the web client request inter-arrival time as the time between requests for inline objects. The `Web Client Request Inter-arrival Time` parameter models this quantity.

The model parameter was measured in microseconds and took on values in the interval (79, 299900000) microseconds. As the interval shows, the parameter data were spread over an extremely large range spanning 8 orders of magnitude. The large range of values was attributed to the fact that some web-pages took a few minutes to download during times of network congestion. We observed that during times of severe network congestion a web-page could take up to five minutes to download completely. We used 5 minutes as the cutoff value for a request to be categorised as a web client request by the heuristic algorithm. Requests which followed 5 minutes or more after preceding requests were categorised as being requests generated by scripts implementing news-tickers or photo-galleries and were removed from the dataset for reasons explained in Section 4.7.

Table 32 shows summary statistics for the aggregated parameter dataset.

Sample Size	14563669
Five Number Summary	(79, 11 000, 65 300, 691 400, 299 900 000)
Sample Mean	1 546 451
Sample Variance	3.248178e+13
Standard Deviation	5 699 279
Coefficient of Variation	3.685
Skewness	10.347
Kurtosis	218.341

Table 32: Summary Statistics for `Web Client Request Inter-arrival Time` Parameter Dataset

The five number summary indicated a severe skew to the right in the distribution of the data. The range between the minimum value and the median was 65221 microseconds and the median and the maximum value was 299834700 microseconds. The very large values for skewness, kurtosis and coefficient of variation confirmed the large skew to the right. The distribution had a heavy tail.

More than 75% of inline object inter-arrival times were smaller than a second. We observed that web browsers usually placed requests for inline objects within a few hundred microseconds of one another. The mean value was severely affected by the skewness in the data. The average inter-arrival time value according to the median was 65300 microseconds or approximately 65 milliseconds. The data were however very variable with a standard deviation of 5.7 seconds and coefficient of variation of 3.7 seconds. The average value for inline object inter-arrival times was larger than we expected but a large number of inter-arrival time values had much smaller, or larger values due to the very large coefficient of variation. A possible reason for the larger inter-arrival time values was the large number of advertising content displayed on web pages. Advertising content usually resided on

different servers to the ones main web pages resided on. Requests for advertising content were often delayed until all requests for other inline objects had been made.

The tail of the distribution was very long with 25% of inline object inter-arrival time having a value between 700 milliseconds and 3 minutes.

Our findings were different to those of Choi et. al. [CL99] who found the average web client inter-arrival time to be approximately 900 milliseconds, with a standard deviation of 2.2 seconds. We observed the mean and standard deviation values to be much larger - 1500 milliseconds and 5.7 seconds. A possible reason for the difference was the simple heuristic they used to differentiate between web user and web client requests. The larger variability and upper tail of the distribution of our data suggested that the heuristic they used did not record larger inter-arrival time values.

78 microseconds were subtracted from every observation in the dataset in order to align the data with zero.

Table 33 shows the results of applying the λ^2 discrepancy measure test to the data.

Distribution	**Parameters**	λ^2	**Number of Bins**
Lognormal	$\zeta = 11.171529$ $\sigma = 2.928586$	0.032 $(0.032, 0.033)$	3682
Exponential	$\alpha = 1546375.237283$	396.152 $(393.44, 398.864)$	241
Weibull	$\gamma = 0.370912$ $\alpha = 315778.506166$	0.046 $(0.046, 0.046)$	1712
Gamma	$\gamma = 0.233279$ $\alpha = 6623382.113855$	0.97 $(0.949, 0.991)$	675
Pareto	$\alpha = 1$ $\beta = 0.089513$	0.879 $(0.878, 0.881)$	3682
Beta	$\alpha = 0.230429$ $\beta = 43.087639$	1.402 $(1.37, 1.434)$	657
Extreme Value	$\alpha = -1018535.977396$ $\beta = 4443709.770717$	45.172 $(45.08, 45.263)$	624
Normal	$\mu = 1546373.302262$ $\sigma = 5699279.211721$	226.203 $(224.62, 227.785)$	339

Table 33: Lambda Discrepancy Test Results for `Web Client Request Inter-arrival Time` Data over the Interval (1, 299899922)

The Weibull, lognormal, gamma and Pareto distributions provided the best fit to the data in that order. The tail of the distribution was modelled well by the Pareto and lognormal distributions according to the number of bins used to calculate λ^2.

Figure 32 is a plot of the lognormal, Weibull and gamma distributions fitted to a histogram of the data.

We shortened the range of the plot by plotting values in the range 1 to 1000000 microseconds instead of 1 to 300000000 microseconds. We shortened the range of the plot plot because the very

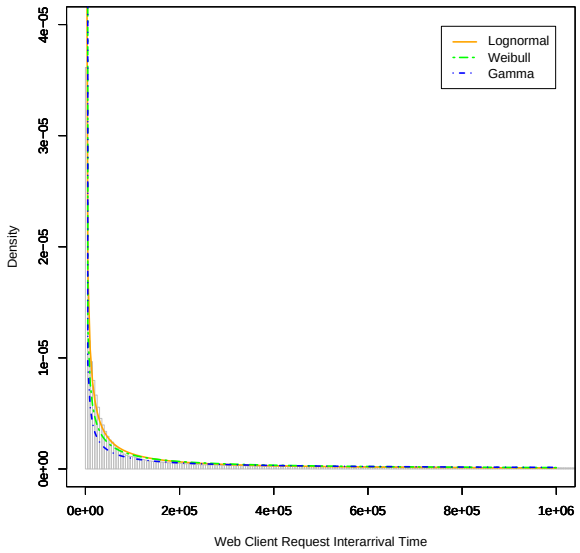


Figure 32: Best-Fit Distributions Plotted against a Histogram of `Web Client Request Inter-arrival Time` Data

long upper tail of the empirical distribution hid characteristics of smaller inter-arrival time values. The heavy tail was still obvious in the shortened plot. Another prominent feature of the empirical distribution was the very large number of small inter-arrival time values. The histogram showed that the three mathematical functions modelled the data well.

Figure 33 shows Q-Q plots for the lognormal and Weibull distributions.

Most of the data were concentrated around a very small part of the plot. 95% of the data were smaller than $10e^8$. The lines indicating the spread of 50%, 75% and 95% of the data did not feature on the Q-Q plot, because 95% of the data were concentrated around the lower tip of the Q-Q plot. The lognormal distribution severely overestimated the upper tail of the empirical distribution. The Weibull distribution underestimated the upper tail of the empirical distribution. The regression statistics in Table 34 confirmed that lognormal distribution modelled the tail of the distribution very poorly. The bulk of the data is however reasonably well modelled by the lognormal distribution.

Distribution	**Regression Statistics**
Lognormal	-0.869
Weibull	-0.061

Table 34: Regression Statistics for Lognormal and Weibull Q-Q Plots

The lognormal and Weibull distributions were good candidates for random number generation

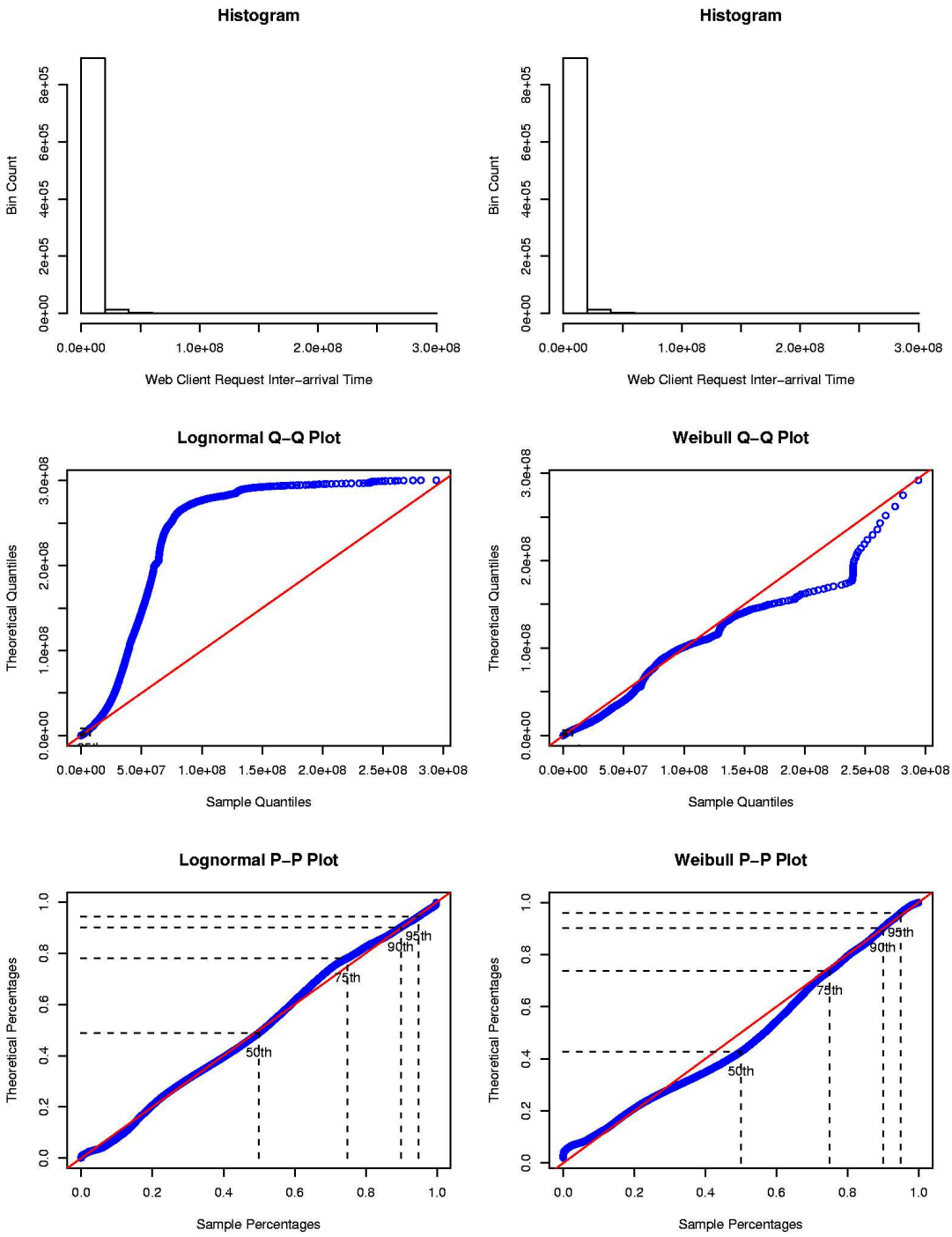


Figure 33: Weibull and lognormal Plots for `Web Client Request Inter-arrival Time` Parameter

for the `Web Client Request Inter-arrival Time` parameter.

6.9 Web User Request Size

Section 2.6 defined a web user request size as being the size of requests generated by web users. The `Web User Request Size` parameter modelled this quantity.

The parameter was measured in bytes and could take on values in the interval (37, 1446) bytes. The minimum value was the result of the smallest HTTP header we measured and the maximum value a result of the maximum number of HTTP data that could be transmitted in an Ethernet protocol data unit (PDU). The smallest HTTP header measured in the data had a size of 37 bytes. The maximum size of an Ethernet PDU was 15000 bytes, of which 54 bytes belonged to the TCP, IP and Ethernet headers (20 bytes each for TCP and IP and 14 bytes for Ethernet headers). The maximum size that HTTP data could occupy in a single TCP/IP packet was 1446 bytes. Web user requests could span more than one TCP/IP packet in which case the size of the request could be larger than 1446 bytes. We did not observe many requests spanning more than one TCP/IP packet. We found that only 0.6% of requests spanned more than one TCP/IP packet. We used 1446 bytes as the maximum possible size of an HTTP request.

Table 35 shows summary statistics for the aggregated parameter dataset.

Sample Size	715232
Five Number Summary	(37, 336, 419, 508, 1446)
Sample Mean	458.4
Sample Variance	39915.23
Standard Deviation	199.79
Coefficient of Variation	0.44
Skewness	2.166
Kurtosis	6.195

Table 35: Summary Statistics for `Web User Request Size` Parameter Dataset

The five number summary indicated a skew to the right. The range between the smallest value and the mean was 382 and the range between the mean and maximum value was 1027. The values of skewness and kurtosis confirmed the skew to the right. The coefficient of variation was much smaller than that of previous datasets, indicating that the data were distributed more closely around the mean.

Web user requests were small. The mean size of a request was 458 bytes, and 75% of requests were smaller than 508 bytes. Web user requests were small because they consisted of an HTTP header with no message body.

Choi et. al. [CL99] found the average web user request size to be approximately 360 bytes, with a standard deviation of 107 bytes. There was a 100 byte difference between their finding and ours. We observed that the requested URL field was usually the largest field in a web user request. Some

URLs were often very long, containing lengthy parameters to be passed to scripts running on web servers. The move to more dynamic web content and server side scripting and the resultant larger URL fields might have been the reason for the results of the study by Choi et. al. which showed a smaller mean web user request size. Their measurements were taken in 1998 and web content had become a great deal more dynamic since then.

36 bytes were subtracted from every observation in the dataset in order to align the data with zero.

Table 36 shows the results of applying the λ^2 discrepancy measure test to the data.

Distribution	**Parameters**	λ^2	**Number of Bins**
Lognormal	$\zeta = 5.958304$ $\sigma = 0.408075$	0.433 $(0.412, 0.453)$	227
Exponential	$\alpha = 422.43439$	1.33 $(1.322, 1.339)$	237
Weibull	$\gamma = 2.201577$ $\alpha = 477.762279$	1.251 $(1.215, 1.288)$	227
Gamma	$\gamma = 5.86105$ $\alpha = 72.074916$	0.643 $(0.626, 0.66)$	229
Pareto	$\alpha = 1$ $\beta = 0.167833$	12.259 $(12.185, 12.332)$	237
Beta	$\alpha = 2.893784$ $\beta = 6.460851$	8.832 $(8.517, 9.147)$	210
Extreme Value	$\alpha = 342.757047$ $\beta = 129.85828$	0.393 $(0.381, 0.404)$	233
Normal	$\mu = 422.434385$ $\sigma = 199.787971$	10.761 $(10.393, 11.129)$	205

Table 36: Lambda Discrepancy Test Results for `Web User Request Size` Data over the Interval $(1,\ 1410)$

The extreme value, lognormal and gamma distributions provided the best fit in that order. All the distributions modelled the tail of the distribution well according to the number of bins used to calculate λ^2.

Figure 34 is a plot of the extreme value, lognormal and gamma distributions fitted to a histogram of the data.

The histogram shows a positive skew in the distribution of the data. The bulk of the data fell in the $(350, 500)$ byte interval. The distribution of the data was very symmetric. The coefficient of variation was much lower than that of any of the previously analysed parameter datasets. The three mathematical functions plotted in Figure 34 seemed to fit the data well according to the histogram.

Figure 35 shows Q-Q plots for the extreme and lognormal distributions.

The Q-Q plots fitted the straight lines well for values up to approximately 800 bytes. The upper tail of the distribution was underestimated by both the mathematical functions. As before, the bulk

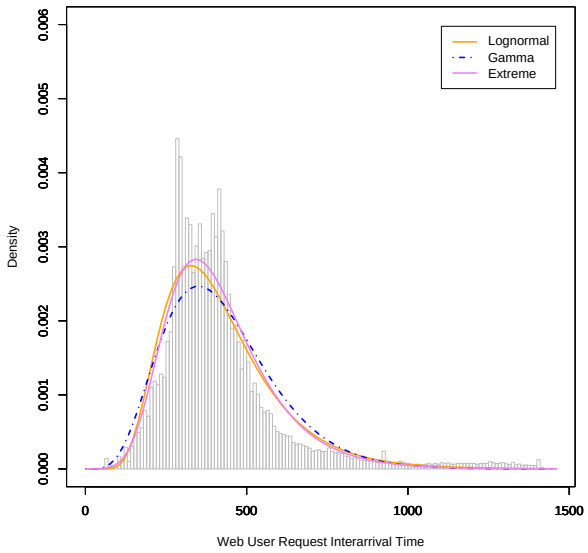


Figure 34: Best-Fit Distributions Plotted against a Histogram of `Web User Request Size` Data

of the data were reasonably well modelled by the distributions. 75% Of the data fell in the interval (0, 510) bytes. Both mathematical functions modelled the (0, 510) byte interval well. Judging by the Q-Q plot, the lognormal distribution modelled the empirical distribution slightly better than the extreme value distribution. The regression results reported in Table 37 indicated that both mathematical functions fitted the data reasonably well.

Distribution	Regression Statistics
Extreme	-0.015
Lognormal	-0.011

Table 37: Regression Statistics for Lognormal and Extreme Q-Q Plots

The extreme and lognormal distributions were good candidates for random number generation for the `Web User Request Size` parameter.

6.10 Web Client Request Size

Section 2.6 defined a web client request size as being the size of requests generated by web clients. The `Web Client Request Size` parameter modelled this quantity.

The parameter was measured in bytes and took on values in the interval (37, 1446) bytes. The interval was the same as for the `Web User Request Size` parameter for the same reasons.

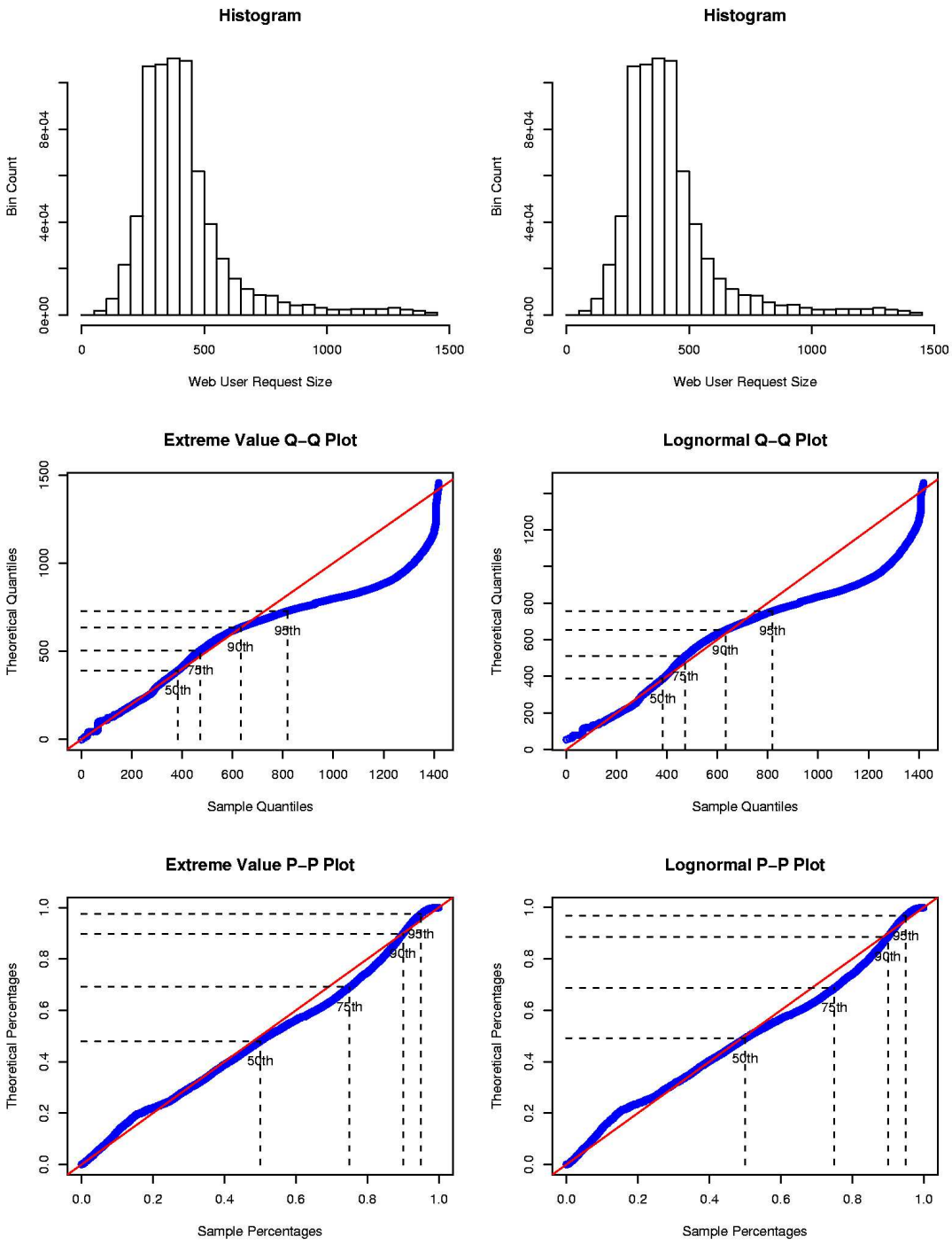


Figure 35: Extreme and Lognormal Plots for `Web User Request Size` Parameter

Table 38 shows summary statistics for the aggregated parameter dataset.

Sample Size	15 284 603
Five Number Summary	(37, 325, 384, 455, 1446)
Sample Mean	417.6
Sample Variance	24132.15
Standard Deviation	155.345
Coefficient of Variation	0.372
Skewness	2.628
Kurtosis	10.003

Table 38: Summary Statistics for `Web Client Request Size` Parameter Dataset

The parameter had similar statistics to the `Web User Request Size` parameter. The values for the sample mean, standard deviation and coefficient of variation were slightly smaller, the skewness had a very similar value. The kurtosis value was larger than that of the `Web User Request Size` parameter indicating heavier tails and the data being more centred around the mean. The five number summary indicated a skew to the right. The dataset had more than 15 million entries.

36 bytes were subtracted from every observation in the dataset in order to align the data with zero.

Table 39 shows the results of applying the λ^2 discrepancy measure test to the data.

Distribution	**Parameters**	λ^2	**Number of Bins**
Lognormal	$\zeta = 5.883715$ $\sigma = 0.330966$	0.766 $(0.753, 0.778)$	672
Exponential	$\alpha = 381.648588$	1.722 $(1.72, 1.725)$	709
Weibull	$\gamma = 2.446442$ $\alpha = 428.976329$	18.125 $(17.894, 18.356)$	605
Gamma	$\gamma = 8.386872$ $\alpha = 45.506516$	4.304 $(4.251, 4.357)$	616
Pareto	$\alpha = 1$ $\beta = 0.169961$	14.357 $(14.338, 14.375)$	709
Beta	$\alpha = 4.125228$ $\beta = 11.242457$	67.26 $(66.57, 67.949)$	570
Extreme Value	$\alpha = 311.736848$ $\beta = 121.12221$	0.257 $(0.255, 0.258)$	709
Normal	$\mu = 381.648588$ $\sigma = 155.34527$	151.612 $(150.161, 153.062)$	545

Table 39: Lambda Discrepancy Test Results for `Web Client Request Size` Data over the Interval (1, 1410)

The extreme value and lognormal distributions provided the best fit in that order. Both distributions modelled the tail of the distribution well according to the number of bins used to calculate λ^2.

Figure 36 is a plot of the extreme value and lognormal distributions fitted to a histogram of the data.

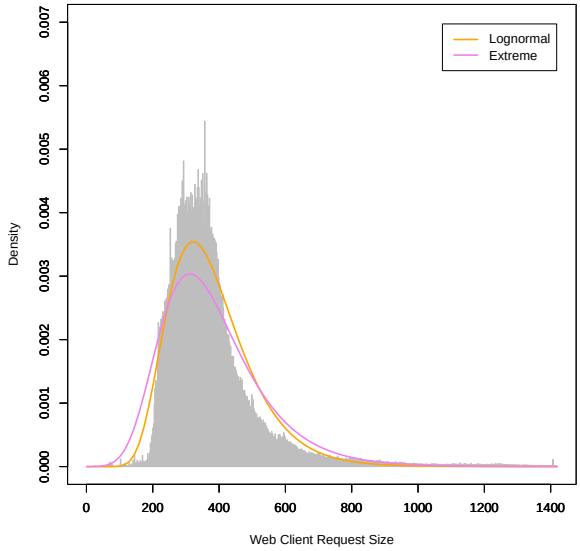


Figure 36: Best-Fit Distributions Plotted against a Histogram of `Web Client Request Size` Data

The two mathematical functions fitted the data reasonably well according to the histogram. The histogram of the `Web Client Request Size` had a similar shape to that of the `Web User Request Size` parameter's histogram. It was not surprising to find that the two parameters were well modelled by the same mathematical functions, with very similar model constants.

Figure 37 shows Q-Q plots for the extreme value and lognormal distributions.

The plots looked very similar to the Q-Q plots of the `Web User Request Size` parameter. The mathematical functions provided a good fit to the data for requests up to a size of approximately 600 bytes. 75% Of the data fall in the interval (0, 455) bytes. The mathematical functions modelled the bulk of the data very well. The mathematical functions however underestimate the upper tail of the distribution.

The regression statistics shown in Table 40 indicate that both the distributions fit the data reasonably well.

The extreme value and lognormal distributions were good candidates for random number generation for the Web Client Request Size parameter.

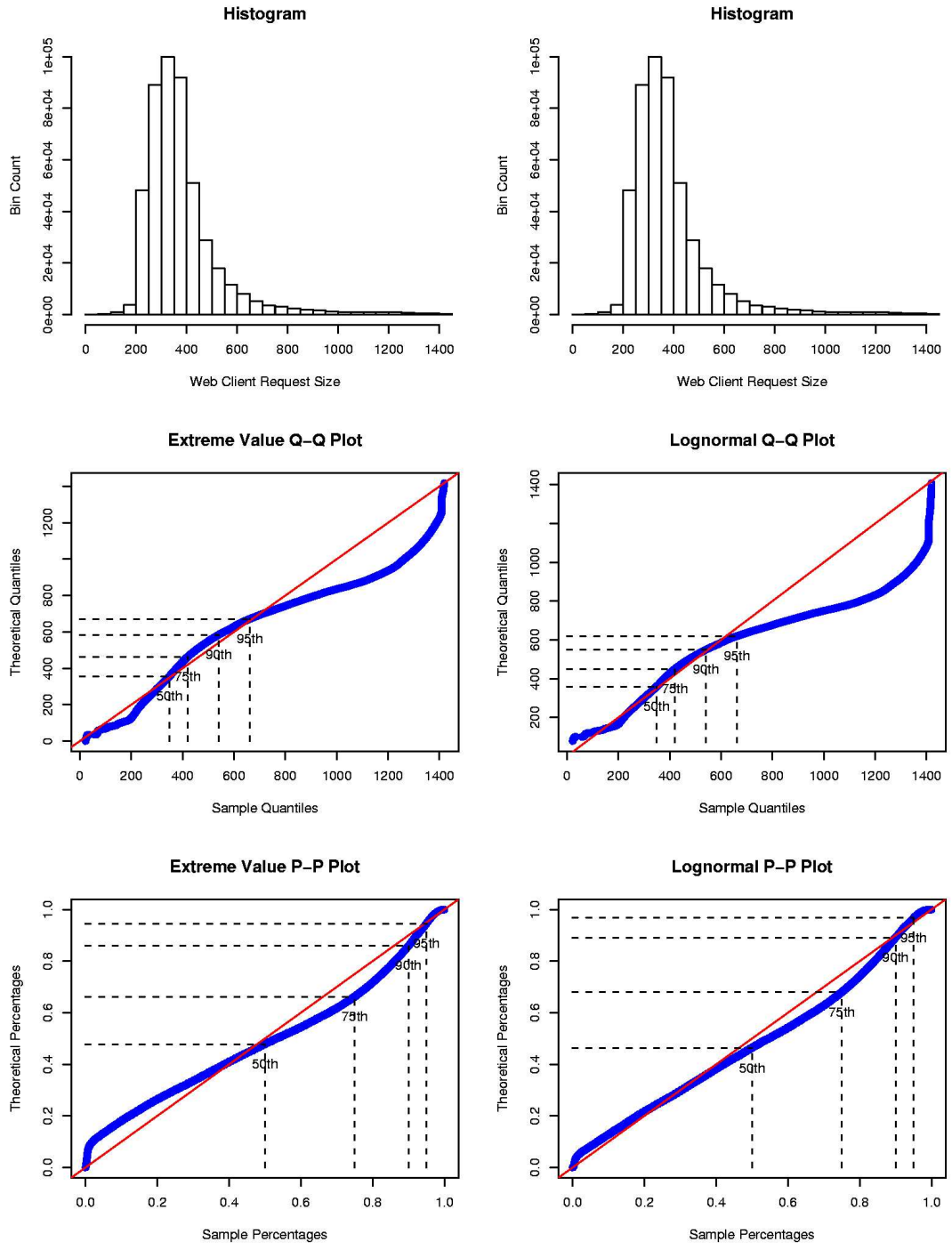


Figure 37: Extreme Value and Lognormal Plots for `Web Client Request Size` Parameter

Distribution	Regression Statistics
Extreme	-0.011
Lognormal	-0.015

Table 40: Regression Statistics for Extreme and Lognormal Q-Q Plots

6.11 Web User Response Size

Section 2.6 defined web user response size as the size of responses to requests by web users. The `Web User Response Size` parameter modelled this quantity.

As mentioned in Section 6.3 Choi et. al. [CL99] made the important distinction between cached and non-cached `Web User Requests`. If a requested object in a host machine's local cache had not expired yet i.e. if the object had not been modified at the server, the web server replied to a web user request with a `304 Not Modified` response code indicating that the object in the cache should be used. Replies of this type consisted of an HTTP header with no message body and therefore had a small size.

A cached web-page was loaded from local cache instead of being downloaded from a web-server. These requests therefore resulted in less data being transmitted across a network. The `Expires` entity-header in an HTTP message indicated the time of expiry of a page. The page could be safely used from the cache until the time indicated in the `Expires` entity-header. The `If-Modified-Since` request modifier allowed for conditional `GET` requests. These requests resulted in `304 Not Modified` responses from a server if the cached object was still fresh. As mentioned before these responses had no message body.

It was important for us to treat `304 Not Modified` responses separately from others as they were smaller than regular responses and occurred very regularly. They contaminated `Web Client` and `Web User Response Size` distributions if not analysed separately.

We modelled cached web client and user responses separately from non-cached web user and client responses as they could be seen to constitute a subclass of responses with different characteristics to non-cached responses.

We next discuss cached web client responses, and then analyse non-cached web client responses.

16% of web user responses were cached. Cached responses had sizes in the interval of $(250, 300)$ bytes. Figure 38 shows a histogram of the sizes of cached responses.

There were two spikes in the histogram for values 262-266 and 276-280. There was not much variation in the data as web server replies with response code `304 Not Modified` were fairly standard, they consisted of an HTTP header with the `304 Not Modified` code as one of its fields. 65% of values had a value smaller than 70 and 35% a value larger than 70. As most of the values in these two groups fell either in the range 262-266 or 276-280 we modelled the parameter as a `Bernoulli Variable` as follows:

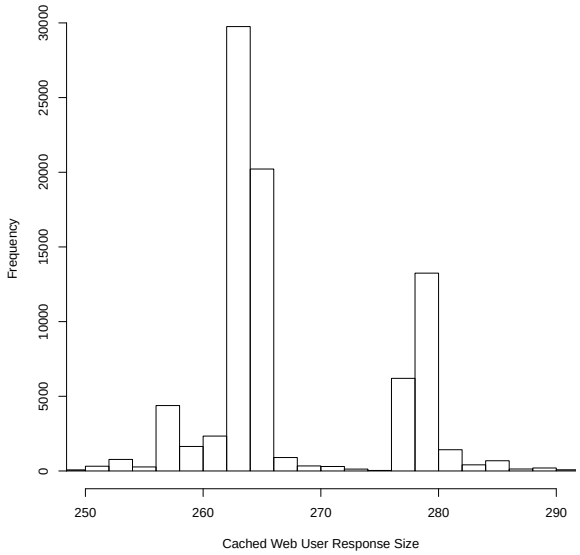


Figure 38: Histogram of `Cached Web User Response Size` Data

$$\begin{aligned} p(265) &= p \\ p(280) &= 1-p \\ p(x) &= 0, \quad if\ x \neq 265\ \ and\ \ x \neq 280 \end{aligned}$$

for $p = 0.65$.

Table 41 shows summary statistics for the aggregated parameter dataset of non-cached web user responses.

Sample Size	328684
Five Number Summary	(182, 817, 4671, 16140, 60000)
Sample Mean	10652.96
Sample Variance	169492 452
Standard Deviation	13018.93
Coefficient of Variation	1.222
Skewness	1.498
Kurtosis	1.552

Table 41: Summary Statistics for `Non-cached Web User Response Size` Parameter Dataset

The parameter was measured in bytes and took on values in the interval (182, 60 000) bytes. The interval was not influenced by the heuristic algorithm. The largest value in the dataset was 60 KB.

The limit imposed by the heuristic algorithm was 1 MB, but no values of this size were recorded for the `Non-cached Web User Response Size` parameter. Web user responses were usually HTML files, which were relatively small. The minimum value recorded for the parameter was 182 bytes which was the size of a small HTTP response header without a message body. HTTP responses with response codes `1xx Informational` or `204 No Content` did not have message bodies.

The average web user response size according to the mean value was approximately 10.5 KB, the standard deviation was approximately 13 KB. The values for skewness and kurtosis indicated that the data were skewed to the right. The large range of values between the median and maximum value confirmed the skew in the data.

Choi et. al. [CL99] found the average web user response size to be approximately 10.7 KB, with a standard deviation of 25 KB. The mean value was very similar to which we had observed, but the standard deviation was larger.

We aligned the data to zero by subtracting 181 bytes from every observation. Table 42 shows the results of applying the λ^2 discrepancy measure test to the data.

Distribution	**Parameters**	λ^2	**Number of Bins**
Lognormal	$\zeta = 8.16926$ $\sigma = 1.714619$	0.338 $(0.332, 0.344)$	91
Exponential	$\alpha = 10471.964185$	0.897 $(0.885, 0.908)$	91
Weibull	$\gamma = 0.684073$ $\alpha = 8204.07327$	0.218 $(0.213, 0.222)$	91
Gamma	$\gamma = 0.572347$ $\alpha = 18312.318592$	0.193 $(0.189, 0.197)$	91
Pareto	$\alpha = 1$ $\beta = 0.12241$	2.614 $(2.584, 2.644)$	91
Beta	$\alpha = 0.473018$ $\beta = 2.076103$	0.147 $(0.143, 0.152)$	91
Extreme Value	$\alpha = 5089.125834$ $\beta = 7875.764979$	2.71 $(2.677, 2.742)$	91
Normal	$\mu = 10471.956831$ $\sigma = 13018.926685$	4.81 $(4.753, 4.867)$	91

Table 42: Lambda Discrepancy Test Results for `Non-cached Web User Response Size` Data over the Interval (1, 59818)

The beta, gamma and Weibull distributions provided the best fit in that order. All the distributions modelled the tail of the distribution well according to the number of bins used to calculate λ^2.

Figure 39 is a plot of the beta, gamma and Weibull distributions fitted to a histogram of the data.

There was a large number of web user responses with small sizes in the range 180 to 200 bytes.

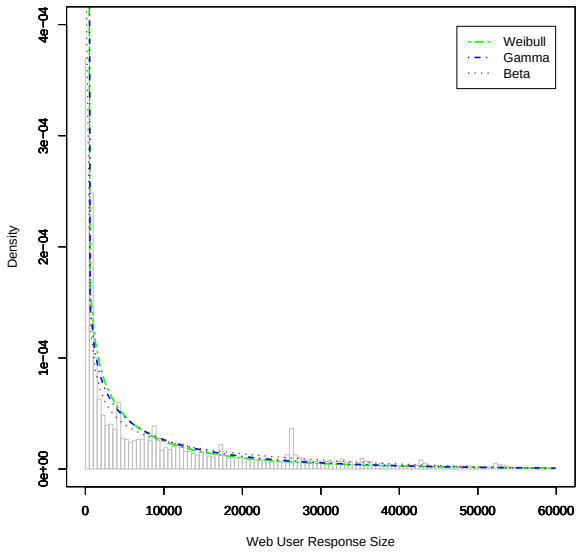


Figure 39: Best-Fit Distributions Plotted against a Histogram of `Non-cached Web User Response Size` Data

The distribution had a long upper tail which contained a significant number of requests. The three distributions fitted the data well over the whole range of values.

Figure 40 shows Q-Q plots for the Weibull and gamma distributions.

The Q-Q plots for both distribution functions fit the straight lines reasonably well. 75% Of the data fell in the interval (0, 16140). The straight lines fit the Q-Q plots reasonably well over the interval (0, 16140). For values larger than 16410 the empirical data had larger values than the mathematical functions. The regression statistics shown in Table 43 showed that both the mathematical functions modelled the data reasonably well.

Distribution	Regression Statistics
Weibull	-0.029
Gamma	-0.019

Table 43: Regression Statistics for Weibull and Gamma Q-Q Plots

The gamma and Weibull distributions were good candidates for random number generation for the `Non-cached Web User Response Size` parameter.

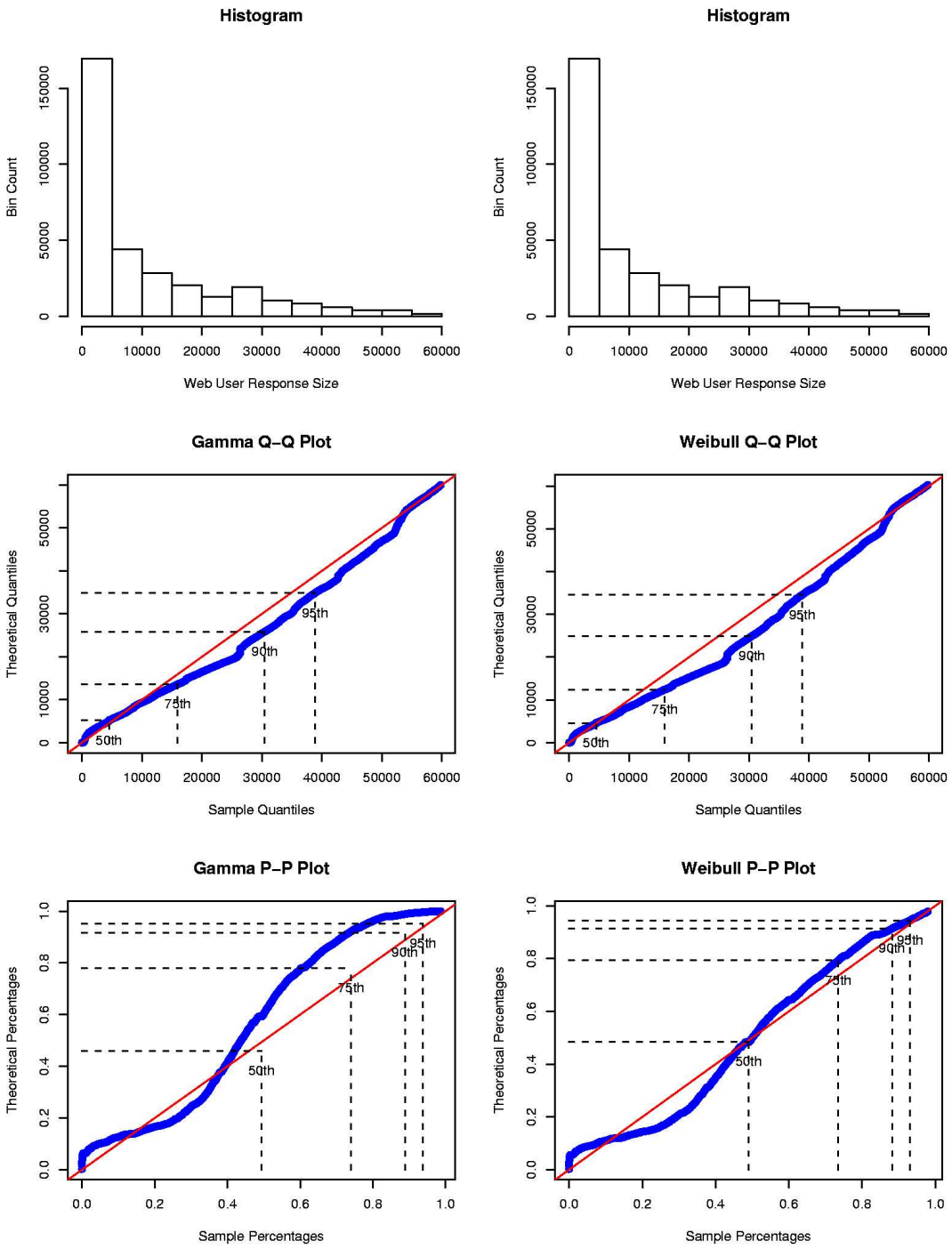


Figure 40: Weibull and Gamma Plots for `Non-cached Web User Response Size` Parameter

6.12 Web Client Response Size

Section 2.6 defined a web client response size as the size of responses to requests by web clients. The `Web Client Response Size` parameter modelled this quantity.

As for the `Web User Response Size`, the `Web Client Response Size` parameter was influenced by `304 Not Modified` responses. 55% of web client responses were cached. This value was much larger than we expected compared to the 16% of cached web user response objects. The result indicated that HTML main pages were updated more regularly than graphical inline images, causing main pages to be invalidated by the server more regularly than inline images. Inline images were downloaded once during a browsing session and were subsequently loaded from local cache when other web pages on the same site were visited.

Cached responses had sizes in the interval of $(250, 300)$ bytes. Figure 41 shows a histogram of the sizes of cached responses.

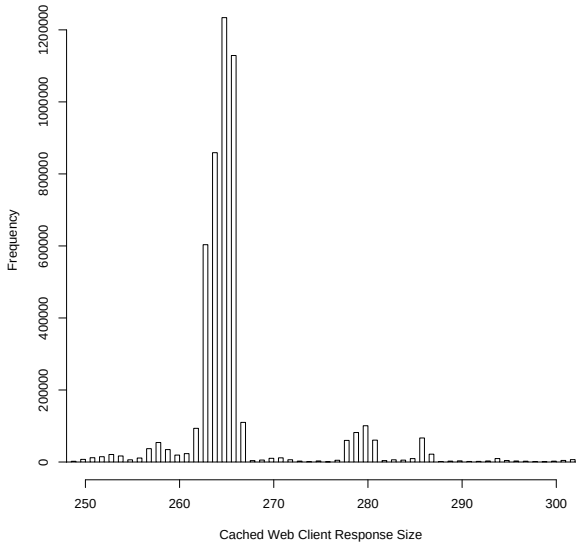


Figure 41: Histogram of `Cached Web Client Response Size` Data

Most of the data fell in the interval 260-270. We modelled the parameter as having a fixed value of 265.

Table 44 shows summary statistics for the aggregated parameter dataset of non-cached web client responses.

The parameter was measured in bytes and took on values in the interval (163, 999600) bytes. The heuristic algorithm removed values greater than 1MB as we observed that inline objects very seldomly had values greater than 1MB. Inline objects were generally small graphical images, reduced

Sample Size	8681843
Five Number Summary	(163, 681, 1511, 4546, 999600)
Sample Mean	5222.176
Sample Variance	255822373
Standard Deviation	15994.45
Coefficient of Variation	3.063
Skewness	21.826
Kurtosis	836.066

Table 44: Summary Statistics for `Non-cached Web Client Response Size` Parameter Dataset

in size as much as possible to reduce download and display time of web pages.

The average size of an inline object was approximately 5 KB with a standard deviation of approximately 16 KB. It was interesting to note that the average size of an inline object was smaller than that of a main object. A main object was had an average size of 11 KB. The fact that the average inline object was smaller than the main object meant that graphical images displayed on web pages were smaller in size than files containing the text displayed on web pages. The standard deviation was however very large as indicated by the coefficient of variation. The probability of encountering a very large inline object, relative to an average sized one, was very large.

The data were severely skewed to the right. The large range of values between the median and maximum value, the skewness and kurtosis confirm this. There was strong evidence that the distribution had a heavy tail.

We aligned the data to zero by subtracting 162 from every observation. Table 45 shows the results of applying the λ^2 discrepancy measure test to the normalised data.

The lognormal distribution was the only distribution that provided a reasonable fit to the data. The Pareto distribution provided a good fit to the tail of the distribution.

Figure 42 is a plot of the lognormal distribution fitted to a histogram of the data.

The histogram showed that a large number of client responses had a size in the range of 160 to 200 bytes. The very long upper tail of the distribution contained a significant number of client responses. The lognormal distribution appeared to provide a good fit to the data.

Figure 43 shows a Q-Q plot for the lognormal distribution. The regression statistic for the fit of the data to the straight line in Figure 43 is shown in Table 46.

The lines indicating values of the quantiles which bound 50%, 75%, and 95% of the data did not show on the Q-Q plot in Figure 43. The reason for this was that most of the data had very small values. 75% Of the data fell in the interval (0, 4546) bytes, which is represented by the lower tip of the Q-Q plot. The straight line fit the Q-Q plot well over the (0, 4546) interval. The very long upper tail of the empirical distribution was not well modelled by the lognormal distribution. The regression statistic shown in Table 46 indicated that the lognormal distribution provided a reasonably good fit to the the data.

The lognormal distribution was a good candidate for random number generation for the `Non-cached`

Distribution	Parameters	λ^2	Number of Bins
Lognormal	$\zeta = 7.400775$ $\sigma = 1.405093$	0.104 $(0.104, 0.105)$	1152
Exponential	$\alpha = 5060.176166$	38.723 $(38.185, 39.26)$	227
Weibull	$\gamma = 0.663302$ $\alpha = 3398.539724$	1.383 $(1.348, 1.417)$	465
Gamma	$\gamma = 0.554266$ $\alpha = 9126.253815$	7.909 $(7.744, 8.075)$	340
Pareto	$\alpha = 1$ $\beta = 0.135121$	2.759 $(2.754, 2.764)$	3675
Beta	$\alpha = 0.094602$ $\beta = 18.595562$	2.057 $(2.051, 2.063)$	1063
Extreme Value	$\alpha = -2137.981187$ $\beta = 12470.820084$	9.066 $(9.014, 9.118)$	511
Normal	$\mu = 5060.176166$ $\sigma = 15994.448181$	31.23 $(30.867, 31.593)$	283

Table 45: Lambda Discrepancy Test Results for `Non-cached Web Client Response Size` Data over the Interval (1, 999438)

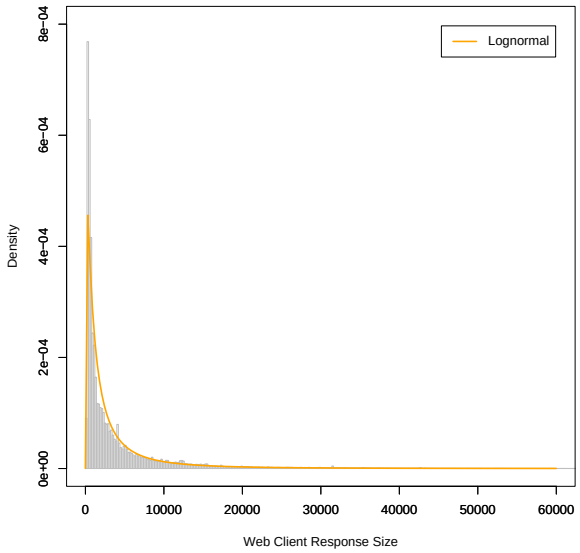


Figure 42: Best-Fit Distributions Plotted against a Histogram of `Non-cached Web Client Response Size` Data

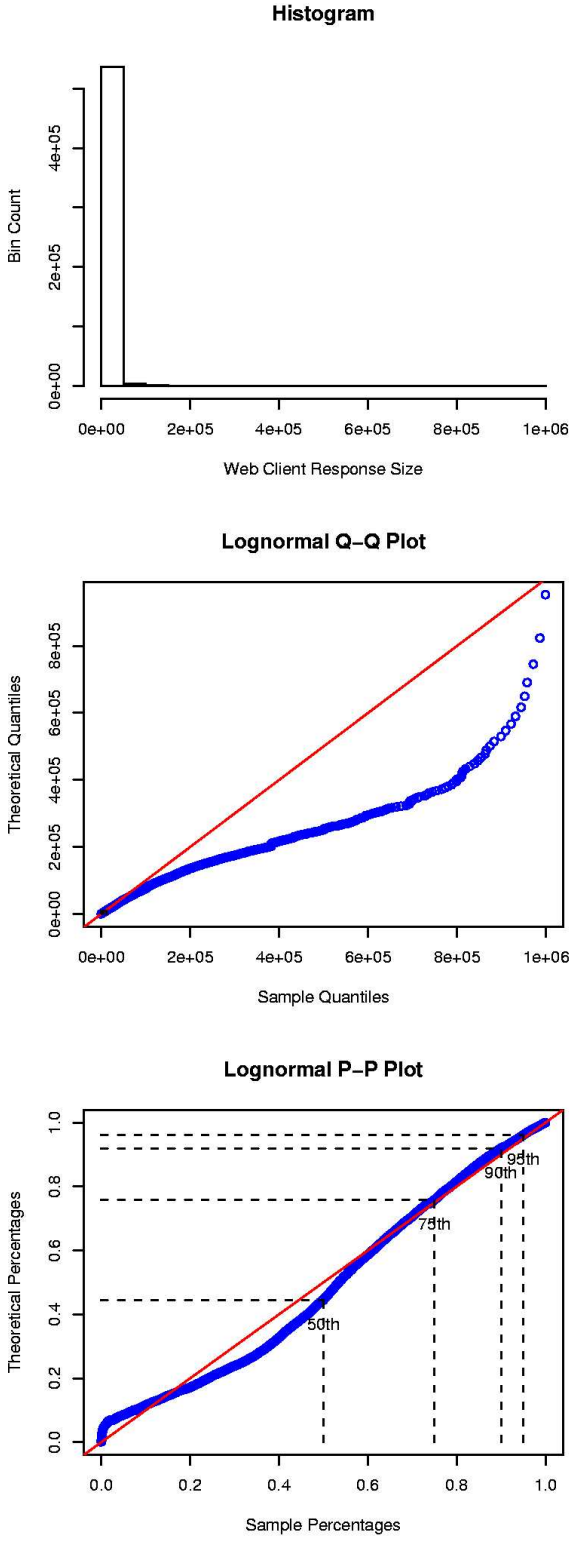


Figure 43: Lognormal Plots for `Non-cached Web Client Response Size` Parameter

Distribution	Regression Statistic
Lognormal	-0.18

Table 46: Regression Statistic for `Lognormal Q-Q Plot`

`Web Client Response Size` parameter.

6.13 Data Variability Analysis

Highly variable inter-arrival time and size distributions have a negative impact on network performance. Computer servers and network components react poorly to highly variable traffic. Computer and network equipment have to be equipped with extra storage capacity and processing power to handle highly variable traffic loads.

The phenomenon of self-similarity in network traffic has a severely negative impact on network performance. Self-similarity occurs when network traffic inter-arrival times are highly variable on a variety of different time scales from microseconds to several minutes. Appendix D discusses concepts related to self-similarity.

It is important to identify components of traffic which are highly variable. By understanding the cause of high variability in traffic it is possible to take preventative steps or to prepare strategies to deal with its results. We quantified variability in the nine model parameters and identified parameters which had the most variability.

The lognormal, Weibull and Pareto distributions model high variability in data well. The Pareto distribution is known to model very high variability in data well. Previous studies had shown that the lognormal, Weibull and Pareto distributions modelled datasets resulting from data network traffic measurements well. Data measured from computer networks were typically highly variable [Fel98, BC98b, CL99]. Six of the nine model parameters we analysed had a lognormal or Weibull distribution and three of the parameters had a Pareto distribution. The parameter datasets we analysed were typically highly variable.

Table 47 shows the result of applying several metrics for variability to the nine model parameter datasets we analysed.

The coefficient of variation (CoV) was previously discussed for each model parameter. A coefficient of variation greater than one indicated high variability. Table 47 shows that all the parameters were highly variable except for the `Web User Request Size` and `Web Client Request Size` parameters. The `Web Client Request Inter-arrival Time` and `Non-cached Web Client Response Size` parameters were very highly variable as indicated by their CoV values which were greater than 3.

The second indicator of variability was the γ parameter of the Weibull distribution. A value smaller than 1 indicated high variability and a value smaller than 0.5 indicated very high variability. Model parameters which did not fit the Weibull distribution did not have this metric. Results based

Name	CoV	γ	β
Browsing Inter-Session Time	1.075	0.65	—
Number of Web User Requests per Browsing Session	1.132	0.93	0.78
Number of Web Client Requests per Web User Request	1.349	0.79	0.46
Web User Request Inter-arrival Time	1.218	0.86	—
Web Client Request Inter-arrival Time	3.685	0.37	0.09
Web User Request Size	0.44	—	—
Web Client Request Size	0.372	—	—
Non-cached Web User Response Size	1.222	0.68	—
Non-cached Web Client Response Size	3.063	—	—

Table 47: Metrics of Variability Applied to the Nine Model Parameter Datasets

on the γ parameter have the same interpretation as those for the CoV.

The Pareto distribution modelled very highly variable data. A Pareto distribution with β parameter between 0 and 2 was a heavy tailed distribution. Heavy tailed distributions were discussed in Section 5.6 and model very highly variable data. The β parameter of the Pareto distribution was the third metric of variability. The lower the value β the higher the variability in the data. (Results based on the β parameter corresponded to results obtained from the preceding two metrics but only applied to three parameters as the Pareto distribution provided a good fit to three parameters only). The `Web Client Request Inter-arrival Time` parameter had β value of 0.09 indicating very high variability.

The `Non-cached Web Client Response Size` parameter had a very high coefficient of variation but surprisingly neither of the two distributions which modelled very high variability i.e. the Pareto and the Weibull distributions, fitted the parameter dataset. We suspected that the underlying distribution was contaminated by the long upper tail, and that the tail was well modelled by the Pareto distribution. Barford and Crovella found this to be the case for web resource sizes where the body of the distribution was well modelled by the lognormal distribution and the tail of the distribution was modelled by the Pareto distribution [BC98b]. It was possible to construct such a hybrid model composed of both the lognormal and Pareto distributions. We did not pursue this avenue as the lognormal distribution provided a good fit to the whole parameter dataset.

We concluded that all the parameters except for the `Web User Request Size` and `Web Client Request Size` parameters had highly variable data and that the `Number of Web User Requests per Browsing Session`, `Number of Web Client Requests per Web User Request`, `Web Client Request Inter-arrival Time` and the `Non-cached Web Client Response Size` parameters had very highly variable distributions with heavy tails.

6.14 Results Summary

Tables 48 and 49 list the distributions that were found to fit each of the model parameters.

Model Parameter	Distributions	Distribution Parameters	Offset	Range
Browsing Inter-Session Time	Gamma	$\gamma = 0.645296$ $\alpha = 102.342465$	14	$(1, 465)$
	Weibull	$\gamma = 0.7437$ $\alpha = 54.902209$		
	Lognormal	$\zeta = 3.242876$ $\sigma = 1.637514$		
Number of Web User Requests per Browsing Session	Pareto	$\alpha = 1$ $\beta = 0.557807$	1	$(1, 99)$
	Beta	$\alpha = 0.777341$ $\beta = 5.164118$		
	Weibull	$\gamma = 0.912485$ $\alpha = 10.681936$		
Number of Web Client Requests per Web User Request	Lognormal	$\zeta = 2.154982$ $\sigma = 1.377062$	0	$(1, 200)$
	Pareto	$\alpha = 1$ $\beta = 0.464041$		
	Weibull	$\gamma = 0.793412$ $\alpha = 17.067434$		
Web User Request Inter-arrival Time	Weibull	$\gamma = 0.85689$ $\alpha = 106.420313$	0	$(1, 900)$
	Gamma	$\gamma = 0.795353$ $\alpha = 145.127336$		
	Exponential	$\alpha = 115.426513$		
Web Client Request Inter-arrival Time	Weibull	$\gamma = 0.370912$ $\alpha = 315778.506$	78	$(1,\ 299899922)$
	Lognormal	$\zeta = 11.171529$ $\sigma = 2.928586$		
	Pareto	$\alpha = 1$ $\beta = 0.089513$		

Table 48: Best-fit Distributions and their Parameters for the Eleven Model Parameters - 1

Model Parameter	Distributions	Distribution Parameters	Offset	Range
Web User Request Size	Extreme Value Lognormal Gamma	$\alpha = 342.757047$ $\beta = 129.85828$ $\zeta = 5.958304$ $\sigma = 0.408075$ $\gamma = 5.86105$ $\alpha = 72.074916$	36	(1, 1410)
Web Client Request Size	Extreme Value Lognormal	$\alpha = 311.736848$ $\beta = 121.12221$ $\zeta = 5.883715$ $\sigma = 0.330966$	36	(1, 1410)
Cached Web User Response Size	Bernoulli Variable	$p = 0.65$	0	fixed value: 265 or 280
Non-cached Web User Response Size	beta Gamma Weibull	$\alpha = 0.473018$ $\beta = 2.076103$ $\gamma = 0.572347$ $\alpha = 18312.318592$ $\gamma = 0.684073$ $\alpha = 8204.07327$	181	(1, 59818)
Cached Web Client Response Size	Fixed value	—	0	fixed value: 265
Non-cached Web Client Response Size	Lognormal	$\zeta = 7.400775$ $\sigma = 1.405093$	162	(1, 999438)

Table 49: Best-fit Distributions and their Parameters for the Eleven Model Parameters - 2

In most cases more than one distribution fitted the data well. The distributions listed for each parameter in Tables 48 and 49 were listed in order of how well they fitted the data i.e. best fitting distributions first. The distributions were fitted to the data over the range of values in the data. The distribution parameters and the range of values over which the distributions were fitted are shown. The offset value in the table is the value which was subtracted from each observation in the dataset in order to align the data with zero.

We failed to find analytic distributions for the `Web User Response Size` and `Web Client Response Size` parameters. We found that by splitting each of these parameters in two and modelling the resultant four distributions as cached and non-cached versions of the original parameters we were able to fit analytic distributions to all four parameters. The extra two parameters and their distributions are shown in Table 49.

Table 50 shows the mean and standard deviation for parameter datasets. The average time between two browsing sessions was approximately one and a half hours and there was a significant chance that a break between browsing sessions might be as long as 8 hours. The average number of web pages visited during a browsing session was approximately 12 which is smaller than we expected. The parameter had a heavy tailed distribution and therefore there was a large likelihood that the number of pages visited was much larger, up to a maximum of 100 pages per session. On average a web page contained 20 inline objects. The variability in the data were very high as this parameter also had a heavy tailed distribution. The likelihood that up to 200 inline objects were contained in a web page was very large.

The average time between two requests for a web page by a user was a little less than 2 minutes. The `Web User Request Inter-arrival Time` parameter was the only parameter which could be modelled by an exponential distribution. The time between requests placed by a web client was measured in microseconds. The average time between two such requests was one and a half seconds according to the mean. The data were very skewed with a heavy tailed distribution and therefore the median of approximately 65 milliseconds was a better indicator of the average time between two web client requests. The likelihood of a value having a value larger than 65 milliseconds was very large. The parameter had a maximum value of 3 minutes.

The average sizes of web user and client requests were very similar. They were also well modelled by the same distributions with very similar distribution parameters. We concluded that both parameters could be accurately modelled by a single distribution. The average size of these requests was approximately 450 bytes. There was not much variability in the data for these two parameters.

The average size of the main object of a web page was approximately 10.5KB. The largest recorded size of a main object was 60KB. The data were not highly variable. The average size of an inline object was approximately 5KB but the data were highly variable with a heavy tail. The average size of an inline image was smaller than that of a main object. There was a large likelihood that images as large as 1MB were requested.

Model Parameter	Mean	Standard Deviation	Unit of Measurement
Browsing Inter-Session Time	81.04	87.102	minute
Number of Web User Requests per Browsing Session	12.22	13.872	count
Number of Web Client Requests per Web User Request	19.687	26.551	count
Web User Request Inter-arrival Time	115.427	140.642	second
Web Client Request Inter-arrival Time	1 546 451	5 699 279	microsecond
Web User Request Size	464.495	213.731	byte
Web Client Request Size	420.2	163.398	byte
Non-cached Web User Response Size	10 652.96	13 018.93	byte
Non-cached Web Client Response Size	5222.176	15994.45	byte

Table 50: Mean and Standard Deviation of Nine Model Parameter Datasets

6.15 Concluding Remarks

It was a very difficult and time consuming task to find analytic distributions for each of the nine workload model parameters. The first challenge which we overcame was to find ways in which to analyse the very large parameter datasets. We have already discussed how we solved this problem in the previous chapter. The second challenge was the implementation of the Anderson Darling test and λ^2 discrepancy measure in the R statistical analysis environment. The Anderson Darling test proved to be inadequate for our purposes, as it could not be used to analyse datasets with more than 200 observations. We therefore used the λ^2 measure in combination with Q-Q Plots to analyse the data. We discussed the implementation of the λ^2 statistic in the previous chapter. The third challenge was to remove all data which were not generated by users browsing the web from parameter datasets. We have discussed this problem in Section 4.7 (page 63), but will elaborate on it here.

Due to the nature of the measurements we were not able to isolate web browsing traffic during the initial measurements. Measured traffic included traffic not generated by web clients but rather by applications which also used the HTTP for purposes other than browsing the web. The traffic also included traffic which was generated by web browsers, but did not constitute "traditional" web browsing i.e. the browsing of web sites which displayed information using HTML or some other web technology such as Flash. This second "untraditional" type of web traffic was traffic such as web-downloads, web-irc, streaming-audio etc. Both of the above-mentioned types of traffic were not part of the workload model and were very difficult to remove from the measurement data. Some of this "unwanted data" could be removed during the initial processing of the measurement data but a large

part of it was removed during the analysis of parameter datasets. By analysing parameter datasets we were able to identify host datasets which contained unwanted data. Parameter datasets which included values which were unlikely to have been caused by users browsing the web were identified, and the relevant host datafiles were removed from the study. The removal of unrepresentative data was a very time consuming process. We removed 4865 datasets from the study. We were left with 1827 datasets for analysis. We discussed how we removed host datasets from the data in Section 4.7.2 (page 64).

Chapter 7

Findings and Future Work

7.1 Findings

We employed the structural approach to network modelling to derive a model for web workload. We researched models used in simulation studies as well as models implemented in simulation packages and found that important web workload characteristics are not included in these models. In particular bidirectional traffic and heavy tailed distributions for inter-arrival time and size workload parameters were not included in many models. The presence of heavy tailed distributions in workload models was particularly important as it contributed to the self-similar nature of web traffic.

We implemented a web traffic measurement system which measured and extracted parameter datasets for our workload model in real-time. The system used low-end hardware and did not store any private user information to secondary storage. The measurement system overcame the problem of processing and storing very large volumes of traffic by selectively extracting information from HTTP, TCP and IP packets. Other web traffic measurement systems commonly captured, processed and stored every TCP/IP packet transmitted between web clients and servers in order to reconstruct HTTP dialogues between clients and servers. The processing and storage requirements of such system were enormous.

We showed that it was possible to reconstruct HTTP dialogues between web clients and servers by processing the incomplete data captured by our measurement system. The system we implemented extracted parameter datasets for our nine workload model parameters. The tool reconstructed dialogues between web clients and servers based on the incomplete information extracted from packets. The most difficult problem the tool solved was to decide whether a request was generated by a web user or a web client. It used a novel heuristic algorithm which categorised a request based on characteristics of the request obtained from the measurement data. The algorithm was novel as it was based on a list of characteristics of web client requests which we compiled by studying packet traces of web traffic. Using the heuristic algorithm it was possible to accurately extract datasets for our workload model parameters from the incomplete information captured by the measurement

system.

The real-time tool failed to extract parameter datasets during peak traffic times on the campus network. We implemented a similar system to the real-time measurement system which processed data off-line after measurements were completed. The off-line system captured packet traces of selected information contained in TCP, IP and HTTP headers. We used the measured data to reconstruct HTTP dialogues between web clients and servers off-line. From these reconstructed dialogues we extracted datasets for our nine model parameters.

We addressed privacy concerns raised by the off-line measurement system recording sensitive personal information of web users to file as follows: We extracted parameter datasets from measured data and deleted the measured data which contained sensitive material under the supervision of the network system administrator. The parameter datasets did not contain sensitive information.

The off-line measurement system gracefully dealt with protocol errors and missing information. In both cases error codes were recorded and appropriate actions taken to assure the completion of processing. The main challenges we overcame in implementing the measurement system were:

- Managing very large volumes of traffic.
- Handling protocol errors.
- Dealing with missing information.
- Synchronising measurement machines.
- Ensuring privacy of network users.

The distinguishing factor of our measurement system was that it recorded limited information but was able to extract the data necessary for our analysis. The storage requirements of recording all the information transmitted between web clients and servers were enormous. In our case the measured network link ran at approximately 6Mbps (full capacity) during the day. A storage requirement of approximately 32GB was needed to record one day's traffic measurement data. Our traffic measurement system recorded 30 days of traffic data using only 25GB. The measurement system enabled us to study web traffic generated by thousands of hosts on a campus network.

We implemented several statistical analysis techniques in the R statistical analysis environment. The package was well suited for the analysis of large datasets and enabled us to perform complicated analysis routines on datasets with up to 15 million entries. Several of the routines ran for several hours on these datasets. We completed the analysis on a machine with 512MB of memory by using strict memory management.

We implemented the Anderson Darling and λ^2 statistics in the R statistical analysis environment. The λ^2 statistic was implemented by calculating the expected number of observations per bin from the cumulative distribution function of the distribution being tested for. The expected number of observations were calculated over the range of values contained in the empirical data. Values outside this range were not included in the calculation of the statistic. We found that it was necessary to

adjust the bin size of skewed distributions when using the λ^2 statistic. We used the "skewness factor" given by Scott [Sco92] to adjust bin sizes for positively skewed distributions. Using the λ^2 statistic we found analytic distributions which fit the workload model parameters.

We found that the Anderson Darling test was not suitable for analysing datasets with more than 200 observations. We used the λ^2 statistic instead to analyse datasets with more than 200 observations. The Q-Q plot was an invaluable tool to visually assess the goodness-of-fit of data to analytic distributions. We used the λ^2 goodness-of-fit statistic in combination with the Q-Q plot to determine whether a distribution fitted the data.

The effect of local caching on the size of response messages was not taken into account in our original workload model. In order to take this factor into account we split two of the nine workload model parameters into two separate parameters each. The `Web User` and `Web Client Response Size` parameters were each split into a cached and non-cached parameter. Our model was extended to have eleven instead of nine parameters.

The ultimate goal of the work was to derive a workload model of traffic generated by an individual browsing the web. We succeeded in this goal. We derived such a model by studying traffic traces, and found analytic distributions for the parameters of the model. These distributions were listed in Tables 48 and 49 (pages 124 and 125). Random numbers could be generated for simulation studies by using the information in these tables.

We extracted parameter datasets from measured data for the eleven model parameters. These datasets could be used to generate random numbers for simulation purposes. Tables 51-53 report values, extracted from the parameter datasets, which could be used to generate random values for each of the eleven parameter datasets. The tables contain 40 values for each parameter. Interpolation could be used to generate values which fall in between the values reported in the tables. Tables containing 100, 1000, and 10000 values are available on request from the author if greater accuracy is required.

We constructed both an analytic and empirical workload model for an individual browsing the web.

7.2 Future Work

It is possible to implement our traffic model in the ns network simulator. The ns simulation package is a good choice for implementation for several reasons:

1. Ns is reputable simulation package widely used by the academic community.

2. Ns is well documented.

3. Ns already has several traffic generation modules implemented.

4. Ns is easily update-able by means of C++ or OTcl.

Browsing Inter-Session Time	**Number of Web User Requests per Browsing Session**	**Number of Web Client Requests per Web User Request**	**Web User Request Inter-arrival Time**
0.626759	1	1	1.863191
1.354413	1	1	3.396194
2.103947	1	1	5.280148
2.965647	1	1	7.257439
3.860197	1	1	9.513114
4.811255	2	2	12.028773
5.865073	2	2	14.841443
7.007023	2	2	18.016951
8.29422	2	2	21.628552
9.694515	2	3	25.816017
11.092213	3	3	30.513289
12.690919	3	4	35.210284
14.418169	3	4	40.168931
16.188715	4	5	44.876923
17.981162	4	5	49.315781
20.042928	4	6	53.887337
22.425237	5	7	58.639622
25.015031	5	7	63.05912
27.781746	6	8	67.719171
30.892287	6	9	72.588324
34.30906	7	11	77.856788
38.183285	7	12	83.415949
42.235154	8	13	89.332347
46.19823	8	15	94.118665
51.077654	9	16	100.662844
56.60192	10	18	108.000762
62.686798	11	19	116.165112
69.632072	12	21	122.239872
77.265855	13	23	130.610564
85.448615	14	26	141.162933
95.48949	16	28	153.39539
105.6352	17	32	168.037119
118.799054	19	35	184.204097
135.163426	22	40	205.549641
156.083167	24	45	233.639875
181.12738	28	51	270.467887
214.897044	33	60	322.027239
264.493583	40	72	405.934174
339.5119	53	97	568.68335
464.671213	99	200	899.956976

Table 51: Summarised Empirical Distribution Function for Parameter Datasets - 1

Web Client Request Inter-arrival Time	Web User Request Size	Web Client Request Size	Non-Cached Web User Response Size
245	176	214	192
363	208	227	253
635	229	237	297
1094	249	246	354
1754	264	255	399
2899	276	263	415
4472	284	270	476
6384	289	277	530
8596	294	284	572
10924	300	289	636
13483	309	295	756
16357	317	301	930
19550	324	307	1087
23108	332	313	1343
27290	342	319	1753
32121	350	326	2191
38197	358	331	2767
45253	366	337	3374
54253	374	343	4100
65217	383	348	4490
80455	391	354	5321
101230	399	360	6328
127808	407	366	7328
164380	414	372	8119
211207	420	378	8803
273926	428	385	9983
354110	436	392	11319
458595	446	400	12455
574213	458	409	14104
691295	472	419	15956
848057	488	431	17572
1082458	502	445	20070
1338620	524	462	23097
1651231	549	482	26150
2171443	582	507	27175
3083356	632	540	30464
4520071	709	589	34335
7357366	818	661	38835
15068176	1066	826	46253
299898439	1419	1419	59817

Table 52: Summarised Empirical Distribution Function for Parameter Datasets - 2

Cached Web User Response Size	**Non-Cached Web Client Response Size**	**Cached Web Client Response Size**
243	366	242
257	402	257
258	433	261
260	468	263
262	499	263
263	534	263
263	567	263
263	594	263
263	634	264
263	681	264
263	722	264
264	766	264
264	810	264
264	870	264
264	943	265
264	1036	265
264	1144	265
265	1248	265
265	1369	265
265	1511	265
265	1713	265
266	1932	265
266	2158	265
266	2388	266
266	2652	266
278	2968	266
278	3309	266
278	3671	266
279	4146	266
279	4546	266
279	5197	266
280	5953	266
280	6975	270
281	8256	279
297	9885	280
300	11964	286
300	14604	300
300	19704	300
300	31587	300
300	999613	300

Table 53: Summarised Empirical Distribution Function for Parameter Datasets - 3

Implementing our traffic model in ns provides a means of validating the traffic model. This can be done by generating traffic with the resultant traffic generator and comparing it to traffic characteristics of measured data.

The traffic generator can also be used to test the hypothesis that heavy tailed distributions of data generated by individual users browsing the web contribute to the self-similarity of aggregate web traffic. Crovella *et al.* [CB96] showed that web traffic is self-similar and that the self-similarity is in all likelihood attributable to the heavy-tailed distributions of transmission times of documents and silent times between document requests. Taqqu *et al.* [TWS97] proved that aggregate World Wide Web traffic as found on Internet links can be modelled by super-positioning many ON/OFF traffic sources where the ON and OFF periods are drawn from heavy tailed distributions. The existence of a relationship between heavy tailed distributions and self-similarity for aggregate web traffic has been confirmed by this research but the relationship between heavy tailed distributions and self-similarity has not been tested for traffic generated by individual users. An implementation of the traffic model we developed in the ns network simulator will provide a platform for investigating such a relationship.

7.2.1 Implementation of Traffic Model

Ns is implemented in the C++ and OTcl languages. C++ is used for the clarity of design which object-orientation affords as well as the speed associated with natively compiled code. OTcl is used for its functionality as scripting language. Network simulation scenarios are easily configured and updated through the OTcl interface. The class tree of the whole package is implemented in both C++ and OTcl. Traffic generation modules can be implemented in either C++ or OTcl.

In order to obtain traffic traces generated by our workload model the ns `Tracing` and `Monitoring` components have to be modified.

In order to generate traffic using our workload model the *web-cache application* class has to be updated to conform to our traffic model. The *web-cache application* has several similarities to our model:

1. Data can be transmitted from one component to another.
2. A traffic model consisting of main and inline objects is used.
3. Probabilistic functions needed to generate random numbers for inter-arrival times and sizes exist in the *web-cache application*.

The following alterations have to be made to the *web-cache application*:

1. A *Browsing Session* layer as illustrated in Figure 6 has to be added to the *web-cache application*.
2. The `HTTP/Client` class has to be updated to generate inter-arrival times according to different distributions for main and inline objects.

3. The `Http/Client` class has to be updated to wait for a main object before it generates inline object requests.

4. The `PagePool/CompMath` class has to be updated to generate the sizes of main and inline objects according to different distributions.

5. The `PagePool/CompMath` class has to be updated to generate the number of inline objects according to an appropriate distribution.

A simulation environment which aggregates traffic streams from many individual web users into a single traffic stream can be constructed in ns using the altered *web-cache application* class. Traffic generated by the simulation environment can be saved to output files via the adapted `Tracing` and `Monitoring` components.

In order to validate our traffic model, traffic generated by the simulation environment should be compared to measured traffic. The traffic model will be validated if characteristics of generated traffic correspond to characteristics of measured traffic.

In order to accept or reject the thesis that individual user web traffic generated by distributions with heavy tails cause self-similarity in aggregate web traffic streams, traffic generated by the simulation environment should be tested for self-similarity.

7.2.2 Extension of Real-time Measurement System

The real-time measurement system failed to extract parameter datasets during peak traffic hours. We believe that with some optimisation the program will succeed in extracting parameter datasets during peak time. The program is well suited to being distributed over two or more processors as it consists of two processes which are synchronised. It will be an interesting experiment to test the system on a dual-processor computer. The program is written with debugging statements which can be enabled to provide timing information. It is therefore easy to see where the system fails during processing.

The measurement system captured data generated by users browsing the web, as well as other HTTP data generated by applications which use the HTTP but not for web browsing purposes. Traffic generated by web clients such as web-download and web-irc traffic were also captured. This made the extraction of parameter datasets and subsequent analysis of these datasets very difficult. The measurement system can be extended to differentiate between these types of traffic, and to take the appropriate action for each type of traffic.

The measurement system also did not record any data generated by HTTPS which uses the secure sockets layer (SSL). Although it is not possible to inspect the content of data transferred using this protocol, the measurement system can record information about these connections for statistical purposes.

Appendix A

Measurement File Extract

Figures 44-46 show selected fields from measurement file entries for traffic generated by a single web user request. The measurement file was created by our data measurement tool. The web user request recorded in the measurement file shown in Figures 44-46 was generated by entering the URL for `CNN` into the web client's URL textbox. The request resulted in 82 web client requests for inline objects.

Figures 44-46 show the following fields:

- Arrival Time
- Host Part of Request URL
- Path Part of Request URL
- Host Part of Referer URL
- Path Part of Referer URL

These fields are used to solve the `Web User vs. Web Client Request Differentiation Problem` as discussed in Section 4.5.

```
1055766928.275495  www.cnn.com  /  no-ref  no-path
1055766929.845104  ar.atwola.com  file/adsWrapper.js  www.cnn.com  /
1055766930.100174  ar.atwola.com  file/adsWrapper.js  www.cnn.com  /
1055766931.076519  ar.atwola.com  file/adsPopup2.js  www.cnn.com  /
1055766931.163398  i.cnn.net  cnn/images/1.gif  www.cnn.com  /
1055766931.167530  i.a.cnn.net  cnn/.element/img/1.0/logo/cnn.gif  www.cnn.com  /
1055766931.179070  ar.atwola.com html/93103300/276280765/aol?SNM=HID&width=468&height=60
                   &target=_top&TZ=-120&TVAR=class%3Dintl&CT=I  www.cnn.com  /
```

Figure 44: Selected Fields Taken from Measurement File for a Web User Request and Subsequent Web Client Requests - 1

```
1055766931.208776  i.a.cnn.net  cnn/.element/img/1.0/searchbar/bar.search.gif  www.cnn.com  /
1055766931.211891  i.a.cnn.net  cnn/.element/img/1.0/searchbar/bar.top.bevel.gif  www.cnn.com  /
1055766931.767854  i.a.cnn.net  cnn/.element/img/1.0/searchbar/bar.right.bevel.gif  www.cnn.com  /
1055766932.052149  i.a.cnn.net  cnn/.element/img/1.0/searchbar/bar.google.gif  www.cnn.com  /
1055766932.056512  i.a.cnn.net  cnn/.element/img/1.0/searchbar/bar.bottom.bevel.gif  www.cnn.com  /
1055766932.648050  i.a.cnn.net  cnn/.element/img/1.0/main/nav_at_money.gif  www.cnn.com  /
1055766932.657479  i.a.cnn.net  cnn/.element/img/1.0/main/nav_at_si.gif  www.cnn.com  /
1055766932.663193  ar.atwola.com  html/93170132/276280765/aol?SNM=HID&width=120&height=90&target=_top
                   &TZ=-120&TVAR=class%3Dintl&CT=I  www.cnn.com  /
1055766932.777874  i.a.cnn.net  cnn/.element/img/1.0/sect/SEARCH/nav.search.gif  www.cnn.com  /
1055766933.288994  i.cnn.net  cnn/.element/img/1.0/misc/premium.gif  www.cnn.com  /
1055766933.289349  i.cnn.net  cnn/2003/LAW/06/15/tulia.suspects/top.main.tulia.jpg  www.cnn.com  /
1055766933.292323  i.a.cnn.net  cnn/.element/img/1.0/main/px_c00.gif  www.cnn.com  /
1055766933.328260  i.a.cnn.net  cnn/.element/img/1.0/main/userpicks.gif  www.cnn.com  /
1055766933.401499  i.a.cnn.net  cnn/.element/img/1.0/main/px_ccc.gif  www.cnn.com  /
1055766934.009960  i.a.cnn.net  cnn/.element/img/1.0/misc/audio.gif  www.cnn.com  /
1055766934.068426  i.a.cnn.net  cnn/images/1.gif  www.cnn.com  /
1055766934.788522  i.a.cnn.net  cnn/.element/img/1.0/main/more.video.blue.gif  www.cnn.com  /
1055766934.918460  i.a.cnn.net  cnn/images/icons/premium.gif  www.cnn.com  /
1055766935.408524  i.a.cnn.net  cnn/video/law/2003/06/15/bf.scotus.sodomy.vs.cnn.jpg  www.cnn.com  /
1055766935.638216  i.cnn.net  cnn/.element/img/1.0/main/px_c00.gif  www.cnn.com  /
1055766935.670040  i.a.cnn.net  cnn/.element/img/1.0/main/superlinks/sports.gif  www.cnn.com  /
1055766935.680084  i.cnn.net  cnn/2003/images/06/16/tz.spurs.ap.jpg  www.cnn.com  /
1055766936.033816  i.a.cnn.net  cnn/.element/img/1.0/main/at_cnnmoney.gif  www.cnn.com  /
1055766936.285528  i.a.cnn.net  cnn/.element/img/1.0/main/business.news.blue.gif  www.cnn.com  /
1055766936.303105  ar.atwola.com  image/93103306/aol  www.cnn.com  /
1055766936.589929  i.cnn.net  cnn/.element/img/1.0/main/arrow_down.gif  www.cnn.com  /
1055766936.592977  i.a.cnn.net  cnn/.element/img/1.0/main/ftn.280x32.ny.times.gif  www.cnn.com  /
1055766936.878700  i.a.cnn.net  cnn/.element/img/1.0/main/ftn.345x32.breaking.news.gif  www.cnn.com  /
1055766936.949734  ar.atwola.com  content/B0/0/H7pTL2Luf0_kw3xmlj8W1sns8a9RRNke8_SAqLzKBa609jmULHVa8jg
                   FKtiL69KX71YyF9xayBuNZf_HnE4-913BJZKLmiPO6hUDYJ_O4lk$/aol  ar.atwola.com  html/9310
                   3300/276280765/aol?SNM=HID&width=468&height=60&target=_top&TZ=-120&TVAR=class%3Dint
                   l&CT=I
1055766937.360944  i.cnn.net  cnn/.element/img/1.0/main/superlinks/cnn_presents.gif  www.cnn.com  /
1055766937.364218  i.a.cnn.net  cnn/CNN/Programs/presents/shows/seeds.of.terror/images/tz.seeds.of.
                   terror.jpg  www.cnn.com  /
1055766937.720672  i.a.cnn.net  cnn/.element/img/1.0/main/superlinks/business.gif  www.cnn.com  /
1055766937.748008  i.a.cnn.net  cnn/2003/SHOWBIZ/Movies/05/30/sprj.cas03.review.nemo/tz.finding.nemo
                   .jpg www.cnn.com  /
1055766938.202319  i.a.cnn.net  cnn/.element/img/1.0/main/us.gif  www.cnn.com  /
1055766938.243148  i.a.cnn.net  cnn/.element/img/1.0/main/world.gif  www.cnn.com  /
1055766938.246930  i.cnn.net  cnn/.element/img/1.0/misc/icon.external.links.gif  www.cnn.com  /
1055766938.252206  i.a.cnn.net  cnn/.element/img/1.0/main/technology.gif  www.cnn.com  /
1055766938.256196  i.a.cnn.net  cnn/.element/img/1.0/main/entertainment.gif  www.cnn.com  /
1055766939.029355  i.a.cnn.net  cnn/.element/img/1.0/main/politics.gif  www.cnn.com  /
1055766939.089859  i.a.cnn.net  cnn/.element/img/1.0/main/law.gif  www.cnn.com  /
1055766939.498793  i.a.cnn.net  cnn/.element/img/1.0/main/health.gif  www.cnn.com  /
1055766939.591628  i.a.cnn.net  cnn/.element/img/1.0/main/space.gif  www.cnn.com  /
1055766939.619819  i.a.cnn.net  cnn/.element/img/1.0/main/travel.gif  www.cnn.com  /
1055766940.050311  i.a.cnn.net  cnn/.element/img/1.0/main/education.gif  www.cnn.com  /
1055766940.190434  i.a.cnn.net  cnn/.element/img/1.0/main/sports.gif  www.cnn.com  /
1055766940.210413  i.a.cnn.net  cnn/.element/img/1.0/main/business.gif  www.cnn.com  /
1055766941.000732  i.a.cnn.net  cnn/.element/img/1.0/misc/diagonal.gif  www.cnn.com  /
1055766941.003855  i.a.cnn.net  cnn/.element/img/1.0/main/tv.schedule.gif  www.cnn.com  /
1055766941.020428  i.cnn.net  cnn/CNN/Programs/american.morning/images/2003/06/tz.iraq.ap.jpg
                   www.cnn.com  /
1055766941.026831  i.a.cnn.net  cnn/.element/img/1.0/main/superlinks/us.gif  www.cnn.com  /
1055766941.674866  i.a.cnn.net  cnn/2003/US/06/16/nyt.safire/tz.fcc.jpg  www.cnn.com  /
1055766941.791149  i.a.cnn.net  cnn/.element/img/1.0/main/superlinks/offbeat.gif  www.cnn.com  /
1055766941.794580  i.a.cnn.net  cnn/2003/WORLD/asiapcf/south/06/15/offbeat.wedding.reut/tz.wedding.
                   gradient.jpg  www.cnn.com  /
1055766941.831598  i.a.cnn.net  cnn/.element/img/1.0/main/quickvote.gif  www.cnn.com  /
1055766941.998885  ar.atwola.com  image/93101912/aol  www.cnn.com  /
```

Figure 45: Selected Fields Taken from Measurement File for Web User Request and Subsequent Web Client Requests - 2

```
1055766942.391177  i.a.cnn.net  cnn/.element/img/1.0/main/icon_external.gif  www.cnn.com  /
1055766942.630637  i.a.cnn.net  cnn/.element/img/1.0/main/partner_time.gif  www.cnn.com  /
1055766942.881244  i.a.cnn.net  cnn/.element/img/1.0/main/partner_si.gif  www.cnn.com  /
1055766943.070420  i.a.cnn.net  cnn/.element/img/1.0/main/partners_nyt.gif  www.cnn.com  /
1055766943.334770  i.a.cnn.net  cnn/.element/img/1.0/sect/SEARCH/dotted.line.gif  www.cnn.com  /
1055766943.529759  i.a.cnn.net  cnn/.element/img/1.0/misc/icon.external.links.gif  www.cnn.com  /
1055766943.532789  ar.atwola.com  html/93103308/276280765/aol?SNM=HID&width=88&height=31&target=_top&
                   TZ=-120&TVAR=class%3Dintl&CT=I  www.cnn.com  /
1055766943.611058  i.a.cnn.net  cnn/.element/img/1.0/main/icon_premium.gif  www.cnn.com  /
1055766943.614448  ar.atwola.com  html/93162673/276280765/aol?SNM=HID&height=300&width=720&target=_
                   blank&CT=J&TZ=-120&htmlpre=adsObj0%3d%27&xlnl=%5cn&xltick=%5c%27&ctype=application
                   /x-javascript&htmlsuf=%27%3badsPopup%280%29&TVAR=class%3Dintl  www.cnn.com  /
1055766943.761865  i.cnn.net  cnn/.element/img/1.0/main/tab_gradient_bg.gif  www.cnn.com  /
1055766943.765055  i.cnn.net  cnn/.element/img/1.0/main/market_bg.jpg  www.cnn.com  /
1055766943.768561  ar.atwola.com  content/B0/0/H7pTL2Luf0_kw3xmlj8W1sns8a9RRNke8_SAqLzKBa609jmULHVa8j
                   gFKtiL69KX71YyF9xayBsGJJ19jhX9-1GwpwGSE_3Fdi_LakYKyQ4$/aol ar.atwola.com  html/931
 70132/i276280765/aol?SNM=HID&width=120&height=90&target=_top&TZ=-120&TVAR=class%3D
 intl&CT=I
1055766944.240712  ar.atwola.com  content/B0/0/H7pTL2Luf0_kw3xmlj8W1sns8a9RRNke8_SAqLzKBa609jmULHVa8j
                   gFKtiL69KXTPCmjMkYOvM5C_xF3lKNjRbUvMSjKyxE7A6B3LIHANk$/aol  www.cnn.com  /
1055766944.244819  ar.atwola.com  content/B0/0/H7pTL2Luf0_kw3xmlj8W1sns8a9RRNke8_SAqLzKBa609jmULHVa8j
                   gFKtiL69KXD5cML0fB7YPtL4GXa6aV7CmZ3HS-849cbJbLDQoJB_k$/aol  www.cnn.com  /
1055766944.256720  ar.atwola.com  content/B0/0/H7pTL2Luf0_kw3xmlj8W1sns8a9RRNke8_SAqLzKBa609jmULHVa8j
                   gFKtiL69KXxTQ8xGluxHcJ2ou_yHWqQUTLN-mqmkAVnojDlh4zLpw$/aol  ar.atwola.com  html/93
 103308/276280765/aol?SNM=HID&width=88&height=31&target=_top&TZ=-120&TVAR=class%3Di
 ntl&CT=I
1055766944.516772  www.cnn.com  cnn_adspaces/adsPopup2.html?0  www.cnn.com  /
1055766944.554569  ar.atwola.com  html/93137910/276280765/aol?SNM=HID&width=101&height=1&target=_top&
                   TZ=-120&TVAR=class%3Dintl&CT=J&hw=docw  www.cnn.com  /
1055766945.739029  ar.atwola.com  file/adsEnd.js  www.cnn.com  /
1055766945.756202  www.cnn.com  cookie.crumb  www.cnn.com  /
1055766945.862627  ar.atwola.com  content/B0/0/H7pTL2Luf0_kw3xmlj8W1sns8a9RRNke8_SAqLzKBa609jmULHVa8j
                   gFKtiL69KXkmxkNXzSYZcMpAhGYOhIdLwaLt1RyxQlfaaMvo09EHk$/aol  www.cnn.com  cnn_adspa
 ces/ adsPopup2.html?0
1055766968.511389  i.cnn.net  cnn/.element/img/1.0/main/weather_bg.jpg  www.cnn.com  /
```

Figure 46: Selected Fields Taken from Measurement File for Web User Request and Subsequent Web Client Requests - 3

Appendix B

Implementation of Heuristic Algorithm

Figures 47 and 48 show the implementation of Group No. 1 characteristics.

```
if(this_filetype == GRAPHICS)                                  /* Characteristic No.  1 */
  for(all previous requests in all_request_queue)
    if(arrival_time_difference >= 10)
      break
    else if(this_referer_url == prev_referer_url){
      this_category = Web_Client_Request
      break
    }
else                                                           /* Characteristic No.  2 */
  for(all previous requests in all_request_queue)
    if(arrival_time_difference >= 2)
      break
    else if(this_referer_url == prev_referer_url){
      this_category = Web_Client_Request
      break
    }
if(file not classified && (this_filetype == GRAPHICS))   /* Characteristic No.  3 */
  for(all previous requests in all_request_queue)
    if(arrival_time_difference >= 5)
      break
    else if(this_host_part_request_url == prev_host_part_request_url){
      this_category = Web_Client_Request
      break
    }
```

Figure 47: Implementation of Heuristic Algorithm - Group No. 1 Characteristics

Figure 49 shows the implementation of Group No. 2 characteristics.

Figure 50 shows the implementation of Group No. 3 characteristics.

```
 else if(file not classified)                              /* Characteristic No.  4 */
   for(all previous requests in all_request_queue)
     if(arrival_time_difference >= 2)
       break
     else if(this_host_part_request_url == prev_host_part_request_url){
       this_category = Web_Client_Request
       break
     }
```

Figure 48: Implementation of Heuristic Algorithm - Group No. 1 Characteristics

```
if(file not classified)
  if(this_filetype == GRAPHICS)
    for(all previous requests in html_request_queue)              /* Characteristic No.  5 */
      if(arrival_time_difference >= 50)
        break
      if(this_referer_url == prev_referer_url){
        this_category = Web_Client_Request
        break
      }
    if(file not classified)
      for(all previous document requests in html_request_queue)   /* Characteristic No.  6 */
        if(arrival_time_difference >= 50)
          break
        if(this_referer_url == prev_request_url){
          this_category = Web_Client_Request
          break
        }
  else
    for(all previous requests in html_request_queue)              /* Characteristic No.  7 */
      if(arrival_time_difference >= 2)
        break
      if(this_referer_url == prev_referer_url){
        this_category = Web_Client_Request
        break
      }
    if(file not classified)
      for(all previous requests in html_request_queue)            /* Characteristic No.  8 */
        if(arrival_time_difference >= 10)
          break
        if(this_referer_url == prev_request_url){
            this_category = Web_Client_Request
            break
        }
```

Figure 49: Implementation of Heuristic Algorithm - Group No. 2 Characteristics

```
if(file not classified && (this_filetype == GRAPHICS))
  for(all previous requests in all_request_queue)              /* Characteristic No.  9 */
    if(arrival_time_difference >= 60)
      break
    if(this_referer_url == prev_referer_url){
      this_category = Web_Client_Request
      break
    }
  if(file not classified)
    for(all previous requests in all_request_queue)             /* Characteristic No.  10 */
      if(arrival_time_difference >= 60)
        break
      if(this_request_host_url == prev_request_host_url){
        this_category = Web_Client_Request
        break
      }
```

Figure 50: Implementation of Heuristic Algorithm - Group No. 3 Characteristics

Appendix C

R Code

R code referenced in the text is listed here. We implemented several "Exploratory Data Analysis" routines which plot different aspects of the data. These routines are listed in Section C.1. We implemented two "goodness-of-fit" statistics, the λ^2 and Anderson Darling statistics. The code for the goodness-of-fit statistics are listed in Section C.2.

C.1 Visual Techniques

C.1.1 Log Empirical Complementary Cumulative Distribution Function Plot

```
last <- function(x) {
   max(x)
}

leccdf <- function(data) {
   sorted <- sort(data)
   sample.size <- length(sorted)
   count <- 1
   accumulator <- 1/sample.size
   ccdf <- numeric()
   for(j in sorted){
      ccdf[count] <- accumulator
      accumulator <- accumulator + 1/sample.size
      count <- count +1
   }
   sorted.factor <- factor(sorted)
   ccdf.agg <- tapply(ccdf, sorted.factor,
      last, simplify = TRUE)
   ccdf.final <- ccdf.agg[1:length(ccdf.agg)-1]
   ccdf.final.inverted <- 1.0-ccdf.final
   ccdf.final.inverted.log <- log(ccdf.final.inverted)
```

```
   sorted.uniq <- unique(sorted)
   sorted.uniq <- sorted.uniq[1:length(sorted.uniq)-1]
   plot(sorted.uniq, ccdf.final.inverted.log, main="", xlab="Interarrival
      Times", ylab="Log(1-Fn(x))", type="l")
}
```

C.1.2 P-P and Q-Q Plots for Selected Distributions

The R Code for the P-P and Q-Q Plots are shown in Figures 51 - 52

C.2 Goodness-of-fit Techniques

C.2.1 Lambda Discrepancy Measure

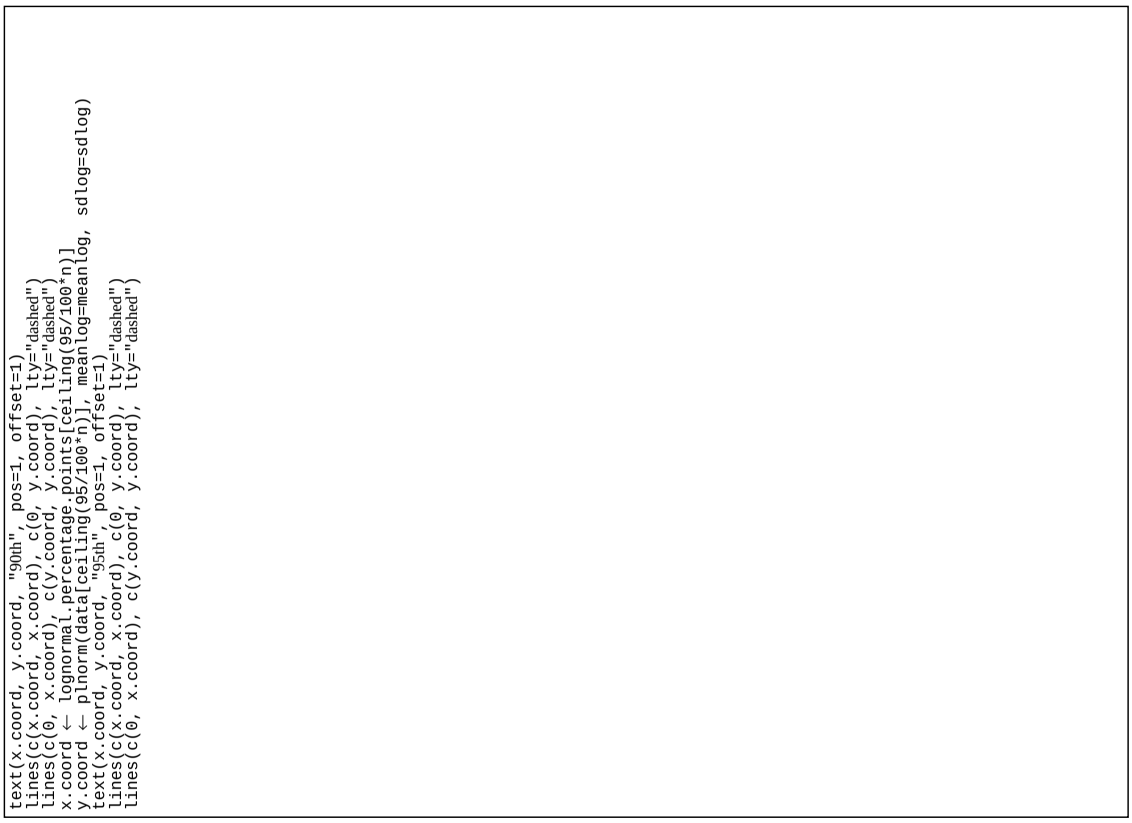
The R Code for the Lambda Discrepancy Measure is shown in Figures 53 - 57

C.2.2 Anderson Darling Test

The R Code for the Anderson Darling Test is shown in Figures 58 - 60.

```
# P-P and Q-Q Plot routines
# Lourens Walters - Cape Town University
# 15/04/2003

library(stepfun)
n ← 43336
gamma ← 0.7437
alpha ← 54.902209
zeta ← 3.242876
sigma ← 1.637514
meanlog ← zeta
sdlog ← sigma
data ← scan("../all_hosts_moved", n)
data ← sort(data)

hist(data, xlim=c(0.0, 465.0), xlab="Browsing Inter-Session Time", ylab="Bin Count", main
=paste("Histogram"))
hist(data, xlim=c(0.0, 465.0), xlab="Browsing Inter-Session Time", ylab="Bin Count", main
=paste("Histogram"))
cutoff.percentile ← pweibull(465, shape=gamma, scale=alpha)
num.weibull.points ← ceiling((1-cutoff.percentile)*n)+n
weibull.percentage.points ← ppoints(num.weibull.points)
weibull.percentage.points ← weibull.percentage.points[1:n]
weibull.quantiles ← qweibull(weibull.percentage.points, shape=gamma, scale=alph
a)
plot(data, weibull.quantiles, xlab = "Sample Quantiles", ylab = "Theoretical Quantiles", t
ype="p", col="blue", main=paste("Weibull Q-Q Plot"))
abline(a=0, b=1, col="red")
x.coord ← data[ceiling(50/100*n)]
y.coord ← qweibull(weibull.percentage.points[ceiling(50/100*n)], shape=gamma, s
cale=alpha)
text(x.coord, y.coord, "50th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← data[ceiling(75/100*n)]
y.coord ← qweibull(weibull.percentage.points[ceiling(75/100*n)], shape=gamma, s
cale=alpha)
text(x.coord, y.coord, "75th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← data[ceiling(90/100*n)]
y.coord ← qweibull(weibull.percentage.points[ceiling(90/100*n)], shape=gamma, s
cale=alpha)
text(x.coord, y.coord, "90th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← data[ceiling(95/100*n)]
y.coord ← qweibull(weibull.percentage.points[ceiling(95/100*n)], shape=gamma, s
cale=alpha)
text(x.coord, y.coord, "95th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
linear.fit ← (1-sum((data-weibull.quantiles)^2))/sqrt(sum(data^2)*sum(weibull.q
uantiles^2))
write(print(paste(linear.fit)), file="linear_fit.txt", append=F)

cutoff.percentile ← plnorm(465, meanlog=meanlog, sdlog=sdlog)
num.lognorm.points ← ceiling((1-cutoff.percentile)*n)+n
lognormal.percentage.points ← ppoints(num.lognorm.points)
lognormal.percentage.points ← lognormal.percentage.points[1:n]
lognorm.quantiles ← qlnorm(lognormal.percentage.points, meanlog=meanlog, sdlog=
sdlog)
plot(data, lognorm.quantiles, xlab = "Sample Quantiles", ylab = "Theoretical Quantiles",
type="p", col="blue", main=paste("Lognormal Q-Q Plot"))
abline(a=0, b=1, col="red")
x.coord ← data[ceiling(50/100*n)]
y.coord ← qlnorm(lognormal.percentage.points[ceiling(50/100*n)], meanlog=meanlo
g, sdlog=sdlog)
text(x.coord, y.coord, "50th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← data[ceiling(75/100*n)]
y.coord ← qlnorm(lognormal.percentage.points[ceiling(75/100*n)], meanlog=meanlo
g, sdlog=sdlog)
text(x.coord, y.coord, "75th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← data[ceiling(90/100*n)]
y.coord ← qlnorm(lognormal.percentage.points[ceiling(90/100*n)], meanlog=meanlo
g, sdlog=sdlog)
text(x.coord, y.coord, "90th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← data[ceiling(95/100*n)]
y.coord ← qlnorm(lognormal.percentage.points[ceiling(95/100*n)], meanlog=meanlo
g, sdlog=sdlog)
text(x.coord, y.coord, "95th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
linear.fit ← (1-sum((data-lognorm.quantiles)^2))/sqrt(sum(data^2)*sum(lognorm.q
uantiles^2))
write(print(paste(linear.fit)), file="linear_fit.txt", append=T)

weibull.percentages ← pweibull(data, shape=gamma, scale=alpha)
plot(weibull.percentage.points, weibull.percentages, xlab = "Sample Percentages", yl
ab = "Theoretical Percentages", type="p", col="blue", main=paste("Weibull P-P Plot"))
abline(a=0, b=1, col="red")
x.coord ← weibull.percentage.points[ceiling(50/100*n)]
y.coord ← pweibull(data[ceiling(50/100*n)], shape=gamma, scale=alpha)
text(x.coord, y.coord, "50th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← weibull.percentage.points[ceiling(75/100*n)]
y.coord ← pweibull(data[ceiling(75/100*n)], shape=gamma, scale=alpha)
text(x.coord, y.coord, "75th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← weibull.percentage.points[ceiling(90/100*n)]
y.coord ← pweibull(data[ceiling(90/100*n)], shape=gamma, scale=alpha)
text(x.coord, y.coord, "90th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← weibull.percentage.points[ceiling(95/100*n)]
y.coord ← pweibull(data[ceiling(95/100*n)], shape=gamma, scale=alpha)
text(x.coord, y.coord, "95th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")

lognorm.percentages ← plnorm(data, meanlog=meanlog, sdlog=sdlog)
plot(lognormal.percentage.points, lognorm.percentages, xlab = "Sample Percentages",
ylab = "Theoretical Percentages",  type="p", col="blue", main=paste("Lognormal P-P Plot"))
abline(a=0, b=1, col="red")
x.coord ← lognormal.percentage.points[ceiling(50/100*n)]
y.coord ← plnorm(data[ceiling(50/100*n)], meanlog=meanlog, sdlog=sdlog)
text(x.coord, y.coord, "50th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← lognormal.percentage.points[ceiling(75/100*n)]
y.coord ← plnorm(data[ceiling(75/100*n)], meanlog=meanlog, sdlog=sdlog)
text(x.coord, y.coord, "75th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← lognormal.percentage.points[ceiling(90/100*n)]
y.coord ← plnorm(data[ceiling(90/100*n)], meanlog=meanlog, sdlog=sdlog)
```

Figure 51: P-P and Q-Q Plot Code - 1



```
text(x.coord, y.coord, "90th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
x.coord ← lognormal.percentage.points[ceiling(95/100*n)]
y.coord ← plnorm(data[ceiling(95/100*n)], meanlog=meanlog, sdlog=sdlog)
text(x.coord, y.coord, "95th", pos=1, offset=1)
lines(c(x.coord, x.coord), c(0, y.coord), lty="dashed")
lines(c(0, x.coord), c(y.coord, y.coord), lty="dashed")
```

Figure 52: P-P and Q-Q Plot Code - 2

```
# Lambda Squared Test Routines
# Lourens O. Walters - Cape Town University
# 16/05/2004

library(evd)
library(MASS)

# Extra distributions not defined in R

beta.distr ← function(x, alpha, beta, a, b, log=FALSE){
  range ← b - a
  start ← a
  standardized.x ← (x-start)/range
  if(length(standardized.x[standardized.x≤0])≠0){
    print(paste("x:", x, "standardized.x:", standardized.x))
    stop("Error, negative value in Beta quantiles")
  }
  probability ← pbeta(standardized.x, shape1=alpha, shape2=beta)
  if(log)
    return(log(probability))
  else
    return(probability)
}

beta.density ← function(x, alpha, beta, a, b, log=FALSE){
  range ← b - a
  start ← a
  standardized.x ← (x-start)/range
  if(length(standardized.x[standardized.x<!0])≠0)
    stop("Error, negative value in Beta quantiles")
  standard.density ← dbeta(standardized.x, shape1=alpha, shape2=beta)
  density ← standard.density/range
  if(log)
    return(log(density))
  else
    return(density)
}

beta.variates ← function(n, alpha, beta, a, b){
  range ← b - a
  start ← a
  standard.variates ← rbeta(n, shape1=alpha, shape2=beta)
  variates ← (standard.variates*range) + start
  return(variates)
}

pareto.distr ← function(x, alpha, beta, log=FALSE){
  # Pareto distribution not defined for x < alpha
  # Test for values x < alpha, taking into account floating point error
  min.val ← min(x)
  error.val ← min.val - alpha
  if (error.val < 0){
    warning("\nx < alpha in pareto.distr(x, alpha, beta, log=FALSE)")
  }
  probability ← 1-(alpha/x)^beta
  if(log)
    return(log(probability))
  else
    return(probability)
}

pareto.density ← function(x, alpha, beta, log=FALSE){
  # Pareto distribution not defined for x < alpha
  # Test for values x < alpha, taking into account floating point error
  min.val ← min(x)
  error.val ← min.val - alpha
  if (error.val < 0)
    warning("\nx < alpha in pareto.density(x, alpha, beta, log=FALSE)")
  density ← beta * (alpha^beta) * (x^(-beta-1))
  if(log)
    return(log(density))
  else
    return(density)
}

pareto.variates ← function(alpha, beta, cnt=1000){
  u ← runif(cnt)
  variates ← (alpha/((u^(1/beta))))
  return(variates)
}

# Bin data using formulae defined by Scott. Adapted formula based on
# assumption that data have lognormal distribution.

bin.actual ← function(data, bin.method){
  length ← length(data)
  std.dev ← sqrt(var(data))
  min.val ← floor(min(data))
  max.val ← ceiling(max(data))
  if (bin.method ≡ "scott"){
    bin.width ← 3.49 * std.dev * length^(-1/3)
    if (bin.width < 1){
      bin.width ← 1
    }
    else{
      bin.width ← round(bin.width, digits=0)
    }
  }
  else if(bin.method ≡ "scott-adapted"){
    std.dev ← sd(data)
    normal.data ← log(data)
    n.std.dev ← sd(normal.data)
    n.var ← n.std.dev^2
    numerator ← 2^(1/3)*n.std.dev
    denominator ← exp(5*n.var/4)*(n.var+2)^(1/3)*(exp(n.var)-1)^(1/2)
    skew.factor ← numerator/denominator
    bin.width ← skew.factor * 3.49 * std.dev * length^(-1/3)
    if (bin.width < 1){
      bin.width ← 1
    }
    else{
      bin.width ← round(bin.width, digits=0)
    }
  }
  else{
    stop("\nERROR: invalid bin.method specified - choose one of the
      following: \n1. scott\n 2. scott-adapted\n")
  }
  num.bins ← ceiling((max.val-min.val)/bin.width)
  max.val ← min.val+(bin.width*num.bins)
  cut.data ← cut(data, seq(min.val, max.val, bin.width), include.lowest =
    TRUE)
  actual ← tabulate(cut.data, nbins=length(levels(cut.data)))
  return(list(min.val=min.val, max.val=max.val, bin.width=bin.width,
    num.bins=num.bins, actual=actual))
}

# Calculate expected values for bins, based on distribution being tested for.
bin.expected ← function(data.list){
  expected ← numeric()
  if(data.list$distribution ≡ "lognormal"){
    if(is.na(data.list$param1) || is.na(data.list$param2)){
      zeta ← mean(log(data.list$observed))
      sigma ← sd(log(data.list$observed))
```

Figure 53: Lambda Discrepancy Measure - Code 1

```
    result.list ← fitdistr(data.list$observed, "lognormal",
      start=list(meanlog=zeta, sdlog=sigma))
    zeta ← result.list$estimate[[1]]
    sigma ← result.list$estimate[[2]]
  } else {
    zeta ← data.list$param1
    sigma ← data.list$param2
  }
  length ← length(data.list$observed)
  df ← length−1−2
  breaks ← seq(data.list$min.val, data.list$max.val,
    by=data.list$bin.width)

  for(i in 2:length(breaks)){
    expected[i−1] ← plnorm(breaks[i], meanlog=zeta,
      sdlog=sigma)−plnorm(breaks[i−1], meanlog=zeta, sdlog=sigma)
    expected[i−1] ← expected[i−1]*length
  }
  parameters ← list(zeta=zeta, sigma=sigma)
  returnlist ← list(expected=expected, parameters=parameters, df=df)
}
else if(data.list$dist ≡ "weibull"){
  if(is.na(data.list$param1) || is.na(data.list$param2)){
    result.list ← fitdistr(data.list$observed, "weibull")
    gamma ← result.list$estimate[1]
    alpha ← result.list$estimate[2]
  } else {
    gamma ← data.list$param1
    alpha ← data.list$param2
  }
  length ← length(data.list$observed)
  df ← length−1−2
  breaks ← seq(data.list$min.val, data.list$max.val, data.list$bin.width)
  for(i in 2:length(breaks)){
    expected[i−1] ← pweibull(breaks[i], shape=gamma,
      scale=alpha)−pweibull(breaks[i−1], shape=gamma, scale=alpha)
    expected[i−1] ← expected[i−1]*length
  }
  parameters ← list(gamma=gamma, alpha=alpha)
  returnlist ← list(expected=expected, parameters=parameters, df=df)
}
else if(data.list$dist ≡ "exponential"){
  if(is.na(data.list$param1)){
    beta ← mean(data.list$observed)
    rate ← 1/beta
    log.rate ← log(rate)
    expLoglik ← function(params, negative=TRUE){
      lglk ← sum(dexp(data.list$observed, rate=exp(params[1]), log=TRUE))
      if(negative)
        return(−lglk)
      else
        return(lglk)
    }
    optim.list ← optim(c(log.rate), expLoglik, method="BFGS")
    if(optim.list$convergence≡0){
      rate ← exp(optim.list$par[1])
      beta ← 1/rate
    }else{
      warning("optim didn't converge in bin.expected − exponential")
    }
  } else {
    rate ← data.list$param1
    beta ← 1/rate
  }
  length ← length(data.list$observed)
  df ← length−1−1
  breaks ← seq(data.list$min.val, data.list$max.val, data.list$bin.width)
```

Figure 54: Lambda Discrepancy Measure - Code 2

```
  for(i in 2:length(breaks)){
    expected[i−1] ← pexp(breaks[i], rate=rate)−pexp(breaks[i−1], rate=rate)
    expected[i−1] ← expected[i−1]*length
  }
  parameters ← list(beta=beta)
  returnlist ← list(expected=expected, parameters=parameters, df=df)
}
else if(data.list$dist ≡ "gamma"){
  if(is.na(data.list$param1) || is.na(data.list$param2)){
    mean.data ← mean(data.list$observed)
    var.data ← var(data.list$observed)
    shape ← mean.data^2/var.data
    scale ← var.data/mean.data
    log.shape ← log(shape)
    log.scale ← log(scale)
    gammaLoglik ← function(params, negative=TRUE){
      lglk ← sum(dgamma(data.list$observed, shape=exp(params[1]),
        scale=exp(params[2]), log=TRUE))
      if(negative)
        return(−lglk)
      else
        return(lglk)
    }
    optim.list ← optim(c(log.shape, log.scale), gammaLoglik)
    if(optim.list$convergence≡0){
      gamma ← exp(optim.list$par[1])
      alpha ← 1/exp(optim.list$par[2])
    }else{
      warning("optim didn't converge in bin.expected − gamma")
      gamma ← shape
      alpha ← 1/scale
    }
  } else{
    gamma ← data.list$param1
    scale ← data.list$param2
    alpha ← 1/scale
  }
  length ← length(data.list$observed)
  df ← length−1−2
  breaks ← seq(data.list$min.val, data.list$max.val, data.list$bin.width)
  for(i in 2:length(breaks)){
    expected[i−1] ← pgamma(breaks[i], shape=gamma,
      rate=alpha)−pgamma(breaks[i−1], shape=gamma, rate=alpha)
    expected[i−1] ← expected[i−1]*length
  }
  parameters ← list(shape=gamma, rate=alpha)
  returnlist ← list(expected=expected, parameters=parameters, df=df)
}
else if(data.list$dist ≡ "pareto"){
  length ← length(data.list$observed)
  if(is.na(data.list$param1) || is.na(data.list$param2)){
    alpha ← data.list$observed[1]
    beta ← 1/((1/length) * sum(log(data.list$observed/alpha)))
    log.alpha ← log(alpha)
    log.beta ← log(beta)
    paretoLoglik ← function(params, negative=TRUE){
      lglk ← sum(pareto.density(data.list$observed,
        alpha=exp(params[1]), beta=exp(params[2]), log=TRUE))
      if(negative)
        return(−lglk)
      else
        return(lglk)
    }
    optim.list ← optim(c(log.alpha, log.beta), paretoLoglik)
    if(optim.list$convergence≡0){
      alpha ← exp(optim.list$par[1])
      beta ← exp(optim.list$par[2])
```

```
      }else{
        warning("optim didn't converge in bin.expected − pareto")
      }
    } else {
      alpha ← data.list$param1
      beta ← data.list$param2
    }
    df ← length−1−2
    breaks ← seq(data.list$min.val, data.list$max.val,
      data.list$bin.width)
    breaks[1]=alpha
    for(i in 2:length(breaks)){
      expected[i−1] ← pareto.distr(breaks[i], alpha,
        beta)−pareto.distr(breaks[i−1], alpha, beta)
      expected[i−1] ← expected[i−1]*length
    }
    parameters ← list(alpha=alpha, beta=beta)
    returnlist ← list(expected=expected, parameters=parameters, df=df)
  }
  else if(data.list$dist ≡ "beta"){
    if(is.na(data.list$param1) || is.na(data.list$param2) ||
      is.na(data.list$param3) || is.na(data.list$param4)){
      min.val ← data.list$min.val
      max.val ← data.list$max.val
      a.val ← max.val − min.val
      b.val ← min.val
      transformed.data ← (data.list$observed−b.val)/a.val
      transformed.data ← transformed.data[transformed.data>0]
      transformed.data ← transformed.data[transformed.data≠1]
      mean.transformed.data ← mean(transformed.data)
      var.transformed.data ← var(transformed.data)
      alpha.transformed ←
        mean.transformed.data*((mean.transformed.data*(1−mean.transformed.data)/
var.transformed.data)−1)
      beta.transformed ←
        (1−mean.transformed.data)*((mean.transformed.data*(1−mean.transformed.da
ta)/var.transformed.data)−1)
      log.alpha.transformed ← log(alpha.transformed)
      log.beta.transformed ← log(beta.transformed)
      betaLoglik ← function(params, negative=TRUE){
        lglk ← sum(dbeta(transformed.data, shape1=exp(params[1]),
          shape2=exp(params[2]), log=TRUE))
        if(negative)
          return(−lglk)
        else
          return(lglk)
      }
      optim.list ← optim(c(log.alpha.transformed, log.beta.transformed),
        betaLoglik, method="BFGS")
      if(optim.list$convergence≡0){
        alpha.transformed ← exp(optim.list$par[1])
        beta.transformed ← exp(optim.list$par[2])
      }else{
        warning("optim didn't converge in bin.expected − beta")
      }
    } else {
      alpha.transformed ← data.list$param1
      beta.transformed ← data.list$param2
      min.val ← data.list$param3
      max.val ← data.list$param4
    }
    length ← length(data.list$observed)
    df ← length−1−2
    breaks ← seq(data.list$min.val, data.list$max.val, data.list$bin.width)
    breaks[1] ← min.val
    for(i in 2:length(breaks)){
      if(i≡2){
        expected[i−1] ← beta.distr(x=breaks[i], alpha=alpha.transformed,
          beta=beta.transformed, a=min.val, b=max.val)−0
      }
      else{
        expected[i−1] ← beta.distr(x=breaks[i], alpha=alpha.transformed,
          beta=beta.transformed, a=min.val,
          b=max.val)−beta.distr(x=breaks[i−1], alpha=alpha.transformed,
          beta=beta.transformed, a=min.val, b=max.val)
      }
      expected[i−1] ← expected[i−1]*length
    }
    parameters ← list(shape1=alpha.transformed, shape2=beta.transformed)
    returnlist ← list(expected=expected, parameters=parameters, df=df)
  }
  else if(data.list$dist ≡ "extreme"){
    if(is.na(data.list$param1) || is.na(data.list$param2)){
      beta ← (sqrt(var.data) * sqrt(6)) / pi
      alpha ← mean.data−0.5772*beta
      log.alpha ← log(alpha)
      log.beta ← log(beta)
      extremeLoglik ← function(params, negative=TRUE){
        lglk ← sum(dgev(data, loc=exp(params[1]), scale=exp(params[2]),
          shape=0, log=TRUE))
        if(negative)
          return(−lglk)
        else
          return(lglk)
      }
      optim.list ← optim(c(log.alpha, log.beta), extremeLoglik)
      if(optim.list$convergence≡0){
        alpha ← exp(optim.list$par[1])
        beta ← exp(optim.list$par[2])
      }else{
        warning("optim didn't converge in bin.expected − extreme")
      }
    } else {
      alpha ← data.list$param1
      beta ← data.list$param2
    }
    length ← length(data.list$observed)
    df ← length−1−2
    breaks ← seq(data.list$min.val, data.list$max.val, data.list$bin.width)
    for(i in 2:length(breaks)){
      expected[i−1] ← pgumbel(breaks[i], loc=alpha,
        scale=beta)−pgumbel(breaks[i−1], loc=alpha, scale=beta)
      expected[i−1] ← expected[i−1]*length
    }
    parameters ← list(alpha=alpha, beta=beta)
    returnlist ← list(expected=expected, parameters=parameters, df=df)
  }
  else if(data.list$dist ≡ "normal"){
    if(is.na(data.list$param1) || is.na(data.list$param2)){
      norm.mean ← mean(data.list$observed)
      norm.sd ← sd(data.list$observed)
      result.list ← fitdistr(data.list$observed, "normal",
        start=list(mean=norm.mean, sd=norm.sd))
      norm.mean ← result.list$estimate[[1]]
      norm.sd ← result.list$estimate[[2]]
    } else {
      norm.mean ← data.list$param1
      norm.sd ← data.list$param2
    }
    length ← length(data.list$observed)
    df ← length−1−2
    breaks ← seq(data.list$min.val, data.list$max.val, data.list$bin.width)
    for(i in 2:length(breaks)){
      expected[i−1] ← pnorm(breaks[i], mean=norm.mean,
```

Figure 55: Lambda Discrepancy Measure - Code 3

```
        sd=norm.sd)-pnorm(breaks[i-1], mean=norm.mean, sd=norm.sd)
      expected[i-1] <- expected[i-1]*length
    }
    parameters <- list(mean=norm.mean, sd=norm.sd)
    returnlist <- list(expected=expected, parameters=parameters, df=df)
  }
  else{
    stop("\nERROR: invalid distribution specified - choose one of the
      following: \n1. exponential\n 2. weibull\n 3. lognormal\n 4. gamma\n 5.
      pareto\n 6. beta\n 7. extreme\n 8. normal")
  }
  return(returnlist)
}

# Combine bins containing less than a predefined number of observations.

combine.bins <- function(expected, actual, bin.threshold){
  length <- length(expected)
  flag <- TRUE
  # Less than 3 bins not allowed for lambda squared
  if(length < 3){
    stop("\nERROR: invalid number of bins - number of bins < 3")
  }
  i <- 1
  while(i<length){
    if(expected[i] < bin.threshold){
      # Combine last bin with second last bin
      if(i == length){
        expected[i-1] <- expected[i-1] + expected[i]
        actual[i-1] <- actual[i-1] + actual[i]
        length = length-1
        expected <- expected[1:length]
        actual <- actual[1:length]
      }
      # Combine bin in body of array with next bin - move array one
      # position to left
      else{
        flag <- FALSE
        # Add value of next bin to this bin
        expected[i] <- expected[i] + expected[i+1]
        actual[i] <- actual[i] + actual[i+1]
        # Move array elements up
        # If second last element
        if(i == length-1){
          length = length-1
          expected <- expected[1:length]
          actual <- actual[1:length]
        }
        # Any other element
        else{
          for(j in (i+1):(length-1)){
            expected[j] <- expected[j+1]
            actual[j] <- actual[j+1]
          }
          length = length-1
          expected <- expected[1:length]
          actual <- actual[1:length]
        }
      }
    }
    if (flag == TRUE){
      i <- i+1
    }
    flag <- TRUE
  }
  return(list(expected=expected, actual=actual))
}
```

Figure 56: Lambda Discrepancy Measure - Code 4

```
# Calculate the lambda squared statistic for a dataset

lambda.squared <- function(observed, distribution, bin.threshold=5,
  bin.method="scott", param1=NA, param2=NA, param3=NA, param4=NA){
  observed <- sort(observed)
  length <- length(observed)
  # Bin actual data
  actual.list <- bin.actual(observed, bin.method)
  actual <- actual.list$actual
  min.val <- actual.list$min.val
  max.val <- actual.list$max.val
  # Bin expected data
  data.list <- list(distribution=distribution, observed=observed,
    min.val=actual.list$min.val, max.val=actual.list$max.val,
    bin.width=actual.list$bin.width, param1=param1, param2=param2,
    param3=param3, param4=param4)
  expected.list <- bin.expected(data.list)
  parameterlist <- expected.list$parameters
  expected <- expected.list$expected
  # Housecleaning - without this we usually run out of memory
  rm(observed, actual.list, expected.list)
  # Combine bins with values < bin.threshold, bin.threshold must be integer > 0
  if(length(actual) ≠ length(expected)){
    stop("\nERROR: unequal actual and expected arrays")
  }
  if(bin.threshold < 1){
    stop("\nERROR: bin.threshold must be integer >= 1")
  }
  combined.list <- combine.bins(expected, actual, bin.threshold)
  expected <- combined.list$expected
  actual <- combined.list$actual
  num.bins <- length(actual)
  # Calculate lambda square
  # Less than 3 bins not allowed for lambda squared
  if(num.bins < 4){
    lambda <- NA
    variance <- NA
    warning("\nWARNING: invalid number of bins (number of bins < 3) - lambda
      set to NA")
  }
  else{
    difference <- numeric()
    for (i in 1:num.bins) {
      if(expected[i] == 0) {
        stop("\nERROR: illegal expected value (expected value = 0)")
      }
    }
    # lambda square
    num.params <- length(parameterlist)
    df <- num.bins-num.params-1
    difference <- actual-expected
    k <- sum(difference/expected)
    x2 <- sum((difference)^2/expected)
    lambda <- (x2-k-df)/(length-1)
    # variance of lambda square
    T <- sum((difference^3-2*difference*expected+5/2*difference^2+3/2*(differenc
e+expected))/expected^2)
    variance <- (2*df+4*length*lambda+4*length*lambda^2+4*T)/length^2
    # 90% confidence intervals for lambda square
    upper.limit <- lambda + 1.64*sqrt(variance)
    lower.limit <- lambda - 1.64*sqrt(variance)
    confidence.interval <- list(upper.limit=upper.limit, lower.limit=lower.limit
)
  }
  # Return list
  return(list(lambda=lambda, chi.square=x2, k=k, variance=variance,
```

```
  confidence.interval=confidence.interval, parameters=parameterlist,
    num.observations=length, num.bins=num.bins, df=df, min.val=min.val,
    max.val=max.val))
}
```

Figure 57: Lambda Discrepancy Measure - Code 5

```
# Anderson Darling Goodness of Fit Routines
# Lourens Walters - Cape Town University
# 04/10/2003

library(MASS)
library(evd)

pareto.andtest ← function(x, alpha=NA, beta=NA)
{
   x ← sort(x)
   length ← length(x)
   if(is.na(alpha) || is.na(beta)){
   alpha ← x[1]
   beta ← 1/((1/length) * sum(log(x/alpha)))
   }
   x ← beta * log(x/alpha)
   x ← x[2:length]
   log.alpha ← log(alpha)
   log.beta ← log(beta)
   paretoLoglik ← function(params, negative=TRUE){
     lglk ← sum(pareto.density(x, alpha=exp(params[1]), beta=exp(params[2]),
     log=TRUE))
     if(negative)
        return(-lglk)
     else
        return(lglk)
   }
   optim.list ← optim(c(log.alpha, log.beta), paretoLoglik)
   if(optim.list$convergence=0){
      alpha ← exp(optim.list$par[1])
      beta ← exp(optim.list$par[2])
   }else{
      warning("optim didn't converge in bin.expected - pareto")
   }
   result.list ← exponential.case2(x)
   return(list(sig.level = result.list$sig.level, and.stat.val = result.list$and
.stat.val, par.param1 = alpha, par.param2 = beta, data.num = length))
}

gamma.case3 ← function(x, shape=NA, scale=NA) {

   # Cannot use fitdistr for gamma, maximum likelihood optimisation passes
   # negative values for parameters to optimisation function.
   # Do maximum likelihood optimisation ourselves by using "optim"
   if(is.na(shape) || is.na(scale)){
      shape ← mean(x)^2/var(x)
      scale ← var(x)/mean(x)
      log.shape ← log(shape)
      log.scale ← log(scale)
      gammaLoglik ← function(params, negative=TRUE){
        lglk ← sum(dgamma(x, shape=exp(params[1]), scale=exp(params[2]),
        log=TRUE))
        if(negative)
           return(-lglk)
        else
           return(lglk)
      }
      optim.list ← optim(c(log.shape, log.scale), gammaLoglik, method="BFGS")
      if(optim.list$convergence=0){
         shape ← exp(optim.list$par[1])
         scale ← exp(optim.list$par[2])
      }else{
         warning("optim didn't converge in bin.expected - gamma")
      }
   }
   rate ← 1/scale
   x ← sort(x)
   z ← pgamma(x, shape=shape, rate=rate)
   n ← length(z)
   astat ← A2(z)
   sig ← gamma.significance.5percent(astat, shape)
   return(list(sig.level = sig, and.stat.val = astat, gam.param1 = shape, gam.pa
ram2 = scale, gam.param3 = rate, data.num = n))
}

normal.case3 ← function(x, mean.val=NA, sd.val=NA){
   x ← sort(x)
   length.val ← length(x)
   if(is.na(mean.val) || is.na(sd.val)){
      mean.val ← mean(x)
      sd.val ← sd(x)
      result.list ← fitdistr(x, "normal", start=list(mean=mean.val, sd=sd.val))

      mean.val ← result.list$estimate[[1]]
      sd.val ← result.list$estimate[[2]]
   }
   z.vals ← pnorm(x, mean=mean.val, sd=sd.val)
   astat ← A2(z.vals)
   astat ← astat*(1 + 0.75/length.val + 2.25/length.val^2)
   sig ← normal.significance(astat)
   return(list(sig.level = sig, and.stat.val = astat, normal.param1 = mean.val,
no.param2 = sd.val, data.num = length.val))
}

lognormal.case3 ← function(x, zeta=NA, sigma=NA) {
   log.x ← log(x)
   length.val ← length(log.x)
   if(is.na(zeta) || is.na(sigma)){
      zeta ← mean(log.x)
      sigma ← sd(log.x)
      result.list ← fitdistr(x, "lognormal", start=list(meanlog=zeta, sdlog=sigma
))
      zeta ← result.list$estimate[[1]]
      sigma ← result.list$estimate[[2]]
   }
   w ← (log.x-zeta)/sigma
   z.vals ← pnorm(w, 0, 1)
   z.vals ← sort(z.vals)
   astat ← A2(z.vals)
   astat ← astat*(1+(0.75/length(z.vals))+(2.25/((length(z.vals))^2)))
   sig ← lognormal.significance(astat)
   return(list(sig.level = sig, and.stat.val = astat, log.param1 = zeta, log.par
am2 = sigma, log.param3 = 0, data.num = length.val))
}

lognormal.case3.censored ← function(x, n) {
   x ← log(x)
   x ← sort(x)
   norm.ppoints ← ppoints(n)
   r ← length(x)
   m.i ← qnorm(norm.ppoints)
   m.i.subset ← m.i[1:r]
   m.mean ← sum(m.i.subset)/r
   b.i ← 1/r - ((m.mean*(m.i.subset - m.mean))/sum((m.i.subset-m.mean)^2))
   c.i ← ((m.i.subset - m.mean)/sum((m.i.subset-m.mean)^2))
   mu ← sum((b.i * x))
   sigma ← sum((c.i * x))
   w ← (x-mu)/sigma
   z ← pnorm(w, 0, 1)
   t ← max(x)
   p ← max(z)
   z ← sort(z)
   astat ← A2.censored(z, n)
   sig ← lognormal.significance.censored.5percent(astat, p)
```

Figure 58: Anderson Darling Test - Code 1

```
   return(list(and.stat.val = astat, sig.level = sig, log.param1 = mu, log.param
2 = sigma, log.param3 = 0, data.num.censored = r, data.num.uncensored = n, p.val
 = p))
}

weibull.case3 ← function(x, gamma=NA, alpha=NA) {
   x ← -log(x)
   x ← sort(x)
   length ← length(x)
   if(is.na(gamma) || is.na(alpha)){
      parameters ← fgev(x, shape=0, std.err=FALSE)
      psi ← fitted(parameters)[1]
      theta ← fitted(parameters)[2]
      gamma ← 1/theta
      alpha ← exp(-psi)
   } else{
      psi ← -log(alpha)
      theta ← 1/gamma
   }
   z ← exp(-exp(-((x-psi)/theta)))
   astat ← A2(z)
   astat ← astat*(1+(0.2/sqrt(length)))
   sig ← extreme.significance(astat)
   return(list(sig.level = sig, and.stat.val = astat, wei.param1 = gamma, wei.pa
ram2 = alpha, wei.param3 = 0, data.num = length))
}

extreme.case3 ← function(x, alpha=NA, beta=NA) {
   x ← sort(x)
   length ← length(x)
   mean.data ← mean(x)
   var.data ← var(x)
   if(is.na(alpha) || is.na(beta)){
      beta ← (sqrt(var.data) * sqrt(6)) / pi
      alpha ← mean.data-0.5772*beta
      log.alpha ← log(alpha)
      log.beta ← log(beta)
      extremeLoglik ← function(params, negative=TRUE){
        lglk ← sum(dgev(data, loc=exp(params[1]), scale=exp(params[2]), shape=0
, log=TRUE))
        if(negative)
           return(-lglk)
        else
           return(lglk)
      }
      optim.list ← optim(c(log.alpha, log.beta), extremeLoglik)
      if(optim.list$convergence≡0){
        alpha ← exp(optim.list$par[1])
        beta ← exp(optim.list$par[2])
      }else{
        warning("optim didn't converge in bin.expected - extreme")
      }
   }
   z ← pgumbel(x, loc=alpha, scale=beta)
   astat ← A2(z)
   astat ← astat*(1+(0.2/sqrt(length)))
   sig ← extreme.significance(astat)
   return(list(sig.level=sig, and.stat.val=astat, alpha=alpha, beta=beta, data.n
um=length))
}

exponential.case2 ← function(x, rate=NA) {
   x ← sort(x)
   if(is.na(rate)){
      exp.mean ← mean(x)
      exp.rate ← 1/exp.mean
      log.rate ← log(exp.rate)
      expLoglik ← function(params, negative=TRUE){
        lglk ← sum(dexp(x, rate=exp(params[1]), log=TRUE))
        if(negative)
           return(-lglk)
        else
           return(lglk)
      }
      optim.list ← optim(c(log.rate), expLoglik, method="BFGS")
      if(optim.list$convergence≡0){
         exp.rate ← exp(optim.list$par[1])
      }else{
         warning("optim didn't converge in bin.expected - exponential")
      }
   } else{
      exp.rate ← rate
   }
   exp.param ← 1/exp.rate
   length ← length(x)
   z ← 1-exp(-x/exp.param)
   astat ← A2(z)
   astat ← astat*(1.0+0.6/length)
   sig ← exponential.significance(astat)
   return(list(sig.level = sig, and.stat.val = astat, exp.param1 = exp.param, ex
p.param2 = 0, data.num = length(x)))
}

exponential.case2.censored ← function(censored, n) {
   censored ← sort(censored)
   t ← max(censored)
   r ← length(censored)
   beta ← (sum(censored) + ((n-r) * t))/r
   z ← 1-exp(-censored/beta)
   astat ← A2.censored(z, n)
   p ← max(z)
   sig ← exponential.significance.censored.5percent(astat, p)
   return(list(and.stat.val = astat, sig.level = sig, exp.param1 = beta, exp.par
am2 = 0, data.num = r, p.val = p))
}

normal.significance ← function(astat){
   levels ← c(0.341, 0.47, 0.561, 0.631, 0.752, 0.873, 1.035)
   sig ← max(c(0.5, 0.25, 0.15, 0.1, 0.05, 0.025, 0.01)[astat ≤ levels])
   if(is.na(sig) || sig ≡ Inf || sig ≡ -Inf)
      sig ← 0
   sig * 100
}

lognormal.significance ← function(astat){
   levels ← c(0.341, 0.470, 0.561, 0.631, 0.752, 0.873, 1.035)
   sig ← max(c(0.5, 0.25, 0.15, 0.1, 0.05, 0.025, 0.01)[astat ≤ levels])
   if(is.na(sig) || sig ≡ Inf || sig ≡ -Inf)
      sig ← 0
   sig * 100
}

extreme.significance ← function(astat){
   levels ← c(0.474, 0.637, 0.757, 0.877, 1.038)
   sig ← max(c(0.25, 0.1, 0.05, 0.025, 0.01)[astat ≤ levels])
   if(is.na(sig) || sig ≡ Inf || sig ≡ -Inf)
      sig ← 0
   sig * 100
}

exponential.significance.censored.5percent ← function(astat, p){
   p.val ← c(0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 1.0)
   levels ← c(0.274, 0.501, 0.746, 1.003, 1.149, 1.232, 1.321)
   p ← max(p.val[p ≤ p.val])
```

Figure 59: Anderson Darling Test - Code 2

```
   if(is.na(p) || p ≡ Inf || p ≡ -Inf){
      return.val ← 0
   }
   else{
      and.dar ← levels[p ≡ p.val]
      if(and.dar ≥ astat){
         return.val ← 5.0
      }
      else{
         return.val ← 0.0
      }
   }
   return.val
}

lognormal.significance.censored.5percent ← function(astat, p){
   p.val ← c(0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 1.0)
   levels ← c(0.133, 0.250, 0.359, 0.528, 0.623, 0.686, 0.752)
   p ← max(p.val[p ≤ p.val])
   if(is.na(p) || p ≡ Inf || p ≡ -Inf){
      return.val ← 0
   }
   else{
      and.dar ← levels[p ≡ p.val]
      if(and.dar ≥ astat){
         return.val ← 5.0
      }
      else{
         return.val ← 0.0
      }
   }
   return.val
}

gamma.significance.5percent ← function(astat, shape){
   shape.param ← c(1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 1000000000)
   levels ← c(0.786, 0.768, 0.762, 0.759, 0.758, 0.757, 0.755, 0.754, 0.754, 0.
754, 0.753, 0.752)
   shape  ← max(shape.param[shape ≤ shape.param])
   if(is.na(shape) || shape ≡ Inf || shape ≡ -Inf){
      return.val ← 0
   }
   else{
      and.dar  ← levels[shape ≡ shape.param]
      if(and.dar ≥ astat){
         return.val ← 5.0
      }
      else{
         return.val ← 0.0
      }
   }
   return.val
}

exponential.significance ← function(astat){
   levels ← c(0.736, 0.816, 0.916, 1.062, 1.321, 1.591, 1.959)
   sig ← max(c(0.25, 0.2, 0.15, 0.1, 0.05, 0.025, 0.01)[astat ≤ levels])
   if(is.na(sig) || sig ≡ Inf || sig ≡ -Inf)
      sig ← 0
   sig * 100
}

A2 ← function(z){
   n ← length(z)
   i ← seq(z)
   -n - (1/n) * sum((2 * i - 1) * log(z) + (2 * n + 1 - 2 * i) * log(1 - z))
}
```

Figure 60: Anderson Darling Test - Code 3

```
A2.censored ← function(z, n){
   r ← length(z)
   z.max ← max(z)
   i ← seq(z)
   -(1/n) * sum((2 * i - 1) * (log(z) - log(1-z))) - 2 * sum(log(1-z)) - (1/n) *
 ((r - n)^2 * log(1 - z.max) - r^2 * log(z.max) + n^2 * z.max)
}

DStat ← function(x) {
   z ← ZStatisticExp(x)
   dplus ← DPlus(z)
   dmin ← DMin(z)
   max(dplus, dmin)
}

VStat ← function(x) {
   z ← ZStatisticExp(x)
   dplus ← DPlus(z)
   dmin ← DMin(z)
   dplus+dmin
}

UStat ← function(x){
   z ← ZStatisticExp(x)
   w ← WStat(x)
   w^2-(n*((mean(z)-0.5)^2))
}

WStat ← function(x) {
   z ← ZStatisticExp(x)
   IValues ← 1:length(z)
   Temp ← z-((2*IValues-1)/(2*length(x)))
   TempSquare ← Temp^2
   sum(TempSquare)+(1/(12*length(x)))
}

DPlus ← function(z) {
   IValues ← 1:length(z)
   max(IValues/length(z)-z)
}

DMin ← function(z) {
   IValues ← 1:length(z)
   max(z-(IValues-1)/length(z))
}
```

Appendix D

Concepts

Traffic processes can be characterised as being: **bursty**, **short-range dependent**, **long-range dependent** and **self-similar**. The probability distribution of a traffic process can be **heavy-tailed**. Definitions of these as well as supporting concepts are given in this section.

D.1 Stationarity

A stochastic process $\{X(t)\}$ is stationary in the strict sense if for $n \geq 1$, its n^{th} order joint distribution satisfies the condition:

$$F(\mathbf{x}; \mathbf{t}) = F(\mathbf{x}; \mathbf{t} + \ell) \tag{15}$$

for all vectors $\mathbf{x} \in \Re^n$ and $\mathbf{t} \in T^n$, and all scalars ℓ such that $t_i + \ell \in T$. The notation $\mathbf{t} + \ell$ implies that the scalar ℓ is added to all components of vector $\mathbf{t}$.

A `strict sense stationary stochastic process` $\{X(t)\}$ has the property that $E[X(t)]$ is the same for all $t \in T$. We say that $E[X(t)]$ is independent of t. This independence can be seen when applying the definition of strict sense stationarity to the first order distribution of $\{X(t)\}$:

$$F(x; t) = F(x; t + \ell) \quad or \quad F_{X(t)} = F_{X_{t+\ell}} \quad for \quad all \quad \ell.$$

The autocorrelation function of a stationary process depends only on the time difference, and is therefore a one-dimensional function written as $\gamma(\ell)$.

Imposing strict stationarity on a process is often too restrictive. A weaker form of stationarity called second-order stationarity (also called wide sense or covariance stationarity) is often used.

D.2 Second-Order Stationarity

A `second-order stationary stochastic process` is defined as a stochastic process $X(t)$ for which:

1. $E[X(t)]$ is independent of t,
2. $\gamma(t_1, t_2) = \gamma(0, t_2 - t_1) = \gamma(\ell), \quad t_2 \geq t_1 \geq 0,$
3. $\gamma(0) = E[X^2(t)] < \infty$ (finite second moment).

It can be seen that a second-order stationary process has an autocorrelation function γ that depends on the time difference ℓ only, and is therefore a one-dimensional function written as $\gamma(\ell)$. More specifically, the autocorrelation function is:

$$\gamma(\ell) = \frac{E[(X(t) - \mu)(X(t + \ell) - \mu)]}{E[(X(t) - \mu)^2]}$$

for $\mu = E[X(t)]$. The value ℓ is referred to as the lag of the autocorrelation function.

D.3 Short Range Dependence

A traffic process $\{X_n\}$ is said to be short-range dependent if non-trivial autocorrelations exist for small values of ℓ and the autocorrelation function .

A `short-range dependent process` is a second-order stationary stochastic process for which the autocorrelation function $\gamma(\ell)$ decays exponentially for increasing values of ℓ:

$$\gamma(\ell) \sim r^{\ell} \quad as \;\; \ell \to \infty, \tag{16}$$

for some $0 < r < 1$.

A summable autocorrelation function is the result of (16):

$$\sum_{k=-\infty}^{\infty} \gamma(\ell) < \infty, \tag{17}$$

D.4 Long Range Dependence

A process is said to long-range dependent if non-trivial autocorrelations exist for large values of ℓ. The decision whether ℓ is large or small is subjective and depends on the situation.

A `long-range dependent process` is a second-order stationary stochastic process for which the autocorrelation function $\gamma(\ell)$ decays slowly i.e. hyperbolically instead of exponentially for increasing values of ℓ. A non-summable autocorrelation function:

$$\sum_{k=-\infty}^{\infty} \gamma(\ell) = \infty, \tag{18}$$

implies a slowly decaying autocorrelation function and hence long-range dependence.

Another characteristic of a long-range dependent process is that its autocorrelation function takes on the form:

$$\gamma(\ell) \sim c\ell^{-\beta} \quad as \;\; \ell \to \infty, \tag{19}$$

where $0 < \beta < 1$ and $c > 0$ is a constant.

It can be seen that (19) implies (18).

Appendix E

Analytic Traffic Models

As mentioned before traffic models are mathematical models based on probability theory which capture the behaviour of network traffic. Stochastic theory deals with phenomena that have probabilistic behaviour which changes with time i.e. the distribution of probabilities for an event is not always the same at different moments in time. The behaviour of network traffic changes with time e.g. there is more traffic during midday on weekdays than there is during midnight on weekends. Stochastic theory is therefore used to model network traffic.

A stochastic process $X(t)$is a family of random variables $\{x(t), t \in T\}$ indexed by a parameter t over some index set T. The index set is usually the time dimension, and $x(t)$ therefore a function of time. A stochastic process is therefore a random variable that is a function of time.

Network traffic consist of the arrival of packets at a point in a network. The traffic can be characterised by two stochastic processes, one describing the time of arrival and the other the size of packets. The time of arrival of packets are represented by the inter-arrival time process denoted by A_n. $\{A_n, n \geq 1\}$ is a non-negative stochastic process where A_n is the length of the time interval separating the n^{th} arrival from the previous one. The size of packets are represented by the packet size process denoted by $\{S_n\}$. $\{S_n, n \geq 1\}$ is also a non-negative stochastic process and is independent of A_n.

Some stochastic traffic models have been selected and are defined in this section. The models have been selected for their utility and analytic tractability. They have been categorised into classes according to their definitions.

E.1 Renewal Traffic Processes

A renewal process is a counting process $\{N(t), t > 0\}$ with interoccurence times $X_1, X_2, \ldots, X_n$ being random variables that are independently and identically distributed. The distribution of the variables is allowed to be general.

E.1.1 Poisson Process

A Poisson process is a renewal process with exponentially distributed interoccurence times and with rate parameter λ,

$$F(t) = 1 - e^{\lambda t}, \qquad t \geq 0$$

The counting process associated with the Poisson process has a Poisson distribution with mean λt,

$$P\{N(t) = k\} = e^{-\lambda t}\frac{(\lambda t)^k}{k!}, \qquad k = 0, 1, \ldots$$

The Poisson process is memoryless. The time between two arrivals has the same distribution, irrelevant of the position between the two arrivals from which the probability of the next arrival is measured. The memoryless property makes the Poisson process mathematically tractable. It has been used successfully for nearly a century to analyse voice-telephony. The super-positioning of independent Poisson processes result in a new Poisson process with rate equal to the sum of the component rates.

E.1.2 Bernoulli Process

The Bernoulli process is the discrete time analog of the Poisson process. The interoccurence times are geometrically distributed with parameter p indicating the probability of an arrival in any time slot,

$$P\{A_n = j\} = p(1-p)^j$$

The counting process associated with the Bernoulli process has a binomial distribution,

$$P\{N_k = n\} = \left(\begin{array}{c} k \\ n \end{array} \right) p^n (1-p)^{k-n}, \qquad 0 \leq n \leq k$$

E.1.3 Phase-type Renewal Processes

Inter-arrival times are modelled as the time to absorption in a continuous-time Markov process with discrete, finite state space. State 0 is absorbing, all other states are transient and absorption is guaranteed in a finite time. The process is started with the same initial distribution π each time.

E.2 Markov and Semi-Markov Models

Unlike renewal traffic processes, Markov models introduce dependence into $\{A_n\}$. A simple continuous time Markov chain model M is defined as follows: inter-arrival times are determined by the times spent in states of M. Each jump from state i to state j $(\forall i, j)$ signals an arrival. Inter-arrival times are therefore exponentially distributed.

A discrete time Markov chain model M defines inter-arrival times to be the number of slots separating successive arrivals. The different states of M represent different inter-arrival times. The probability of a j slot separation following an i slot separation is therefore given by the one step probability $p_{i,j}$.

Compound Markov traffic models are created by defining $\{B_n\}$. Workload $\{W_n\}$ can also be modelled by Markov models.

E.2.1 Semi-Markov Models

A semi-Markov process is obtained by allowing the time between state transitions to follow an arbitrary probability distribution.

The Markovian Arrival Process (MAP) is a broad class of semi-Markov processes that is analytically tractable. The inter-arrival distribution is that of the phase-type renewal process. The process is however not restarted with the same initial distribution π. The restart distribution depends on the previous state from which absorption was reached. The super-positioning of independent MAP traffic processes results in a process whose state space is the cross-product of the component state spaces.

E.2.2 Markov Modulated Models

A Markov modulated or doubly stochastic process process uses an auxiliary Markov process in which the current state of the Markov process controls the probability distribution of the traffic. The auxiliary process is usually a continuous time Markov chain which keeps the model analytically tractable. Semi-Markov processes can however be used i.e. holding times are not restricted to the exponential distribution.

E.2.3 Markov Modulated Poisson Process

The modulated process is a Poisson process with rate λ_k. As the modulating process changes from state s_k to state s_j so does the rate from λ_k to λ_j. The Markov modulated Poisson process (MMPP) allows the modelling of variable rate sources while keeping the analytical solution of related queueing performance tractable. Parameters of the MMPP can be estimated easily from empirical data. Super-positioning of traffic streams can take advantage of the Poisson process' property of the aggregate rate being equal to the sum of the rates of component processes.

E.2.4 Markov Modulated Bernoulli Process

The modulated process is a Bernoulli process with p_k indicating the probability of an arrival in any time slot. As with the MMPP p_k changes as the state s_j changes.

E.3 Fluid Traffic Models

Fluid models characterise traffic by means of flow rate as a volume (e.g. bits per second) as opposed to packet counts. Typical fluid models make use of the alternating state renewal process. These models can be analysed as Markov-modulated constant rate traffic. The superposition of identical independent alternating state renewal processes has a binomial distribution.

E.4 Autoregressive-Type Traffic Models

Autoregressive models define a variate in terms of previous variates.

E.4.1 Linear Autoregressive (AR) Processes

An $AR(p)$ process of order p is defined as,

$$X_n = a_0 + \sum_{r=1}^{p} a_r X_{n-r} + \epsilon_n, \qquad n > 0,$$

where $(X_{-p+1}, \ldots, X_0)$ is a prescribed random vector, the a_r, $0 \leq r \leq p$, are real constants, and the ϵ_n are zero-mean uncorrelated random variables (white noise) called residuals which are independent of X_{n-r}. Residuals are used to get a closer fit to empirical data.

E.4.2 Moving Average (MA) Processes

An MA(q) process of order q is defined as:

$$X_n = \sum_{r=0}^{q} b_r \epsilon_{n-r}, \quad n > 0,$$

where the $b_r, 0 \leq r \leq q$ are real constants and the ϵ_n are zero-mean uncorrelated random variables. MA models are autocorrelated time series, since successive variates are defined in terms of common subsets of $\epsilon n - r$.

E.4.3 Autoregressive Moving Average (ARMA) Processes

An ARMA(p,q) process of order (p,q) is defined as:

$$X_n = a_0 + \sum_{r=1}^{p} a_r X_{n-r} + \sum_{r=0}^{q} b_r \epsilon_{n-r},$$

it can be seen that this process is a combination of AR and MA models.

E.4.4 Autoregressive Integrated Moving Average (ARMA) processes

ARIMA processes are obtained by replacing X_n in:

$$X_n = a_0 + \sum_{r=1}^{p} a_r X_{n-r} + \sum_{r=0}^{q} b_r \epsilon_{n-r},$$

by the d^{th} differences of the process $\{X_n\}$. An ARMA model is therefore "integrated" (summed) to yield an ARIMA model. ARIMA processes are stochastic models whose parameter estimation, model identification and selection are well understood. Both ARMA and ARIMA models have autocorrelation functions that decay geometrically and are therefore not suited for modelling long-range dependent traffic.

Bibliography

[Ada97] A. Adas. Traffic Models in Broadband Networks. *IEEE Communications Magazine*, pages 82–89, July 1997.

[AEV97] J. Aracil, R. Edell, and P. Varaiya. An empirical Internet traffic study. In *35th Allerton Conference*, Urbana Il, October 1997.

[AW95] M. Arlitt and C. L. Williamson. A synthetic workload model for internet mosaic traffic. In *Summer Computer Simulation Conference*, pages 24–26, Ottawa, July 1995.

[AW96] M. F. Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Measurement and Modeling of Computer Systems*, pages 126–137, 1996.

[AZN98] R.G. Addie, M. Zukerman, and T.D. Neame. Broadband Traffic Modeling: Simple Solutions to Hard Problems. *IEEE Communications Magazine*, pages 88–95, August 1998.

[BC98a] P. Barford and M. Crovella. An Architecture for a WWW Workload Generator. 1998.

[BC98b] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Performance 1998/ACM SIGMETRICS 1998*, pages 151–160, 1998.

[Ben86] Jon Bentley. *Programming pearls*. ACM Press, 1986.

[BLFea97] T. Berners-Lee, R. Fielding, and H. Frystyk et. al. RFC 2068: Hypertext Transfer Protocol (HTTP/1.1), January 1997. Internet Standard.

[BLFea99] T. Berners-Lee, R. Fielding, and H. Frystyk et. al. RFC 2616: Hypertext Transfer Protocol (HTTP/1.1), June 1999. Internet Standard.

[BLFF95] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol (HTTP/1.0), October 1995. Informational Track.

[CB96] M. E. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. In *Proceedings ACM SIGMETRICS 1996: The ACM*

International Conference on Measurement and Modeling of Computer Systems, pages 151–160, Philadelphia, Pennsylvania, May 1996. Also, in Performance evaluation review, May 1996, 24(1):160-169.

[CBC95] C. Cunha, A. Bestavros, and M. E. Crovella. Characteristics of World Wide Web Client-based Traces. Technical Report BUCS-TR-1995-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[CDJ91] R. Caceres, P. Danzig, and S. Jamin. Characteristics of Widearea TCP/IP Conversations. In *Proceedings of ACM SIGCOMM '91.* ACM PRESS, November 1991.

[CKO99] E. Cohen, H. Kaplan, and J. Oldham. Policies for managing tcp connections under persistent http, 1999.

[CL99] H. Choi and J. Limb. A Behavioural Model of Web Traffic. In *7th International Conference on Network Protocols (ICNP 1999)*, Toronto, Canada, October 1999.

[Com] Combined Research Effort - See Internet Site. The network simulator - ns. Internet - http://www.isi.edu/nsnam/ns/. Last accessed: 9/10/2001.

[CP95] L. D. Catledge and J. E. Pitkow. Characterizing browsing strategies in the World-Wide Web. *Computer Networks and ISDN Systems*, 27(6):1065–1073, 1995.

[CT99] M.E. Crovella and M.S. Taqqu. Estimating the Heavy Tail Index from Scaling Properties. *Methodology and Computing in Applied Probability*, 1(1), 1999.

[CTB98] M. Crovella, M. Taqqu, and A. Bestavros. Heavy-Tailed Probability Distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*, pages 3–26. Chapman & Hall, 1998.

[DdHR00] H. Drees, L. de Haan, and S. Resnick. How to make a Hill plot. *Annals of Statistics*, 28(25):254–274, 2000.

[Den96] S. Deng. Empirical Model of WWW Document Arrivals at Access Link. In *IEEE International Conference on Communication*, June 1996.

[DJC+92] P. Danzig, S. Jamin, R. Caceres, D. Mitzel, and D. Estrin. An Empirical Workload Model for Driving Wide-Area TCP/IP Network Simulations. *Internetworking: Research and Experience*, 3(1):1–26, March 1992.

[ENW96] A. Erramilli, O. Narayan, and W. Willinger. Experimental Queueing Analysis with Long-Range Dependent Packet Traffic. *IEEE/ACM Transactions on Networking*, pages 209–223, April 1996.

[Fel98] A. Feldmann. Characteristics of TCP Connection Arrivals. Technical report, AT&T Labs Research, 180 Park Av., A175, Florham Park, NJ 07932, December 1998.

[Fel00] Anja Feldmann. BLT: Bi-layer tracing of HTTP and TCP/IP. *WWW9 / Computer Networks*, 33(1-6):321–335, 2000.

[FM94] V.S. Frost and B. Melamed. Traffic Modeling for Telecommunications Networks. *IEEE Communications Magazine*, pages 70–81, March 1994.

[HOT97] J. Heidemann, K. Obraczka, and J. Touch. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Transactions on Networking*, 5(5):616–630, October 1997.

[KMM00] R. Kalden, I. Meirick, and M. Meyer. Wireless Internet Access Based on GPRS. *IEEE Personal Communications*, 7(2):8–18, April 2000.

[LCW97] D. Lam, D. C. Cox, and J. Widom. Teletraffic modeling for personal communications services. *IEEE Communications Magazine*, pages 79–87, February 1997.

[LTWW93] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the Self-Similar Nature of Ethernet Traffic. In D. P. Sidhu, editor, *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[LWDW97] M. T. Lucas, D. E. Wrege, Bert J. Dempsey, and A. C. Weaver. Statistical characterization of wide-area IP traffic. In *Proceedings of Sixth International Conference on Computer Communications and Networks (IC3N'97)*, 1997.

[Mah97] B. A. Mah. An Empirical Model of HTTP Network Traffic. In *Proc. InfoComm '97*, April 1997.

[Mil92] D.L. Mills. RFC 1305: Network Time Protocol, March 1992. Draft Standard.

[Mil94] D.L. Mills. Precision synchronization of computer network clocks. *ACM Computer Communication Review*, 24(2):28–43, April 1994.

[MJ93] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *USENIX Winter*, pages 259–270, 1993.

[Moo86] D. S. Moore. Tests of Chi-squared Type. *STATISTICS: textbooks and monographs*, 68:63–95, 1986.

[Nor99] Norbert Vicari and Stefan Köhler. Measuring Internet User Traffic Behavior Dependent on Access Speed. Technical Report TR 238, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany, 1999.

[Pax93] V. Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Connections: Extended Report. Technical Report LBL-34086, Lawrence Berkeley Laboratory and EECS Division, 1 Cyclotron Road, Berkeley, CA 94720, June 1993.

[Pax94] V. Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking*, pages 316–336, August 1994.

[Pax97] V. Paxson. Why We Don't Know How to Simulate the Internet. In *Proceedings of 1997 Winter Simulation Conference*, Atlanta GA, U.S.A., December 1997.

[PF95] V. Paxson and S. Floyd. Wide-area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, pages 226–244, June 1995.

[PJ90] S. P. Pederson and M. E. Johnson. Estimating Model Discrepancy. *TECHNOMETRICS*, 32:305–314, 1990.

[Pos82] J. Postel. RFC 821: Simple Mail Transfer Protocol (SMTP), August 1982. Internet Standard.

[PR85] J. Postel and J. Reynolds. RFC 959: File Transfer Protocol (FTP), October 1985. Internet Standard.

[Res97] S. Resnick. Heavy Tail Modeling and Teletraffic Data. *Annals of Statistics*, 25:1805–1869, 1997.

[RLGPC+99] A. Reyes-Lecuona, E. Gonzalez-Parada, E. Casilari, J. C. Casasola, and A. Diaz-Estrella. A page-oriented WWW traffic model for wireless system simulations. In D. Smith and P. Key, editors, *16th International Teletraffic Congress(ITC16)*, volume 3.b, pages 1271–1280, Edinburgh (UK), June 1999.

[SCJO01] F. Donelson Smith, Felix Hernandez Campos, Kevin Jeffay, and David Ott. What TCP/IP Protocol Headers Can Tell Us About the Web. In *ACM SIGMETRICS*, pages 245–256, Cambridge, MA, June 2001.

[Sco79] D. W. Scott. On Optimal and Data-Based Histograms. *Biometrica*, 66:605–610, 1979.

[Sco92] D. W. Scott. *Multivariate Density Estimation*. John Wiley and Sons, Inc, 1992.

[SLT01] D. Staehle, K. Leibnitz, and K. Tsipotis. QoS of internet access with GPRS. In *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 57–64, Rome, Italy, 2001. ACM Press.

[SLTG99] D. Staehle, K. Leibnitz, and P. Tran-Gia. Source Traffic Modeling of Wireless Applications. Technical Report TR 261, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany, 1999.

[Sta00] W. Stallings. *Data and Computer Communications*. Prentice Hall, Sixth edition, 2000.

[Ste86] M. A. Stephens. Tests Based on EDF Statistics. *STATISTICS: textbooks and monographs*, 68:97–185, 1986.

[TWS97] M. S. Taqqu, W. Willinger, and R. Sherman. Proof of a Fundamental Result in Self-Similar Traffic Modeling. *ACMCCR: Computer Communication Review*, 27, 1997.

[UC 02] UC Berkeley, LBL, USC/ISI and Xerox PARC. *The ns Manual*, April 2002.

[UD96] R. Ulrich and W. Dulz. OSSCAR — object oriented simulation of slotted communication architectures. *Lecture Notes in Computer Science*, 1067:858–??, 1996.

[WPT98] W. Willinger, V. Paxson, and M. S. Taqqu. Self-similarity and Heavy Tails: Structural Modeling of Network Traffic. In R. J. Adler, R. E. Feldman, and M. S. Taqqu, editors, *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*. Birkhauser, Boston, 1998.

[WTSW95] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson. Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. *Proceedings ACM SIGCOMM 1995*, pages 100–113, 1995.