GNUTELLA NETWORK TRAFFIC MEASUREMENTS AND CHARACTERISTICS

DRAGOS ILIE

APRIL 2006
DEPARTMENT OF TELECOMMUNICATION SYSTEMS,
SCHOOL OF ENGINEERING,
BLEKINGE INSTITUTE OF TECHNOLOGY

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2006:05
ISSN 1650-2140
ISBN 91-7295-084-6

Published 2006
Printed by Kaserntryckeriet AB
Karlskrona 2006
Sweden

This publication was typeset using LATEX.

To my parents

"The only true wisdom is in knowing you know nothing"
- Socrates (469–399 BC) -

Abstract

Wide availability of computing resources at the edge of the network has lead to the appearance of new services based on peer-to-peer architectures. In a peer-to-peer network nodes have the capability to act both as client and server. They self-organize and cooperate with each other to perform more efficiently operations related to peer discovery, content search and content distribution.

The main goal of this thesis is to obtain a better understanding of the network traffic generated by Gnutella peers. Gnutella is a well-known, heavily decentralized file-sharing peer-to-peer network. It is based on open protocol specifications for peer signaling, which enable detailed measurements and analysis down to individual messages. File transfers are performed using HTTP.

An 11-days long Gnutella link-layer packet trace collected at BTH is systematically decoded and analyzed. Analysis results include various traffic characteristics and statistical models. The emphasis for the characteristics has been on accuracy and detail, while for the traffic models the emphasis has been on analytical tractability and ease of simulation. To the author's best knowledge this is the first work on Gnutella that presents statistics down to message level.

The results show that incoming requests to open a session follow a Poisson distribution. Incoming messages of mixed types can be described by a compound Poisson distribution. Mixture distribution models for message transfer rates include a heavy-tailed component.

Acknowledgments

This thesis is a compressed summary of my research achievements from the past three years. Many of these achievements would not have been possible without the direct or indirect support from a number of people.

First and foremost, I would like to express my gratitude and appreciation to Docent Adrian Popescu from Blekinge Institute of Technology (BTH). Already while I was a M. Sc. student, he encouraged me to pursue graduate studies. His tenacity, enthusiasm and belief in my capacity to get the job done were key elements in finalizing this thesis.

My colleague David Erman was of invaluable help in my research. He was always there to challenge new ideas and ask difficult questions. Discussions with him about research topics resulted often in fresh, new insights.

Special thanks go to Doru Constantinescu for settling the scientific arguments between David and I. In addition to that, he was always there for me when I was facing research challenges with no solution in sight. He always encouraged me to find new ways to move forward.

I would like to thank Prof. Arne Nilsson for accepting me as a Ph. D student at the department and for being my secondary advisor.

I have benefited from several interesting discussions with Dr. Markus Fiedler. For this, I thank him very much.

My fellow graduate students Lennart Isaksson, Stefan Chevul, Patrik Arlos and Henric Johnson deserve acknowledgments for encouragement and many interesting discussions.

I would like to thank our head of department, Civ. Eng. Anders Nelsson, who has dealt admirably with practical issues related to my studies, such as literature, equipment and conference travel.

Dr. Parag Pruthi, CEO of Niksun Inc., has my gratitude for helping me with the transition from being a software engineer in his company to becoming a Ph. D. student at BTH.

Much of my early scientific skills were trained by Dr. T. V. Kurien, now with Microsoft Corp. He often reminded me that if I do not start graduate studies before the age of 30, I probably never will. Looking back, I know he was right.

I would like to express my deep gratitude to my parents who were always there for me. Without their love, help and encouragement I would not have made it this far.

Last, but not least, I express my warmest and deepest appreciation to Zohra for her love and support and for bearing with me through the last year.

Dragos Ilie
Karlskrona, March 2006

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter begins with a short account of the Internet history, from early beginning to present day. The aim is to expose the factors and chronological events that led to the emerge of Peer-to-Peer (P2P) file-sharing technology as dominant form of Internet traffic in terms of volume.

An overview of the motivation for this thesis is the subject of Section 1.2. That is followed by a presentation of related work in Section 1.3. The main contributions of this thesis work are described in Section 1.4. An outline for the contents of this document is provided in Section 1.5.

1.1 Evolution of P2P

Two characterizing aspects of the Internet are its tremendous growth and heterogeneity. A search for "Internet growth" on Google reveals that Internet has grown from four nodes in 1969 to over 350 million hosts in 2005 [Zak97, Zak05]. This figure is in fact a low estimate of the true number of users since it relies only on hosts with registered IP addresses. Many more hosts use private IP addresses behind Network Address Translators (NATs). When taking those nodes into account the Internet size estimates go beyond 900 million hosts [IWS05]. Furthermore, measurements indicate that Internet growth in terms of network traffic has been roughly doubling every year since 1997 [Odl03] and there seems to be no indication that the growth will slow down any time soon. Increasing availability of low-priced telecommunication and networking equipment on the Far East market, in particular in China and India, provides further evidence towards that claim.

Internet heterogeneity comes in terms of hardware, software and underlaying algorithms. There is an abundance of Internet nodes using x86-compatible processors (e.g., Intel Pentium and AMD Athlon), tightly followed by nodes using other processor types, such as PowerPC (e.g., Apple Macintosh), Motorola 68xxx, DEC Alpha, and Transmeta Crusoe. The Microsoft Windows Operating Systems (OSs) (NT, 98, and XP) are widespread since they are typically bundled with many of the Personal Computers (PCs) sold on the market. However, alternatives such as Linux, Mac OS X and FreeBSD are quickly gaining ground and high-performance server nodes often use some form of UNIX OS such as SUN OS or OpenBSD with a long history of scalability, security and reliability. The TCP/IP suite of protocols has become the de-facto standard for Internet communication and is available for all hardware and software described above. Both the Internet Protocol (IP) and the Transport Control Protocol (TCP) are still evolving. IPv6 is aimed to resolve shortcomings in IPv4 related to address space exhaustion, autoconfiguration and multicast [Com00]. There is much ongoing research into adapting TCP algorithms to new environments (e.g., wireless environments) as well as research on improving the existing TCP congestion control. Currently, there are three main TCP implementations (Tahoe, Reno and Vegas) and a number of experimental implementations (HighSpeed TCP, BIC TCP and Fast TCP).

Internet growth and heterogeneity were fueled mainly by two driving forces: wide availability of integrated circuits using Very Large-Scale Integration (VLSI) technology coupled with the appearance of “killer applications”. A killer application, generally speaking, refers to software or hardware which is so useful that it is adopted immediately by a large number of users that push the technology to its limits.

E-mail, which appeared in the early 1970s, is very likely the first Internet killer application. Although e-mail by itself did not push the technology to its limits, it attracted a large number of users (most of them from academia) to the Internet by making it easy to quickly exchange information. In the beginning of 1980 distributed newsgroups in the form of Usenet were competing with e-mail and File Transfer Protocol (FTP) for being the killer application.

Another decade passed until November 1990, when Tim Berners-Lee and Robert Cailiau published their formal specification for the World Wide Web [BLC90]. Two months later Tim Berners-Lee released the first web browser and web server [Wik05]. It took another two years for the Mosaic web browser to become available, followed one year later by Netscape Navigator. The modern user interface implemented by these browsers were very appealing to users and contributed significantly to the emergence of the new killer application: the Web. The emergence of the Web coincided with a sharp increase in the popularity of the *client-server* architecture paradigm in the early 1980s [ES05a]. This coincidence was most likely not entirely arbitrary, but correlated with the fact that the Web was based on the client-server paradigm. Client-server application architectures segregate nodes into servers and clients. Server nodes are used solely for providing services useful for client nodes.

By the late 1990s millions of commercial users had obtained access to the Internet. Many of them had access to PCs with multimedia capabilities, in particular the ability to play music Compact Discs (CDs). A process called "ripping the CD" was used to copy the audio tracks from the CD to the PC's harddisk allowing the users to mix songs into custom playlists. The size of a full music CD is about 600 MB, which was too large to be practical with the harddisk sizes at that time. Fortunately for users, a new efficient audio compression format known as MP3[1] [Ins06] was already developed in 1994. Using MP3 compression, data reduction by a factor of 10 was possible. Many users exchanged MP3 files by e-mail or posted them to newsgroups and personal web pages. However, finding specific songs was not easy. One way to search for them was through web search engines such as Lycos and Altavista. There was also the possibility to request specific songs on Internet Relay Chat (IRC) channels. A dramatic improvement in finding songs occurred in May 1999 when Shawn Fenning released a software called Napster, which is regarded as the first widespread P2P application for file-sharing.

In the P2P approach, nodes are no longer segregated into servers and clients. Instead, each host acts as a server when other nodes request services from it, but also as a client when it demands service for itself from other nodes. Thus, in the P2P architecture paradigm nodes are in a continuous state of flux, acting simultaneously as clients and servers.

Nodes using Napster software reported to the Napster central server the MP3 files they made available for download. Users interested in finding a particular song queried the server and obtained a list with all hosts that had matching files. Users could then download the file directly from any host on the list. This approach to handle content search and distribution via a central server is called *centralized directory*. The ease of use and the efficiency of looking up files in the centralized directory transformed Napster into a success almost over night.

The success of Napster became quickly a source of serious concern for major record companies, who rapidly filed a lawsuit against Napster on grounds of copyright infringement. The lawsuit made Napster immensely popular, attracting additional millions of users. However, Napster could not withstand the pressure of the lawsuit and in July 2001 they were forced to shut down the central server [Enc05]. Without the central server the client nodes could no longer search for files. Thus, the fragility of a centralized directory system became clear.

Following Napster, a large number of P2P networks appeared: Gnutella, eDonkey,

[1]The official name is ISO-MPEG Audio Layer-3.

FastTrack (used by Kazaa and Grokster), DirectConnect, BitTorrent and others. Many of them implemented a P2P architecture with distributed control to avoid a fate similar to that of Napster's. In addition to that, the new systems were able to share any type of computer files, not only MP3 songs. Advances in video compression such as DivX [Div06] and XviD [LMMR06] allowed users to share entire movies, further exacerbating the conflict with media copyright holders. This conflict has also contributed to new ideas on how to improve data and user anonymity in P2P networks.

Currently, research is performed on the next generation of P2P systems that promise to offer better scalability and more efficient content search and distribution through the use of Distributed Hash Tables (DHTs).

The legal controversy surrounding P2P networking (or P2P networks) is an artifact created by news reporting media, which erroneously use the term P2P as a synonym for illegal file-sharing. It is the actual act of sharing *copyrighted* files that is not legal, not the method by which this is done.

Actually, the P2P networking paradigm has been in use since the early days of the Advanced Research Projects Agency Network (ARPANET). Legacy applications such as the Domain Name System (DNS) and the Simple Message Transfer Protocol (SMTP) still use it today. Similarly, the concept of file-sharing has been incorporated in many applications with a long history: the UNIX command `rcp`, FTP, Network File System (NFS), File and Printer Sharing for Microsoft Networks, etc.

1.2 Motivation

Research performed at Blekinge Institute of Technology (BTH) has a tradition of investigating new Internet services and applications, in particular their behavior in terms of network traffic. Papers discussing measurements and modeling were published in many areas, reflecting the state of art for communication technology at the time: Ethernet, ATM, TCP, Web with emphasis on HyperText Transfer Protocol (HTTP) and security, just to name a few.

Several publications from 2002–2003 such as [SGG02, GDS$^+$03, KBB$^+$03] indicated the emergence of P2P as the next killer application. These papers sparked an interest in the researchers at BTH to take a closer look at P2P services, and are to a large extent responsible for planting the seeds for this thesis work.

In 2004, the company CacheLogic presented data from measurements performed over a 6-month period with various Tier 1 Internet Service Providers (ISPs) and cable providers across the world [Par04]. The data provided strong evidence that P2P traffic is the largest data contributor in ISP networks, with BitTorrent being the leading application. Newer measurements performed in 2005 [Par05] show that P2P is still generating as much as 60% of the traffic on the Internet. However, the P2P application landscape has changed, with eDonkey and Gnutella generating traffic volumes comparable to BitTorrent. It is likely that this is a reaction to the legal actions pursued by copyright holders against major BitTorrent sites (e.g., Supernova), which forced users to switch over to technologies that provide better anonymity. Other measurement reports conclude that P2P will continue to grow in the future [KBB$^+$04, KBFC04, AG04].

Several companies have realized the potential in P2P networking and are currently developing or offering products and services incorporating P2P technologies. The general idea behind new services and applications is to push content distribution away from central servers towards end-nodes, in an effort to diminish the traffic load on ISP links and servers. Probably the best example is Skype [Com06], an application which offers free phone calls over the Internet. Another example is BBC's iMP [BBC06] that provides recorded TV and radio programs. Other examples include using BitTorrent to off-load web servers in content distribution. This approach is used by software companies, in particular by the video game industry, to distribute demos of their products. It is also the modern way employed by companies such as Novell and RedHat to distribute the latest Linux releases.

Other ideas rely on the fact that users use only a fraction of the harddisk storage and computing capacity of their PCs. For example, users joining OceanStore [Ber06a] share a portion of their local storage in exchange for economic compensation. OceanStore service providers trade capacity among themselves transparently to the users. The project SETI@Home [Ber06b] harnesses the computing power of idle PCs to analyze radio data from outer space, hoping to discover signs of extraterrestrial intelligence. SETI@Home is a form of Internet computing. Although, SETI@home is a client-server systems, it exhibits several properties attributed to P2P systems [MKL$^+$03].

All these developments indicate that P2P is here to stay and that it will be the dominant form of digital content distribution, in terms of network traffic volume, in the foreseeable future.

The main goal of this thesis is directed towards gaining a better understanding of Gnutella traffic patterns. There are three main reasons why the Gnutella network was selected for this thesis work. First, Gnutella was and still is a well-documented open protocol. This means no reverse engineering or other type of guess-work is necessary to decode protocol messages, thus allowing the research to focus on measurements and analysis. Second, at the time when the thesis work started the Gnutella protocol was among the most popular protocols. Last, but not least, the Gnutella Development Forum and mailing list was and still is an invaluable source of information, especially if one needs to follow the bleeding edge development of Gnutella.

A problem related to real-time content distribution such as video and audio is the lack of Quality of Service (QoS) guarantees. There are two main proposals to adopt

QoS guarantees into the Internet architecture: Integrated Services (IntServ) and Differentiated Services (DiffServ)[Wan00]. Unfortunately, none of them has been widely adopted at present, mostly due to failure to convince ISPs to invest into these technologies. A number of suggestions [And01, LM04, SSBK04] have appeared that aim at providing QoS by using overlay networks built on end-nodes without requiring the involvement of ISPs. These overlays are very likely to become viable alternatives to IntServ and DiffServ.

An important goal for future research at BTH is to use the results and models presented here to implement and validate a QoS overlay on top of a Gnutella-like P2P network.

1.3 Related Work

There is a large body of literature that describes measurements and modeling of Gnutella traffic. Only a subset is presented in this section, in particular the papers that influenced the work presented in this thesis.

Perhaps the oldest and most cited paper is [AH00], which looks into the social aspects of the Gnutella network. The authors instrumented a Gnutella client to log protocol events. The main contribution of the paper was to show that only a few peers contribute with hosting or adding new content to the Gnutella network, whereas the majority of nodes would retrieve content without sharing any. The authors used the term *free-riding* to describe this behavior and showed that it was just another form of the tragedy of the commons phenomenon described more than three decades earlier [Har68]. The conclusion of the paper was that the common belief that the Gnutella network is more resilient to shutdowns due to distributed control does not hold very well when only few nodes host the majority of content.

A dooms-day prediction was made by [Rit01], which through mathematical analysis, argued that due to its architectural design, in particular the volume of signaling traffic, the Gnutella network will not be able to scale to more than a few hundred users. Enhancements in message caching, flow control and dynamic hierarchical routing implemented by major Gnutella vendors have rendered most of the conclusions in [Rit01] obsolete.

In [SGG01, SGG02] the authors created *crawlers* for Napster and Gnutella networks. A crawler is a special purpose software agent, which discovers and records the network topology through an automated, iterative process. The authors used information from the crawlers to measure properties of individual peers (e.g., bandwidth and latency). The data from their measurements indicated that both Gnutella and Napster exhibit highly heterogeneous properties (e.g., connectivity, speed, shared data). This is contrary to the design assumptions used when those systems were built. Another important finding, which supports the conclusions in [AH00], is that users are typically unwilling to cooperate with each other, few of them acting as servers and the remaining majority acting as clients.

A different approach was taken in [SW02]. The authors performed non-intrusive flow measurements at a large ISP instead of using a crawler. Their goal was to analyze FastTrack (a protocol used by Kazaa and Grokster), Gnutella and DirectConnect networks. Flows belonging to any of these networks were identified by well-known port numbers. The major findings in the paper are that all three networks showed increases in the traffic volume across consecutive months, skewed distributions for traffic volume, connectivity and average bandwidth, few hosts with a long uptime, and uniformity in terms of number of P2P nodes from individual network address prefixes.

Measurements from a 1 Gbit/s link in the France Telecom IP backbone [AG04] network revealed that almost 80 % of traffic on the link in question was produced by P2P applications. Further, the authors showed that flows were partitioned into "mice" — short flows, mostly due to signaling, and "elephants" — long flows due to data transfers.

The P2P traffic identification in [SW02, AG04] assumes that applications use well-known ports. This assumption rarely holds nowadays, when P2P applications use dynamic ports in order to camouflage themselves. Karagiannis et al. [KBB$^+$03, KBFC04, KBB$^+$04] use increasingly better heuristics to detect P2P traffic. Their measurement results showed that, if anything, P2P traffic was not declining in volume. Further, they showed that P2P traffic is using predominantly dynamic ports. Applications that currently use or will use encrypted connections would make the P2P flow identification task even harder, if not impossible.

1.4 Main Contributions

The focus of this work has been on workload characterization for a single Gnutella peer. In that context, the following contributions were made:

- Highly detailed statistical models and characteristics of Gnutella traffic crossing an ultrapeer.
- Statistical methods to fit distributions to very large number of samples.
- Gnutella traffic decoder and accompanying TCP flow reassembly.
- Flexible software library for P2P traffic decoding, based on `tcptrace`.

- Measurement infrastructure for P2P traffic based on `tcpdump`.

1.5 Thesis Outline

This licentiate thesis is organized as follows. The current chapter, Chapter 1, summarizes major developments in the Internet until the current day, focusing on the concepts of P2P networking and file-sharing. It also sheds some light upon the motivation behind this work and presents concisely the main contributions.

Chapter 2 provides a short overview of the state-of-the-art of P2P technology. In particular, it introduces the reader to various terms and definitions used throughout the thesis. Further, it describes different types of P2P topologies, the process of peer discovery, several approaches to content management and it concludes by listing important challenges lying ahead for P2P systems.

Chapter 3 offers a detailed presentation of the Gnutella protocol.

Different type of measurements and their advantages and disadvantages are the topic of Chapter 4. The measurement infrastructure developed at BTH is described as well.

Chapter 5 reviews basic statistics and notation. Then, various methods to fit statistical models to data and test the quality of the fit are presented.

Chapter 6 describes characteristics of the Gnutella traffic inferred from a 11-days long measurement trace. In addition, it presents statistical models for session, messages and byte rates.

Finally, Chapter 7 summarizes this thesis by presenting the conclusions of this work along with avenues for future work.

Chapter 2

Peer-to-Peer Networks

The primary goal of this chapter is to introduce P2P terminology and definitions that will be used throughout the rest of the thesis.

In the field literature the term P2P is often used together with the words “computing” or “networking”. Computing denotes the act of performing a sequence of operations on some data by means of a computer. In distributed computing (e.g., P2P computing) the sequence of operations is often distributed to a set of computers to achieve some goal more efficiently. Networking is the exchange of data by interconnected computers. Distributed computing typically relies on networking to perform its task. This work focuses exclusively on networking.

In general, technical P2P literature tends to define P2P networking as a fuzzy relationship among interconnected nodes that alternate between the roles of client and server. The exact characteristics defining a P2P network seem quite elusive at first, since authors tend to focus on the characteristics relevant to their work. Some attempts to settle this situation can be found in [Sch01, MKL$^+$03, ES05a]. In general these definitions establish that a P2P network architecture implies a distributed network with decentralized control and dynamic membership, in which participants share resources (e.g., storage space, processing power, bandwidth) in order to achieve some common goal. It should be noted that in this case the term bandwidth is used to denote available channel or link capacity.

Section 2.1 provides basic definitions, which should ensure that more advanced concepts can be described without any accidental confusion. The remaining sections focus mostly on P2P taxonomy based on various characteristics of P2P networks.

The three main characteristics that define a P2P network are the overlay topology, the bootstrap and peer discovery process as well as the functions of content management. Section 2.2 considers various forms of overlay topology and compares the advantages and disadvantage of centralized architectures versus decentralized architectures as well as the advantages and disadvantages of structured versus unstructured topologies. Section 2.3 summarizes the concepts of bootstrap and peer discovery. Content management is the topic of Section 2.4 and covers content insertion, distribution and search. In addition, various transport protocols are considered also. Finally, the last section discusses some of the main challenges faced by P2P systems.

2.1 Definitions

The networking field relies heavily on the term "host" or "node". Although trivial, the following definition is adopted here for completeness.

Definition 2.1. A *host* or a *node* denotes a network entity (often a PC or workstation) capable of communication. For the purpose of this work both names are equivalent. □

Sometimes, the term host implies a node that offers some sort of service or stores valuable information. Here, a host is no different from a node.

Definition 2.2. A *network* is a set of nodes interconnected in some way. Networking is a term describing the exchange of data or services in a network. □

It is important to differentiate between logical and physical networks. In a physical network (e.g., an IP network) the node interconnections can be either optical links, a shared medium (e.g., Ethernet), cable modems, serial cables or other type of physical information carrier. In an overlay network, node interconnections are *virtual*. Directly connected nodes in an overlay may be several hops away from each other in the physical network, as shown in Figure 2.1.

Use of overlays is not restricted to P2P networks. Technologies such as Multiprotocol Label Switching (MPLS), IntServ and DiffServ span some form of overlay network as well.

Definition 2.3. The way in which nodes of a network are interconnected defines the network *topology*. □

Unless otherwise specified, the thesis will consider only overlay topologies. Furthermore, the physical network supporting the overlay will be assumed to be an IPv4 network.

Definition 2.4. A *link* is a direct node interconnection (one hop) in the overlay. Nodes may be several hops away in the physical network. □

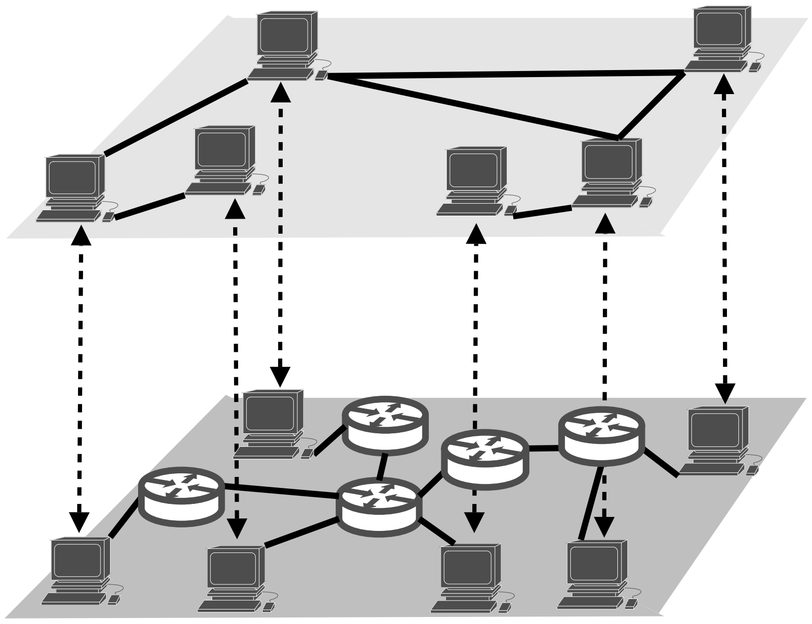


Figure 2.1: Physical network at the bottom, overlay network at the top

Definition 2.5. A *path* is a contiguous sequence of links connecting two nodes. The node at each end of the path is called an *end-node* or *end-point*. Nodes on the path between the end-nodes are called *intermediate nodes*. For directly connected nodes, there are no intermediate nodes and the path between the end-nodes is equivalent to the link connecting them. □

Definition 2.6. A network in which every node is directly connected to the rest of the nodes in the network has a *full-mesh* topology and it is called a fully-connected network. □

Full mesh networks are very expensive to implement since the number of links increases[1] as $O(n^2)$. Therefore, in practice different (less expensive) types of topologies are employed together with some form of routing algorithm, which enables finding the path between two arbitrary nodes.

When describing the direction of the data flow between end-nodes, the *reverse path* is the opposite direction of the path. For example, given two end-nodes **A** and **C** on a

[1]The number of links required for a full-mesh network consisting of n nodes is $n(n-1)/2$.

1-hop bi-directional path crossing node **B**, where the *forward path* is $\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C}$, the *reverse path* becomes $\mathbf{C} \rightarrow \mathbf{B} \rightarrow \mathbf{A}$. On the forward path the node **A** is typically referred to as the source node, node **B** is the transit node and node **C** is called the destination node. The opposite relation (i.e., **C** is the source node and **A** is the destination node) governs the reverse path. The adjective *forward* in forward path is optional and it is used only if there is a risk of misunderstanding.

On path $\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C}$, node **A** is upstream from node **B** and **C**, node **B** is downstream from node **A** and upstream from **C**, and finally node **C** is downstream from both node **A** and node **B**. Upstream and downstream notions are interchanged for the reverse path.

Definition 2.7. Network *connectivity* is a measure that defines on average the number of paths connecting any two nodes in the network. □

Network connectivity is a measure associated with routing resilience, which is the ability to adapt to changes in the network topology. It is also important in terms of content search and distribution.

Definition 2.8. *Content*, refers to any type of digital media (e.g., text, video, audio) not necessarily as files explicitly, but also as streams of bytes or packetized in the form of datagrams (e.g., Internet radio).

There are three main operations related to content, which can be performed in a network:

Insertion	Insertion is the function of adding content to the resource pool of a network (e.g., making files available for download).
Distribution	Distribution is the function of retrieving content from the resource pool of a network (e.g., downloading available files).
Control	Control is the function of managing the resource pool of a network (e.g., admission control, resource discovery, content removal).

□

Definition 2.9. A *client-server* network is a network composed of two type of nodes: *server* nodes dedicated to offering services or content and *client* nodes that use services or consume content provided by server nodes. □

A network is a form of *system*. Systems discussed in the thesis are based on specific *architectures*, which are the high-level blueprints for the system. Thus, it is important to differentiate between an *architecture*, which is a logical concept (e.g., a protocol or software specification) and a system, which is a working implementation of the architecture in question.

In a pure client-server architecture all three content operations (insertion, distribution, control) are performed by the server.

Definition 2.10. A *P2P* network is the antithesis of a client-server network, in the sense that nodes can arbitrary switch between the roles of server and client, or perform them simultaneously. P2P networks rely on all nodes to share various resources: bandwidth, processing power, memory, storage space, etc. □

Since nodes implement both server and client functionality, the view is that they are all equal to each other in that respect. This led to adoption of the term *peer* as synonym for a P2P node. Another name in use is *servent*, which is a combination of the words *serv*er and cli*ent*.

In a pure P2P architecture all content operations (insertion, distribution, control) are distributed equally among the peers. However, this is not entirely true for some current P2P architectures, as it will be discussed in Chapter 2.4.

The majority of applications in IP-based networks use the Berkeley socket Application Programming Interface (API) for remote communications. The following terminology is required for many concepts discussed in the remaining part of the thesis.

Definition 2.11. A socket pair defines the end-points in the Berkeley socket API. It consists of a 5-tuple: <source IP address, source port, destination IP address, destination port, protocol>. A socket is just one end of the communication channel consisting of a 3-tuple: <IP address, port number, protocol>. □

The term *listening socket* denotes an end-point that is waiting for a connection attempt. The *listening port* denotes the port of a listening socket. According to the Internet Assigned Numbers Authority (IANA), there are three categories of ports:

Designation	Port number range
Well-known	0–1023
Registered	1024–49151
Dynamic	49152–65535

Table 2.1: Port division by IANA

Well-known ports are used by OS services. Traditionally, application use registered ports for listening sockets and dynamic ports for the end-point that initiates a connection. P2P applications prefer to use dynamic ports exclusively to camouflage their existence, mostly to escape censorship through network traffic filtering.

2.2 Overlay Topology

This section is concerned with the organization of nodes in a P2P network. The network architecture has a large impact on the resulting topology.

2.2.1 Centralized vs. Distributed Architectures

Client-server systems tend to use a star topology, with the server at the center and the clients at the periphery. This is the typical example of a *centralized architecture*. The first P2P system in wide-use, Napster, used a centralized architecture with a star topology. In particular, Napster used a server where a list of active peers was kept together with the file index shared by each peer. This allowed queries to be responded to very quickly. However, the entire network depended on the central server. When that server was shutdown the network vanished. The reason why Napster is regarded as a P2P system in spite of its topology, is that peers could download data from each other without involving the server.

The main lesson taught by Napster's demise is that a resilient network must be organized in such a way that failure of a few nodes should not noticeably affect the other peers. This lesson pervaded the P2P community, since the next generation of P2P networks were fully *decentralized* networks, Gnutella being the best example. Fully decentralized networks use some form of *mesh* topology[2], in which nodes have the same function in the network (i.e., they are all equal). It turned out that, when these networks reached a critical mass of peers, they could not scale due to massive volumes of signaling messages among peers [SR02, CRB03].

A solution to this problem was presented almost three decades earlier in [KK77]. The paper suggests that large scalability can be achieved by introducing hierarchy into the network. In a hierarchical network, nodes from a lower hierarchical level cluster around nodes from higher hierarchical level. This principle has been successfully implemented in DNS, interdomain routing (i.e., Classless Inter-Domain Routing (CIDR)) and mobile ad-hoc networks [XHG03]. P2P networks have also adopted a form of hierarchy, such as *ultrapeers* in Gnutella and *superpeers* in Kazaa. Hierarchical P2P networks are called *hybrids* [ES05b]. Ideally, hierarchical networks have a tree topology. However, since cycles (path loops) occur in practice, it is more likely that real hierarchical networks have some form of mesh topology. It should be noted that [KK77] proposed a *hierarchical routing* scheme, which means that routing occurs at several levels of aggregation. This scheme did not require a hierarchical topology. For hierarchical P2P networks, the cost of maintaining a hierarchical topology is rather small

[2]In a mesh topology there are at least two nodes with two or more paths between them.

since an overlay topology deals entirely with virtual connections. Therefore, the hierarchy is implemented in this case either through routing, topology or a combination of both.

2.2.2 Structured vs. Unstructured Architectures

Another form to differentiate among P2P networks is to partition them into *structured* and *unstructured* systems.

In an unstructured system, peers are free to connect to any other peer in the overlay. One problem with unstructured systems is that resource discovery is not very efficient since it has to be implemented through some form of flooding or other forms of intensive communication, as described in Chapter 2.4.

Attempts on how to improve the problem of content location have lead to the idea of forcing peer connections to follow a specific structure that allows a distributed index for content to be used. Structured systems rely on Distributed Hash Tables (DHTs).

A *hash record* is a data type consisting of a *key* with the corresponding *value*. Hash keys are typically numerical values or strings, while the hash values are indexes in an array and are therefore usually numerical. The array itself is called the *hash table*. A *hash function* operates on a given hash key producing a corresponding unique hash value (index) that points to a location in the hash table. This is the location where the data is stored or the best place to start a search for it. The quality of a hash table and its hash function is related to the probability of *collisions*. A collision happens when two or more keys point to the same location in the hash table. This problem can be solved by enlarging and rearranging the hash table, but will in general lead to severe performance degradation.

DHTs are hash tables spread across many nodes in a network. Each node participating in a DHT is responsible for a subset of the DHT keys. When the hash function is given a key it will produce a hash value that identifies the node responsible for that particular key.

To facilitate more efficient key lookups, DHT algorithms force overlay peers to form topologies that follow a certain structure, a so called DHT geometry. For example, Chord [SMK$^+$01] overlays use a circle structure while Content-Addressable Networks (CANs) [RFH$^+$01] uses a N-dimensional hypercube. Some DHT algorithms include the notion of peer proximity, thus forcing new nodes arriving to the overlay to accept low proximity neighboring nodes. This is a way to enforce locality, which means that nodes are close in the real (IP) network.

Implementing hierarchical topology in a structured system is possible, but not very common. To the author's best knowledge, Coral [FM03] and Brocade [ZDHJ02] are the only structured overlays to have currently implemented a hierarchical topology.

2.3 Bootstrap and Peer Discovery

A new node wishing to join an P2P overlay for the first time has no a-priori knowledge about other peers in the overlay. To join the overlay successfully, a node must first have the listening socket of at least another peer, preferably a well-connected one. This is called the *bootstrap problem* and solutions to it are called *bootstrap algorithms*. A bootstrap algorithm must be efficient in the sense that it must find well-connected neighbors in order to avoid small disconnected islands of peers.

Historically, one could obtain a peer's listening socket either from web pages or from selected IRC channels. This was however not very practical and therefore the next step was to include a list of available peers with the distributions of new P2P software releases.

However, this approach did not work well either, since listed peers were not available for long. Currently existing bootstrap algorithms rely on some form of central servers as in the case of trackers in BitTorrent and GWebCache servers in Gnutella. This implies that P2P networks take a few steps towards the client-server paradigm. This is seen as a tradeoff between easier peer discovery and increased vulnerability due to centralized control.

Peer discovery is the process of gathering information about the overlay topology (e.g., neighbor peers, nodes joining or leaving the overlay), for the purpose of maintaining or improving connectivity. Peer discovery is usually implemented by having nodes periodically exchange information about known peers. In a hierarchical network, it is usually the top nodes that can offer the best information about available peers.

2.4 Content Management

Content management can be divided roughly into three categories: insertion, distribution and control. It should be noted that, in terms of P2P content control, the focus will be on content search, since the other areas of content control have not been widely implemented.

2.4.1 Content Insertion

Content insertion deals with making content available to peers in the overlay. Depending on the P2P system in question, content insertion may require a number of protocol messages or it can be entirely passive (i.e., no information about the new content is sent to the overlay). The purpose of protocol messages is to inform other peers about the new content or to obtain authorization to publish the content. A node that controls access to some content, including that content's availability, is called the content *owner* [SM05].

2.4.2 Content Distribution

Content distribution is the function of retrieving content from the resource pool. In its simplest form it involves two peers, where one peer downloads the entire content from the other peer, over a direct connection between them. The download time is proportional to the bottleneck capacity (at link layer) between the two hosts and also to the overall error rate on the IP path between them, given that errors trigger retransmissions.

In a more refined form of content distribution, the requester obtains a list with several peers holding the resource. In the Gnutella protocol this is called a *download mesh*, which is the name that will be used in the thesis. The download mesh can be created in the following ways:

i) A dedicated entity keeps track of which peers serve the resource. This is similar to the Napster server, or to the BitTorrent tracker.

ii) Several peers that have the desired resource respond to the resource query.

iii) Other peers that have recently retrieved the resource successfully save temporarily to a local cache the host address or the address of the group of hosts from where the resource was obtained. The information from the cache is used to respond to the resource query.

iv) Peers that have recently served the content save temporarily to a cache the addresses of the hosts that have retrieved successfully the content. The information from this cache can be used to respond to the resource query

v) A combination of the options above.

Given the download mesh, the resource requester may attempt to perform active measurements to obtain statistics on the bottleneck capacity, response time and error rate between the requester and each peer in the list. Then the requester selects the best peer from the download mesh according to some criteria (e.g., highest capacity, low response time, low error rate).

A natural step forward in the evolution of content distribution techniques was *swarming*. With swarming, a peer instructs all or a subset of peers from the download mesh to start uploading unique parts (blocks or byte ranges) of the content. This is in essence a form of capacity aggregation with the benefit of added redundancy. If a peer has a corrupted file, then other peers can supply it.

Since corrupted content is quite common due to incomplete downloads, storage media errors or malicious activity, the issues of error detection and correction become quite important. The emergent solution for content in the form of stored files, is to apply a hash function (e.g., Message-Digest algorithm 5 (MD5) or Secure Hash Algorithm One (SHA-1)) to the file contents and send the hash value to requesters immediately before uploading the file to them. Once the requester has downloaded the entire file it applies the same hash function to its contents. If the computed hash value matches the initial one, then the file was downloaded without errors. Otherwise, the peer can attempt to re-download the file.

Due to the fact that initially, before distribution, a new file is available as a whole only on one host, swarming could not be utilized until one or more peers downloaded the complete file. Hosts that store the entire file are called *seeds* in BitTorrent terminology. Further optimization to swarming was possible by using a technique called Partial File Sharing (PFS). The term PFS appears to have been coined by the Gnutella community, but the technique itself was used earlier as part of the BitTorrent swarming functionality. Peers participating in PFS start downloading the new file from the seed. As soon as they have a complete block of data they make it available for download to other peers without waiting for the entire file to be retrieved. This enables swarming at a very early stage, without waiting for several seeds to appear. The downside of this technique is that if one of the pieces has errors this will not be detected until the hosts downloading have received the entire file. At that point they apply the hash function to the file contents and discover that the hash values don't match. They have no other choice but to restart from scratch since there is no way to tell which block was erroneous.

The solution to the problem of erroneous data blocks led to the idea of computing a hash value for each block of the file. A more sophisticated approach called Tree Hash EXchange format (THEX) [CM03] has been adopted by Gnutella. THEX defines the format for a hierarchical type of hashes that can be applied to each data block of the file and can also be used together to yield a hash for the entire file. Using THEX downloading peers can check each piece and upon error request only that particular piece, thus lowering the link utilization. Another solution is found in [MM03], where the authors introduce path diversity into the network by dividing the stream of pieces in smartly coded substreams. The receiver can rebuild the original file using only a fraction of the substreams.

2.4.3 Transport Protocols

The optimal transport protocol to be used for content distribution has been the source of much debate. The contenders are three well-established protocols: TCP, HTTP and User Datagram Protocol (UDP).

The initial use of TCP was motivated by its reliable nature and low overhead compared to HTTP. However, advances in content distribution techniques moved the focus towards HTTP. HTTP retains the majority of TCP's advantages and has additional benefits such as flexible connection management, bandwidth optimization and error notification [KR01]. The stateless request-response architecture was also appealing for message-based P2P systems. In particular swarming and PFS benefited from the range-request feature in HTTP.

TCP encounters however several problems when it is used in a heterogeneous P2P-environment with many connections. When a node performs swarming it maintains multiple simultaneous TCP connections to hosts from widely different geographic locations. The connections are then likely to have very different Round Trip Times (RTTs). It was shown [LM97] that, in these conditions, the TCP behaves extremely unfair toward the connections with a high RTT. In particular, connections with a high RTT have been shown to get a smaller part of available bandwidth compared to those with lower RTT. Other problems are:

i) TCP assumes that all segment loss is due to congestion and immediately reduces the congestion window to half when an ACK times out. This is counterproductive in P2P, where most losses are due to data corruption and leads to below optimal throughput performance.

ii) A segment lost in a stream can cause several other segments to be discarded. Current implementations for TCP selective acknowledgments do not work well for long fat pipes and lead to non-optimal throughput [Lei03, ELL06].

iii) The TCP three-way handshake adds considerable overhead when used in sporadic communication with many peers.

Some of these problems have prompted a transition to UDP-based transport protocols. This is an unfortunate development, which in the long term may have a negative impact on the Internet as a whole. The TCP congestion avoidance was developed in order to prevent networks from going into congestion collapse. The UDP specification does not define any flow or congestion control mechanisms. When an aggressive UDP stream competes with a TCP stream on available bandwidth, the TCP connection gives up bandwidth in order to avoid congestion. The UDP stream, on the contrary, expands to consume the bandwidth released by the competing TCP stream. If the number of UDP streams grows much larger than the number of TCP streams, then congestion collapse may become a phenomenon much more frequent than it is today.

2.4.4 Content Search

Content control is the management function that controls access to the resource pool (e.g., admission control, resource discovery). P2P file-sharing systems have currently favored distributed content control, where each peer controls access to locally stored files. BitTorrent chooses a hybrid approach, in which a tracker is used to control resource location [Coh03].

Resource discovery (i.e., content search) is the process in which a node, called *requester*, queries its peers about specific content. As far as users are concerned, a query is a case-insensitive text string such as "`adam smith wealth of nations`". The query is matched by any file name that contains all the words in the query string, perhaps ignoring words that appear frequently (e.g., the word "of"). More advanced filtering may be implemented as well. The way the query string is processed in the network is highly dependent on the P2P system in question.

In Napster's centralized architecture the query strings were sent to the central server. A database lookup was performed on the string and the results returned to requester.

This approach is not feasible for a decentralized system. Instead, a process called *query routing* is used. Query routing means that a query is relayed towards the data owner by intermediate nodes.

In an unstructured network this relies on some form of limited flooding. Each peer that receives a query on one connection forwards it on all other peer connections it maintains. The distance in number of hops that the query is allowed to travel, the *search horizon* is controlled by a Time to Live (TTL) variable in the query. This approach is wasting large amounts of bandwidth due to two reasons: *i)* the query reaches many nodes that are unable to answer it, *ii)* for popular content there may be lots of redundant replies sent back. An improvement to limited flooding is called *selective forwarding*. In selective forwarding, the requester sends its query on a very limited number of connections and waits a while to see how many replies it gets. If it does not get enough replies, it repeats the procedure on another small number of connections.

In a structured network the location of the content is defined by the DHT employed. Query routing involves following a path in the DHT geometry (e.g., a straight line path in Cartesian geometry for CAN [RFH$^+$01]). Consequently, DHT-based networks perform resource discovery more efficiently than unstructured networks. However, this is true as far as users search by hash values, so called *exact searches*.

An exact search operates on a hash value, usually created by the host responsible for the content insertion operation. The hash value is obtained by applying a hash function (e.g., SHA-1) to the content to be shared. To search for the content one has to know the hash value. Knowing the file name, the author or the type of content does not help. This is clearly more limiting than the *keyword search* allowed by unstructured networks. On the other hand, the main advantage provided by exact searches is that they locate the exact content as it was created by the original author.

Exact searches have been implemented on top of unstructured networks, although for a slightly different purpose. A keyword search is performed initially in those implementations. The query results sent back to the requester are of the form `<file name, file size, hash value, IP address, port>`, perhaps with additional elements. The file name denotes the name of the file matching the query, the file size is the size in bytes, the hash value is the result of a hash function applied to the file contents and the IP address and port define the listening socket. The requester selects one of the entries to download the corresponding file. This triggers an exact search using the hash value of the file. The way the results are aggregated depends on the P2P software, but it is common that the results are ranked by bandwidth, proximity, RTT etc. From the results, a number of different hosts are selected and each host receives a download request for a range of bytes from the file size. This is in fact the swarming functionality described in Section 2.4.2.

The reverse, implementing keyword searches in a DHT-based network has been implemented as well, working in some cases more efficiently then in unstructured networks [CCR04].

2.5 Challenges

Although P2P networking has seen a tremendous development in recent years, there are still several challenges to be overcome before it becomes a mature technology. The main three challenges are: free-riding, junk content, security.

The free-riding problem was brought to attention in [AH00]. By analyzing Gnutella traffic, the authors discovered significant amount of free riding. In particular they discovered two types of free-riding:

i) A majority of nodes download files without offering any for upload.

ii) Only few nodes share desirable files.

The first type of free riding pushes P2P systems back towards the client-server paradigm. In a fast-growing network, at some point the number of peers sharing no data (i.e., acting as clients) will dwarf the number of sharing peers (i.e., peers that act as servers). Soon enough, the ”servers” will reach the maximum operational capacity (in terms of bandwidth, storage, memory or processing power) and will collapse under load causing the network utility to decrease dramatically. Even if congestion collapse does not happen, peers are likely to experience high packet losses and low throughput due to congestion. Further, since only few peers offer files, crippling a P2P network becomes a matter of just shutting down the few ”servers” in operation.

In the second type of free-riding the majority of nodes share files of little interest to each others, so-called *junk content*. Often, this is the case of P2P systems with a basic form of admission control that allow users to join the overlay only if they share a minimum volume of data (e.g., the case of DirectConnect). To pass the admission control users share, for example, their main Windows directory or the `/usr` directory in UNIX systems. The difference between this form of free-riding and the previous one is that in this case the possibility of hash or file name collision increases. In other words, it becomes more likely for two files with widely different content to have the same file name or hash value. This would generate false positive query replies. The net effect of this is a decrease in the service value provided by the network.

There are some reports [Orl03] revealing that junk content is not only a phenomenon of free-riding but also a deliberate attempt by media record companies to fight online piracy. The idea was to swamp P2P networks with a huge volume of decoys, for example files that appear to contain popular music but consist of a random set of bits. The decoys would make it difficult to find the real content and thus drastically diminish the value of the network. Further, it was hoped that decoy downloads, particularly those using swarming will acerbate the effect, since users would not check the quality of the content until the file download finished. The practice of maliciously inserting replicating junk content is called *network poisoning*.

Network poisoning is just one form of attack on P2P networks. Unfortunately, for current P2P systems network security appears to be a low priority item. Perhaps, more whistle-blowers such as [Orl03] will contribute to an increased security awareness for P2P system developers.

Chapter 3

The Gnutella Protocol

Gnutella was first released on March 14th, 2000 by Justin Frankel and Tom Pepper, founders of Nullsoft, a company which was acquired by America Online (AOL)[1]. As soon as the Gnutella software was released it was downloaded by a large number of users. Fearing legal problems, AOL stopped the distribution of Gnutella. However, Gnutella users were able to reverse engineer the protocol and create compatible software [Enc05]. The fall of Napster urged its users to look for a system without a central point of failure. Many of the users adopted Gnutella as a Napster replacement for the better.

Gnutella is a heavily decentralized P2P system. This is the opposite of Napster, which used a centralized directory. Servents can share any type of resources, although the currently available specification covers only computer files. The first "official" Gnutella protocol was labeled version 0.4 [Cli03]. Eventually, Gnutella version 0.6 [KM02] was released with improvements based on the lessons learned from the predecessor. The protocol is easily extensible, which has led to a variety of proprietary and non-proprietary extensions (e.g., ultrapeers and the Query Routing Protocol). For a while, the two protocol versions lived side by side and improvements were merged from the v0.6 line into the legacy v0.4 line. However, there are indications that July 1st 2003 was sort of a "flag day" when Gnutella v0.4 peers were blocked from the network. This was first discovered in the source code for `gtk-gnutella v0.92`, a Gnutella servent [Man06]. The software checks if the current date is later than July 1 2003. If true, it disables Gnutella v0.4 signaling.

The activities of Gnutella peers can be divided into two main categories: signaling

[1]Before Gnutella, Nullsoft created the famous Winamp media player.

and user data transfer (further referred to as data transfer). Signaling activities are concerned with discovery and maintenance of overlay topology, content search and other management functions. Data transfer occurs when a peer has localized a file of interest. Peers transfer files over direct HTTP connections to the nodes hosting the files.

3.1 Ultrapeers and Leaf Nodes

Initially, the Gnutella Network (GNet) was non-hierarchical. However, experience has shown that the abundance of signaling was a major threat to the scalability of the network [Rit01]. LimeWire (a company promoting an enhanced Gnutella servent) suggested therefore the introduction of a two-level hierarchy: Ultrapeers (UPs) and Leaf Nodes (LNs). UPs are faster nodes in the sense that they are connected to high-capacity links and have a large amount of processing power available. LNs maintain a single connection to their UP. A UP maintains 10-100 connections, one for each LN and 1-10 connections to other UPs [SR02]. The UPs perform signaling on behalf of the LNs, thus shielding them from large volumes of signaling traffic. A UP does not necessarily have leaf-nodes, in which case it works standalone.

Some servents may not be capable to become LNs or UPs for various reasons (e.g., they lack required functionality). In this case, they are labeled *legacy* nodes. In order to improve the overall scalability of the GNet and to preserve bandwidth, UPs and LNs may refuse to connect to legacy nodes.

According to the Gnutella Development Forum (GDF) mailing list, the Gnutella community has recently adopted what is called support for high outdegree [Fis03a]. This implies that UPs maintain at least 32 connections to other UPs and 100–300 connections to different lead nodes. LNs are recommended to maintain approximately 4 connections to UPs. The numbers may differ slightly between different Gnutella vendors. The claim is that high-outdegree support allows a peer to connect to the majority of GNet peers in 4 hops or less.

3.2 Peer Discovery

A Gnutella node that wants to join the overlay must solve the bootstrap problem. This means the node must obtain information about the listening socket of at least another peer that is already member in the overlay.

The old way to solve the bootstrap problem was to visit a web site that published up-to-date lists of known peers. The first step involved selecting one of the peers listed on the page, cut-and-paste its address (i.e., the listening socket) from the web browser into the Gnutella servent and trying to open a connection to it. This process would continue until at least one connection was successfully opened. At this point signaling traffic would, hopefully, reveal more peers to which the servent could connect. The addresses of newly found peers were cached in the local *hostcache* and reused when the servent application was restarted.

Since peers in general have a short life span (i.e., they enter and leave the network very often) [SGG02] the hostcache kept by each node frequently gets outdated. Gnutella Web Cache (GWC) servers[2] try to solve this problem. Each GWC server is essentially an HTTP server offering a list of active peers with associated listening sockets. The web page is typically rendered by a Common Gateway Interface (CGI) script or Java servlet, which is also capable of updating the list contents. UPs update the list continuously, ensuring that new peers can always join the overlay.

A list of available GWC servers is maintained at the main GWebCache web site. This list contains only registered GWC servers. Unofficial GWC servers exist as well.

New Gnutella peers implement the following bootstrap algorithm: upon start they connect to the main GWC Web site, obtain the list of GWC systems, try to connect to a number of them, and finally end up building their own hostcache. Alternatively, the node can connect to an unofficial GWC system or connect directly to a node in the GNet. The last option requires a priori knowledge about the listening socket of a GNet node.

Recently, it was observed that GWC servers were becoming overloaded. There appeared to be two reasons behind the heavy load: an increase in the number of GWC-capable servents and the appearance of a large number of misbehaving servents. The UDP Host Cache (UHC) protocol was suggested as a way to alleviate the problem. The protocol works as a distributed bootstrap system, transforming UHC-enabled servents into GWC-like servers [GDF05].

3.3 Peer Connections

Assuming that a Gnutella servent has obtained the listening socket of a peer, it will then attempt to establish a full-duplex TCP connection. The explanation below uses typical TCP terminology calling the servent that has done the TCP active open *client* and its peer *server*. Once the TCP connection is in place, a handshaking procedure takes place between the client and the server:

[2] Also abbreviated as GWebCache servers.

1. The client sends the string `GNUTELLA CONNECT/0.6<CR><LF>` where `<CR>` is the ASCII code for carriage return and `<LF>` is the ASCII code for line feed.

2. The client sends all capability headers in a format similar to HTTP and ends with `<CR><LF>` on an empty line, e.g.,

   ```
   User-Agent: BearShare/1.0<CR><LF>
   X-Ultrapeer: True<CR><LF>
   Pong-Caching: 0.1<CR><LF>
   <CR><LF>
   ```

3. The server responds with the string `GNUTELLA/0.6 <status code><status string><CR><LF>`. The `<status code>` follows the HTTP specification with code 200 meaning success. The `<status string>` is a short human readable description of the status code (e.g., when the code is 200 the string is typically set to OK).

4. The server sends all capability headers as described in step 2.

5. The client parses the server response to compute the smallest set of common capabilities available. If the client still wishes to connect, it sends `GNUTELLA/0.6 <status code><status string><CR><LF>` to the server with the `<status code>` set to 200. If the capabilities do not match, the client sets the `<status code>` to an error code and closes the TCP connection.

If the handshake is successful, the client and the server start exchanging binary Gnutella messages over the existing TCP connection. The connection lasts until one of the peers decides to terminate the session. At that point the peer ending the connection can send an optional signaling message to notify its peer. Then it closes the TCP connection.

Modern servents include a `X-Try` header in their response if they reject a connection. The header contains a list of listening sockets of recently active servents, to which the other peer can try to connect. The purpose of the `X-Try` header is to increase connectivity and reduce the need to contact a GWC server.

If the capability set used by the peers includes stream compression [Man03a] then all data on the TCP connection, with the exception of the initial handshake, is compressed. The type of compression algorithm can be selected in the capability header, but the currently supported algorithm is *deflate*, which is implemented in `zlib` [GA05].

3.4 Gnutella Message Headers

Each Gnutella message starts with a generic header that contains the fields shown in Figure 3.1 (the numbers in the figure denote bytes):

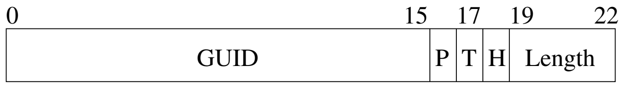


Figure 3.1: The Gnutella header

- Message ID using a Globally Unique ID (GUID) to uniquely identify messages on GNet. Leaving out some details, the GUID is a mixture of the node's Ethernet MAC address and a timestamp [LMS05].
- Payload type code, denoted by P in Figure 3.1, that identifies the type of Gnutella message. The currently supported messages are:

Message	Code (hex)
PING	0x00
PONG	0x01
BYE	0x02
QRP	0x30
VEND	0x31
STDVEND	0x32
PUSH	0x40
QUERY	0x80
QUERY_HIT	0x81
HSEP	0xcd

Table 3.1: Supported Gnutella messages and associated payload codes

- TTL to limit the signaling radius and its adverse impact on the network. Messages with TTL > 15 are dropped[3]. This field is denoted by T in Figure 3.1.
- Hop count to inform receiving peers how far the message has traveled, denoted by H in Figure 3.1.

[3]Nodes that support high outdegree drop messages with TTL > 4.

- Payload length in bytes to describe the length of the message, not including this header. The payload length indicates where in the byte stream the next Gnutella generic message header can be found.

The generic Gnutella header is followed by the actual message which may have its own headers. Also, the message may contain vendor extensions. Vendor extensions are used when a specific type of servent wants to implement experimental functionality not covered by the standard specifications. It is recommended to implement vendor extensions using the Gnutella Generic Extension Protocol (GGEP) [Tho02]. The protocol provides a transparent way for regular servents to interact with the vendor servents.

3.5 Topology Exploration

Each successfully connected pair of peers starts periodically sending PING messages between each other. The receiver of the PING message decrements the TTL in the Gnutella header. If the TTL is greater than zero the node increments the hop counter in the message header and then forwards the message to all its directly connected peers, with the exception of the one from where the message came. Note that PING messages do not carry any user data (not even the sender's listening socket). This means that the payload length field in the Gnutella header is set to zero.

PONG messages are sent only in response to PING messages. More than one PONG message can be sent in response to one PING. The PONG messages are returned on the *reverse path* used by the corresponding PING message. Each PONG message contains detailed information about *one* active Gnutella peer. It also contains the same GUID as the PING message that triggered it. The PONG receiver can, optionally, attempt to connect to the peer described in the message.

UPs use the same scheme, however they do *not* forward PINGs and PONGs to and from the LNs attached to them.

Gnutella peers are required to implement some form of flow control in an effort to prevent PING-PONG traffic generated by malfunctioning servents from swamping the network. A simpler flow control mechanism is specified in [Roh02b].

The BYE message is an *optional* message used when a peer wants to inform its neighbors that it will close the signaling connection. The message contains an error code along with an error string. The message is sent only to hosts that have indicated during handshake that they support BYE messages.

3.6 Resource Discovery

A Gnutella peer wishing to locate some specific resource (e.g., file) must assemble a QUERY message. The message describes the desired resource using a text string. For a file resource this is the file name. In addition, the minimum speed (i.e., upload rate) of servents that should respond to this message is specified as well. There may be additional extensions attached to the message (e.g., vendor extensions) but those are outside the scope of the thesis.

In Gnutella v0.4, the QUERY message is sent to all peers located one hop away, over the signaling connections established during the handshake. Peers receiving a QUERY message forward it to all directly connected peers unless the TTL field indicates otherwise. This is the controlled flooding approach, presented in Chapter 2.4, which is inefficient in terms of consumed bandwidth.

The newer Gnutella v0.6 attempts to alleviate the problems by introducing a form of selective forwarding called *dynamic query* [Fis03a]. A dynamic query first probes how popular the targeted content is. This is done by using a low TTL value in the QUERY message that is sent to a very limited number of directly connected peers. A large number of replies indicate popular content, whereas a low number of replies imply rare content. For rare content, the QUERY TTL value and the number of directly connected peers receiving the message are gradually increased. This procedure is repeated until enough results are received or until a theoretical limit of the number of QUERY message receivers is reached. This form of resource discovery requires all LNs to rely on UPs for their queries (i.e., LNs do not perform dynamic queries).

If a peer that has received the QUERY message is able to serve the resource, it responds with a QUERY_HIT message. The GUID for the QUERY_HIT message must be the same as the one in the QUERY message that triggered the response. The QUERY_HIT message lists each resource name that matches the resource description from the QUERY message[4] along with the resource size in bytes and other information. In addition, the QUERY_HIT messages contain the listening socket which to be used by the message receiver when it wants to download the resource. The Gnutella specification discourages the use of messages with size greater than 4 KB. Consequently, several QUERY_HIT messages may be issued by the same servent in response to a QUERY message.

Some servents use the metadata extension mechanism [Tha01] to allow for richer queries. The idea is that metadata (e.g., author, genre, publisher) is associated with files shared by a servent. Other servents can query those files not only by file name, but

[4]For example the string `linux` could identify a resource called `linux_redhat_7.0.iso` as well as a resource called `linux_installation_guide.txt.gz`. Thus, this query yields two potential results. Both results will be returned to the QUERY sender.

also by the metadata fields.

3.6.1 Query Routing Protocol

The purpose of ultrapeers is to reduce the burden put on the network by peer signaling. They achieve this goal by eliminating the PING messages among leaf nodes and by employing query routing. There are various schemes for ultrapeer query routing but the recommended one is the Query Routing Protocol (QRP) [Roh02a]. Ultrapeers signal among themselves by using PING and PONG messages.

QRP [Roh02a] was introduced in order to mitigate the adverse effects of flooding used by the Gnutella file queries and it is based on a modified version of *Bloom filters* [HB70]. The idea is to break a query into individual keywords and have a hash function applied to each keyword. Given a keyword, the hash function returns an index to an element in a finite discrete vector. Each entry in the vector is the minimum distance expressed in number of hops to a peer holding a resource that matches the keyword in the query. Queries are forwarded *only* to leaf nodes that have resources that match *all* the keywords. This substantially limits the bandwidth used by queries. Peers run the hash algorithm over the resources they share and exchange the routing tables (i.e., hop vectors) at regular intervals.

Individual peers (legacy nodes or UPs) may run QRP and exchange routing tables among themselves [Fis03b]. However, the typical scenario is that legacy nodes do not use QRP, LNs send route table updates only to UPs, and UPs propagate these tables only to directly connected UPs.

3.7 Content Distribution

Data exchange takes place over a direct HTTP connection between a pair of peers. Both HTTP 1.0 and HTTP 1.1 are supported but use of HTTP 1.1 is strongly recommended. Most notably, the use of features such as *range request* and *persistent connections* is encouraged as well.

The range request feature allows a peer to continue an unfinished transfer from where it left off. Furthermore, it allows servents to utilize swarming. Swarming is not part of the Gnutella protocol and regular Gnutella servents (i.e., servents that do not explicitly support swarming) can be engaged in swarming without being aware of it. From their point of view, a peer is requesting a range of bytes for a particular resource. The intelligence is located at the peer downloading data.

The persist connection feature is useful for swarming. It allows a peer to make several requests for different byte ranges in a file, over the *same* HTTP connection.

When a peer locates interesting content it should open a direct HTTP connection to the listening socket specified in the QUERY_HIT message. If the QUERY_HIT sender (i.e., the resource owner) is behind a firewall, incoming TCP connections are usually rejected.

To work around this problem, when a firewall is detected the downloader must send a PUSH message over the signaling connection. The message contains the listening socket of the resource requester and is routed along the reverse path of the received QUERY_HIT message. The resource owner can use the information in the PUSH message to establish a TCP connection to the requester's listening socket. If the TCP connection is established successfully, the host behind the firewall sends the following string over the signaling connection:

```
GIV <File Index>:<Servent Identifier>/<File Name><LF><LF>
```

The `<File Index>` and `<Servent Identifier>` are the values found in the corresponding PUSH message and `<File Name>` is the name of the resource requested. Upon the receipt of the message the receiver issues an HTTP GET request on the newly established TCP connection:

```
GET /get/<File Index>/<File Name> HTTP/1.1<CR><LF>
User-Agent: Gnutella<CR><LF>
Connection: Keep-Alive
Range: bytes=0-<CR><LF>
<CR><LF>
```

Figure 3.2 shows a simple GNet scenario, involving three legacy peers. It is assumed that Peer A has obtained the listening socket of Peer B from a GWC server. Using the socket descriptor, Peer A attempts to connect to Peer B. In this particular example, Peer B already has a signaling connection to Peer C.

The first three messages between Peer A and Peer B illustrate the establishment of the signaling connection between the two peers. The two peers may exchange capabilities during this phase as well.

The next phase encompasses the exchange of network topology information with the help of PING and PONG messages. The messages are sent over the TCP connection established previously (i.e., during the peer handshake). It is observed that PING messages are forwarded by Peer B from Peer A to Peer C and in the opposite direction. Also, it can be observed that PONG messages follow the reverse path taken by the corresponding PING message.

At a later time the Peer A sends a QUERY message, which is forwarded by Peer B to Peer C. In this example, only Peer C is able to serve the resource, which is illustrated

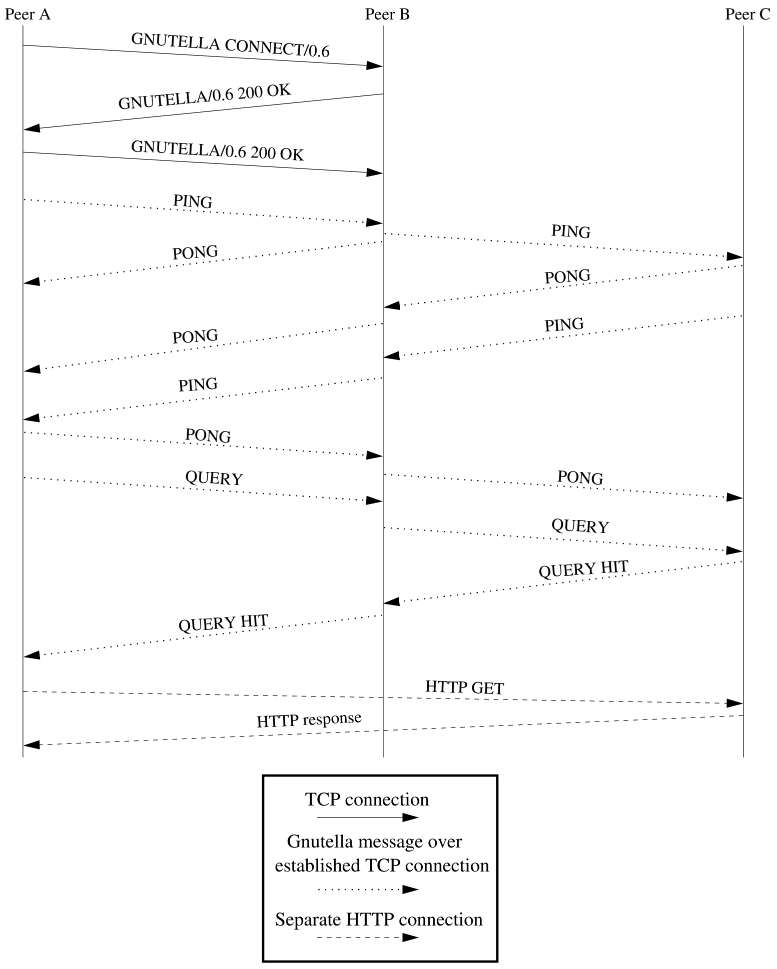


Figure 3.2: Example of a Gnutella session

by the QUERY_HIT message. The QUERY and QUERY_HIT messages use the existing TCP connection, just like the PING and PONG messages. Again, it is observed that the QUERY_HIT message follows the reverse path taken by the corresponding QUERY message.

Finally, Peer A opens a direct HTTP connection to Peer C and downloads the resource by using the HTTP GET method. The resource contents are returned in the HTTP response message.

The exchange of PING-PONG and QUERY-QUERY_HIT messages continues until one of the peers tears down the TCP connection. A Gnutella BYE message may be sent as notification that the signaling connection will be closed.

3.7.1 Optimizations

The Hash/URN Gnutella Extensions (HUGE) specification [Moh02] provides a way to identify files by Uniform Resource Names (URN) and in particular by hash values, such as SHA-1. The advantages of using HUGE and SHA-1 is that files with the same content but different names can be discovered through the QUERY–QUERY_HIT mechanism and that file integrity can be checked upon download by recomputing the SHA-1 hash value.

In order to speed up file downloads and to distribute the load among servents, when a peer sends a QUERY_HIT message, it includes a list of peers that are known to have the same file (i.e., the download mesh). The simplest way a servent can build such a list, is to remember the list it obtained itself when it downloaded the file. Download meshes require support for the HUGE extension. The main benefit lies in their ability to enable efficient swarming.

Partial File Sharing (PFS) is an optimization of swarming and download meshes. Servents that support PFS do not wait to download the whole file before replying to matching QUERY messages. If a servent requests the file before its download has completed, the servent that has the partial file sets the `Content-Range` header in the HTTP reply informing the other peer about the amount of available data.

3.8 Other Features

Gnutella has support for many other important features, albeit outside the scope of the thesis. The remainder of this section will present some of these features briefly.

3.8.1 Horizon Size Estimation Protocol

The Horizon Size Estimation Protocol (HSEP) [Sch04] is used to obtain estimates on the number of reachable resources (i.e., nodes, shared files and shared kilobytes of data). Hosts that support HSEP announce this as part of the capability set exchange during the Gnutella handshake. If the hosts on each side of a connection support HSEP, they start exchanging HSEP message approximately every 30 seconds. The HSEP message consists of `n_max` triples. Each triple describes the number of nodes, files and kilobytes of data estimated at the corresponding number of hops from the node sending the message. The `n_max` values is the maximum number of hops supported by the protocol. 10 hops is the recommended value [Sch04].

The horizon size estimation can be used to quantify the quality of a connection. For example the higher the number of reachable resources, the higher the quality of the connection.

3.8.2 File Magnets and Magma Lists

Building on HUGE, file magnets are the bridge between the Web and P2P networks. Web pages can include special Universal Resource Locator (URL) links called file magnets, which encode URNs to resources available on the P2P network. When a user clicks on such a link, the web browser transfers the URN to the local Gnutella servent, which will perform a query on GNet.

A Magma list is a list of file magnets, for example the favorite documents, music or pictures shared by a Gnutella user.

3.8.3 Passive/Active Remote Queueing (PARQ)

Most servents limit the number of uploads that can occur simultaneously, in order to preserve bandwidth. When PARQ is used, download requests are queued at the servent hosting the file [Man03b]. The specification allows servents to check their place in the download queue. It allows hosts to temporarily (maximum 5 minutes) become unavailable. This feature can be useful if the servent crashes or temporarily looses its Internet connection.

3.8.4 Reliable UDP File Transfer

Firewall-to-Firewall (F2F) or Reliable UDP File Transfer (RUDP) allows two firewalled hosts, both of them connected to Internet through NAT servers, to transfer files among themselves over UDP [Gnu05]. The technique to open the UDP ports in the firewall is known as “UDP hole punching” and it is fairly well documented in [SFK05].

3.8.5 LAN Multicast

Recently, a specification [BMQD04] was made available that allows servents located on the same Local Area Network (LAN) to take advantage of IP multicast when transferring files. The advantages of LAN multicast is that file transfers are more efficient due to IP multicast and generally much faster due to higher bandwidth, lower latency and lower number of hops.

Chapter 4

Traffic Measurements

Traffic measurements tend to be divided into active and passive measurements. The main difference between the two is that in active measurements specific patterns of traffic are injected into the network and analyzed when they exit the network. Changes on the injected traffic pattern are used to draw inferences about various properties of the network. The `ping` and `traceroute` applications are very simple examples of active measurement tools. Web and P2P crawlers that discover the network topology and properties of the nodes in it are another example. In the case of passive measurements, traffic flows seen at specific nodes are observed or recorded, without the necessity of sending any traffic in the network. This is the approach used for this thesis and the subject of this chapter. In particular, the focus is on passive application layer measurements.

The goal of network traffic measurements is to capture observable properties of data transfers over a network. The properties can be ordered in a three-level hierarchy of statistics: message unit statistics, session statistics and flow statistics.

An *application message unit* is a chunk of application data, which, if received correctly, is recognized as a complete protocol data unit (PDU).

Related data are said to form a *session*. For example, a TCP session covers all data between and including the three-way handshake and the FIN segments. In the case of HTTP, a session is roughly defined as all TCP sessions required to download a web page with all embedded items. A general application layer definition for a session can be that of a group of related message units exchanged for the purpose of accomplishing a specific task.

A *flow* is defined by arbitrary data exchanged between two hosts. A flow typically contains one or more sessions. Sometimes, it is interesting to observe the behavior of several flows as a whole, e.g., all the flows from a specific source or all the flows going to a particular destination. In this case the group of flows in question is called an *aggregate flow*.

General statistics that apply to all levels of the hierarchy are interarrival times, duration, transfer rates and sizes. Other specialized statistics may be available, depending on the application, e.g., peer swarm size [EIP05] in BitTorrent, a replicating P2P distribution system.

The simplest way to report the result of measurements is to organize the statistics as time series, where each element of the time series consists of a timestamp, statistic type and measured value. In practice, the element structure is more sophisticated, containing meta-information, events triggered within the application and even nested elements.

There are two main approaches to perform application layer measurements for network traffic. In the first approach, called *application logging*, the traffic is measured by the application itself. The other approach is to obtain measurements indirectly, by monitoring traffic at the link layer and performing application flow reassembly using a specially designed application. This approach is referred to as *flow reassembly*. A mixture of these two approaches is possible as well.

4.1 Application Logging

Unless already supported, application logging requires changes in the application software to record incoming and outgoing network data and other events of interest, e.g., transitions between application states, failures, CPU and memory usage. This implies modifying the application source code. In the case of open source software these changes can be performed rather straightforwardly. However, for closed source software one needs to negotiate an agreement with the vendor to obtain and modify the source code.

The advantage of application logging is that measurement data is readily available from within the application itself. Measurement code embedded at relevant places in the application is able to continuously monitor all variables of interest. On the other hand, the main disadvantage associated with this method is related to timestamp accuracy. The accuracy of the timestamp is affected by three main factors: drift in the frequency of the crystal controlling the system clock, latency in the operating system (OS) and latency in the network stack.

The frequency drift of the crystal is due to temperature changes and age. Its influence on the timestamp is in the order of about 1 μs [PV02].

Latency in the OS refers to the delay between the time when a user-space process requests a timestamp from the operating system and the time when the timestamp is available to the process. The delay is largely accounted for by scheduling in the kernel when the calling process is temporarily preempted by other processes. The problem is increasingly worse for interpreted programs, e.g., the reference BitTorrent client which is a Python script. In this case the timestamps are subject to additional scheduling imposed by the interpreter.

A significant amount of queueing and scheduling occurs in the TCP/IP stack as well, especially in the routines for IP and TCP reassembly. The effect is that timestamps at the application layer are only indicative for the actual time when packets enter or leave the link-layer at the node in question.

4.2 Flow Reassembly

The flow reassembly method attempts to address some of these problems by moving the measurements closer to the network. Link-layer measurements have enjoyed a long tradition in the network community. However, since the interest is now moving towards events at the application layer, one needs to develop dedicated software able to decode application layer messages from the observed link-layer traffic, essentially replicating parts of the application of interest.

Flow reassembly involves mainly three stages: link-layer capture, transport stream reassembly, (e.g., TCP reassembly), and application message decoding.

A plethora of link-layer capture software is available under very liberal licenses on the Internet, (e.g., `tcpdump` and `ethereal` [JLM05, Cc05]). The common denominator for this software is that in a shared-medium LAN such as Ethernet, the capturing software forces the network interface to work in promiscuous mode, thus enabling it to monitor all traffic in the LAN. However, an issue to consider carefully when selecting capture software, is the timestamping operation. The operation should be performed as close as possible to the place where the frame is read from the network card (if possible in the network driver or on the card itself). Failure to do so may lead to inaccuracies that are similar to those of application logging.

Transport stream reassembly deals with missing or duplicate packets and with packets arriving in wrong order. At the IP layer this involves reassembly of IP fragments. At the TCP layer, the transport stream reassembly replicates the TCP reassembly functionality from the network stack.

The main problem with regards to TCP reassembly is to obtain the same TCP state transitions that occurred at the time when the traffic was recorded. This is particularly hard to do in a heterogeneous network environment, since different OSs handle special TCP conditions in different ways. For example, retransmitted segments may overlap the data received previously and one must decide whether to keep the old or the new data. Windows and UNIX take opposing views of this scenario [PN98]. A solution to minimize the inconsistencies in protocol implementations is to use a traffic normalizer [HP01]. Similar problems apply to reassembly of IP fragments.

Application message decoding uses reassembled transport layer flows to obtain application messages exchanged by end-points. This is also the point where statistics (e.g., the ones described at the beginning of this chapter) can be computed.

The main advantage of flow reassembly is that it provides a more accurate view of how the application affects the network. Furthermore, flow reassembly can be run on a dedicated host different from the hosts participating in the application session. Such a dedicated node has also the possibility to analyze all traffic passing by the recording interface. In contrast, application logging can only provide information about the flows in which the measuring host is an active participator. Furthermore, the flow reassembly method can save link-layer traffic to disk for off-line analysis.

A major disadvantage associated with flow reassembly is that all application states must be inferred from the recorded network traffic. This is not always possible, since certain application state transitions may be independent of network events. Another disadvantage is that a lot of existing functionality (e.g., IP and TCP reassembly) are duplicated. A well-known programming mantra states that the probability to encounter bugs increases proportionally to the volume of new code. An even more serious problem is related to the link-layer capture. On heavily loaded links, the hardware may not be able to record all data and will start dropping frames. This has an impact on the host performing the measurements but not necessarily on the host participating in the application layer session.

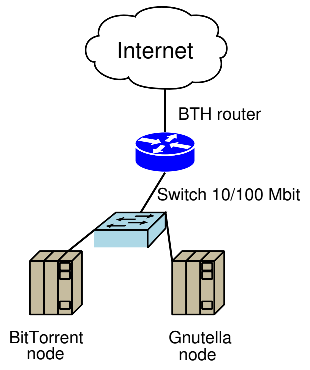
Off-line traffic analysis features similar to those found in flow reassembly can be implemented using application logging, by adding suitable message recording points in the software application. This means in fact a measurement method that is a mixture between application logging and flow reassembly. Such a mixed methodology has the advantages of both methods, e.g., no need to infer application state from link-layer traces, and no need to decide beforehand what statistics to collect.

4.3 BTH Measurement Infrastructure

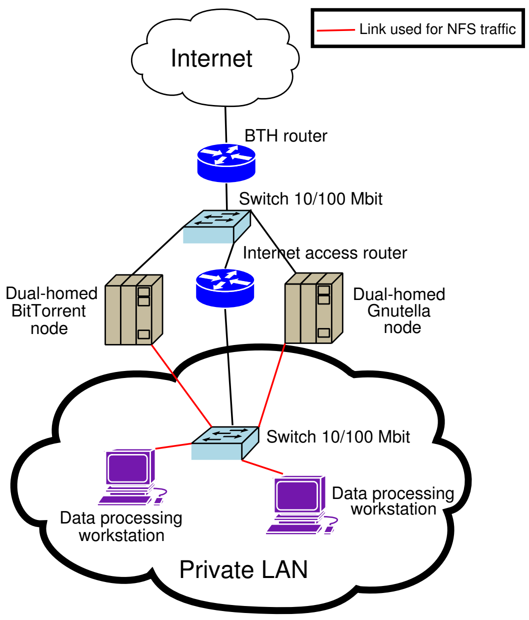
A measurement infrastructure dedicated to P2P measurement has been developed at BTH [IEPN04a, IEPN04b, EIP05]. It consists of peer nodes and protocol decoding software. `Tcpdump` [JLM05] and `tcptrace` [Ost05] are used for traffic recording and protocol decoding. Although the infrastructure is currently geared towards P2P protocols, it can be easily extended to measure other protocols running over TCP.

The BTH measurement nodes run the Gentoo Linux 1.4 operating system, with kernel version 2.6.5. Each node is equipped with an Intel Celeron 2.4 GHz processor, 1 GB RAM, 120 GB hard drive, and 10/100 Mbit/s Ethernet network interface. The network interface is connected to a 100 Mbit/s switch in the lab at the Department of Telecommunication Systems, which is further connected through a router to the GigaSUNET backbone as shown in Figure 4.1(a).

Measurements performed at BTH showed that the recording step alone accounts for about 70% of the total time taken by measurements. Traffic analysis is not possible when the hosts are recording traffic. The main reason is the protocol decoding phase, which is I/O intensive and requires large amounts of CPU power and RAM. To overcome this problem a new distributed measurement infrastructure shown in Figure 4.1(b) is under development.



(a) Measurement setup



(b) Distributed measurement setup

Figure 4.1: Measurement network infrastructures

When used in the distributed infrastructure, the P2P nodes are equipped with an additional network interface, which we refer to as the *management* interface. P2P traffic is recorded from the primary interface and stored in a directory on the disk. The directory is exported using the Network File System (NFS) over the management interface. Data processing workstations can read recorded data over NFS as soon as it is available. Optionally, the data processing workstations can be located in a private LAN or Virtual Private Network (VPN) in order to increase security, save IP address space and decrease the number of collisions on the Ethernet segment. In this case, the Internet access router provides Internet access to the workstations, if needed.

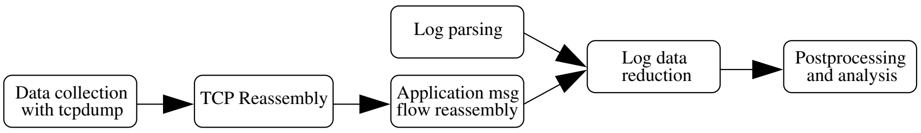

Log parsing
Data collection with tcpdump
TCP Reassembly
Application msg flow reassembly
Log data reduction
Postprocessing and analysis

Figure 4.2: Measurement and traffic stages

Figure 4.2 shows the measurement process flow. The first stage from the left is the traffic collection stage that is described in Section 4.3.1. The following two stages are the TCP reassembly stage described in Section 4.3.2 and the application message flow reassembly described in Section 4.3.3. The last two stages, log data reduction and postprocessing and analysis, are described in Section 4.3.4. The log parsing stage was used for analysis of BitTorrent application logs [EIPN04, EIP05]. However, the log parsing stage is not relevant to this thesis and will not be described any further.

4.3.1 Traffic Collection

Each measurement node has `tcpdump` 3.8.3 installed on it. When the node is running measurements, `tcpdump` is started before the Gnutella servent in order to avoid missing any connections. `Tcpdump` can also be run on a different node in the network, provided that the ultrapeer switch port is mirrored to the port where the `tcpdump` host is recording or if the switch is replaced with a hub and both the `tcpdump` host and the ultrapeer are connected to it.

During the data collection stage, `tcpdump` collects Ethernet frames from the switch port where the ultrapeer node is connected. The collected data is saved in PCAP format. Since most P2P applications can use dynamic ports, all traffic reaching the switch port must be collected. However, to increase the performance during data collection and data processing, one can turn off most or all server software on the ultrapeer node. It is possible, in addition, to apply a filter to `tcpdump` that drops packets used by traditional services, which are running on well-known ports (e.g., HTTP, FTP, SSH).

The volume of collected data can be quite large, (e.g., the resulting trace file could grow well beyond 2 GB in less than one day). The data volume is directly related to the number of peers the servent connects to. In our case we observed on average 130 peers (100 leaf nodes and 30 ultrapeers) and collected approximately 33 GB PCAP data in eleven days. Most file systems do not allow this amount of data to be stored in one file. The solution to this problem is to have `tcpdump` spread the recorded data across several files, each 600 MB large. This file size is suitable for storage on a recordable CD.

4.3.2 TCP Reassembly

Assuming that the measured P2P application runs over TCP, the next step is to reassemble the TCP frames to a flow of ordered bytes. The TCP reassembly module builds on the TCP engine available in `tcptrace`.

The module reads the `tcpdump` traces in the order they were created. Each trace is scanned for TCP connections. When found, they are stored in a list with connection records. Further, when a new TCP segment is found in the trace file, the module scans the connection list comparing the socket pair of the segment with each entry in the list. If no entry matches the socket pair of the new segment, then a new connection is considered to be found and a record is created for it, which finally is added to the connection list. Otherwise, the connection record matching the socket pair is retrieved and sent together with the new segment to the TCP reassembly engine.

The TCP reassembly engine is similar to the one used by the FreeBSD TCP/IP stack as described in [WS95]. For each active connection, the reassembly engine keeps a doubly linked list, which is referred to as the reassembly list. When given a connection record and a new segment, it retrieves the correct reassembly list and then it inserts the new segment in the correct place in the list. The reassembly engine is capable of handling out-of-order segments as well as forward and backward overlapping between segments.

4.3.3 Application Flow Reassembly

Whenever new data is available, the application data reassembly module is notified. Upon notification, it asks the TCP reassembly module for a new segment from the reassembly list corresponding to the socket pair received with the notification. When it receives the new segment, it interprets the contents according to the specification for the protocol it decodes. Since application messages may span several segments and since a segment may contain data from two consecutive messages, each segment is appended to the end of a data buffer before further processing. A contiguous data flow containing at least one application message is thus created.

In the case of a new Gnutella connection, the application reassembly module first waits for the handshake phase to begin. If the handshake fails, the connection is marked invalid and it is eventually discarded by the memory manager.

If the handshake is successful, the application reassembly module scans the capability lists sent by the nodes involved in the TCP connection. If the nodes have agreed to compress the data, the connection is marked as compressed. Further segments received from the TCP reassembly module for this connection are first sent to the decompressor, before being appended to the data buffer.

The decompressor uses `zlib`'s[GA05] *inflate()* function to decompress the data available in the new segment. Upon successful decompression the decompressed data is appended to the data buffer.

Immediately after the handshake phase, the application reassembly module attempts to find the Gnutella message header of the first message. Using the payload length field, it is able to discover the beginning of the second message. This is the only way to discover message boundaries in the Gnutella protocol and thus track application state changes. Based on the message type field in the message header, the corresponding decoding function is called, which outputs a message record to the log file. The message records follow a specific format required by the postprocessing stage. The format is described briefly in Appendix C.

4.3.4 Postprocessing

Since the logs can grow quite large, they can be processed through an optional stage of data compression. The compression is achieved by using the on-the-fly `deflate` compression offered by `zlib`. Additional data reduction can be achieved if the user is willing to sacrifice some detail by aggregating data over time.

The postprocessing module interprets the (optionally compressed) log data and it is able to demultiplex it based on different types of constraints: message type, IP address, port number, etc. The data output format of this stage is suitable for input to numerical computation software such as MATLAB and standard UNIX text processing software such as `sed`, `awk` and `perl`.

Chapter 5

Statistical Modeling

The measurement infrastructure described in the previous chapter was used to collect P2P network traffic crossing the BTH ultrapeer. By decoding the recorded traffic data, flows were recreated at several layers in the TCP/IP stack. The flows consist of discrete protocol data units: IP datagrams at the network layer, TCP segments at the transport layer, and finally, Gnutella messages at the application layer. The Gnutella messages can also be logically grouped in peer sessions. The time when the protocol data units reached the link layer and their size were recorded. For peer sessions, the session duration was recorded in addition to time and size. These quantities randomly change value over time. The term random does not imply that the Gnutella system behaves haphazardly, but rather that its traffic patterns are so complex that a deterministic description is not feasible. Instead, a statistical approach similar to the one introduced in [Pax94] is better suited to describe the quantities of interest.

The first section of this chapter presents the framework used for data analysis in this thesis. Elements of traffic self-similarity and heavy-tailed distributions are discussed in Section 5.2. Methods to identify distribution family from which a data sample is drawn are the subject of Section 5.3. Section 5.4 focuses on the problem of parameter estimation for such distributions. After the parameters of the distribution have been estimated, the distribution is fully specified. Section 5.5 is concerned with goodness-of-fit methods, which are procedures for quantifying how well the fully specified distribution matches the data. Since single distributions cannot always provide a good fit, especially when data shows multi-modal tendencies, Section 5.6 presents a method to fit finite mixture distributions. The information from these sections is combined into a modeling methodology, detailed in Section 5.7.

5.1 Framework

Each quantity of interest is modeled by a random variable X that changes its value whenever a new protocol data unit (or session) is considered. In other words, X obtains its values from the *entire* population of possible values, not only from those available from the recorded network traffic. The actual values taken by X are denoted by the small letter x. The random variable X is assumed to have a theoretical distribution $F_X(x;\theta)$.

Definition 5.1. The theoretical Cumulative Distribution Function (CDF) $F_X(x;\theta)$ of a random variable X is defined as

$$F_X(x;\theta) \triangleq P[X \leq x], \tag{5.1}$$

where $\triangleq$ is the equality by definition operator, x is some value on the real line, and θ is a set of one or more parameters that control the distribution function (e.g., $\theta = \{\mu, \sigma\}$ in the case of the normal distribution). □

Definition 5.2. It is assumed that a CDF $F_X(x;\theta)$ has a corresponding Probability Density Function (PDF) $f_X(x;\theta)$ defined as

$$f_X(x;\theta) \triangleq \frac{dF_X(x;\theta)}{dx}. \tag{5.2}$$

The derivative must exist at all points of interest otherwise impulse functions are required as in the case of discrete distributions [Kle75]. □

It is often useful to observe how fast the CDF decays for large values of x. For that purpose it is better to use the Complementary CDF (CCDF) function.

Corollary 5.1. Assuming a CDF function $F_X(x;\theta)$, the corresponding CCDF function is:

$$\overline{F}_X(x;\theta) \triangleq 1 - F_X(x;\theta) = P[X > x] \tag{5.3}$$

□

For each quantity of interest, the set of values extracted from the recorded traffic is considered a *random sample* from the population of the random variable X. The elements of the random sample are denoted by $X_1, X_2, \ldots, X_n$ and the actual recorded values by $x_1, x_2, \ldots, x_n$. The index n is the number of available values from the measurement. The set of recorded values $x_1, x_2, \ldots, x_n$ is called the *data sample* to differentiate it from the *random sample* where the elements are random variables.

The term random sample is used in the traditional sense, which implies that the samples $X_1, X_2, \dots, X_n$ are independent and identically distributed. This may be seen as a problem since there have been several reports over the years showing evidence that certain types of network traffic exhibit correlations over large time scales [LTWW94, PF95, CTB98]. Models that can take into account that specific type of correlations (e.g., superpositioned ON-OFF sources or M/G/∞) tend to be very complex. Whereas accounting for correlations can be critical for some applications, the aim here is to obtain analytically tractable models suitable for use in standard simulators, such as `ns-2` [sof06] and `OmNeT++` [Var06]. Therefore, the models presented in this work take the form of single probability distributions or mixtures of two distributions.

The modeling methodology employed in this work consists roughly of three steps:

i) Identify a distribution family $F(\cdot)$ through exploratory data analysis.

ii) Using the available data, estimate the parameter(s) θ of the distribution from the previous step. Denote the estimated parameter(s) by $\hat{\theta}$ and the estimated distribution by $\hat{F}_X(x;\hat{\theta})$.

iii) Quantify the quality of the fit.

Each step is thoroughly explained in the subsequent sections. Before that, terminology and notation used to describe several important properties of network traffic are explained.

5.2 Traffic Self-Similarity

In 1994, Leland et al. [LTWW94] published a seminal paper in which they show that aggregated Ethernet traffic in a LAN presented burstiness on a wide range of time scales. The implications of the paper were that traditional traffic models based on the Poisson process would underestimate the variations of network traffic and the peak values generated.

The observed phenomenon is called statistical *self-similarity* and it implies that specific statistical properties are repeated across many time scales, in a fractal-like behavior. In order to describe self-similarity, the authors of the paper used a discrete-time covariance stationary stochastic process[1] X_t, $t = 1, 2, \dots$ to describe the volume of

[1]A covariance stationary process is a second-order (wide-sense) stationary stochastic process.

traffic collected from the LAN. The following characteristics of a discrete-time second-order stationary process are useful for describing their finding [Sta02]:

$$\begin{aligned}
\mu &= E[X_t] && \text{mean}\\
\sigma^2 &= \mathrm{Var}[X_t] = E\left[(X_t-\mu)^2\right] && \text{variance}\\
R_{X_t}(\tau) &= E[X_t X_{t+\tau}] && \text{autocorrelation}\\
C_{X_t}(\tau) &= R_{X_t}(\tau) - \mu^2 && \text{autocovariance}\\
\rho_{X_t}(\tau) &= \frac{C_{X_t}(\tau)}{\sigma^2} && \text{correlation coefficient}
\end{aligned} \tag{5.4}$$

It should be noted that many papers on the topic of self-similarity refer to the correlation coefficient as autocorrelation function. Furthermore, these papers often assume zero mean, which means that $R(\tau)$ and $C(\tau)$ are equivalent.

What Leland et al. did, was to analyze the X_t process at increasingly larger scales of time aggregation. They defined a new process $X_k^{(m)}$ such that

$$X_k^{(m)} = \frac{1}{m} \sum_{t=m(k-1)+1}^{mk} X_t \tag{5.5}$$

where $k = 1, 2, \ldots$ and m defines the scale of aggregation. The traditional theory based on Poisson processes predicted that when m grows the process tends to white noise. However, the researchers found that $X_k^{(m)}$ retains its bursty structure across the time scales. Further analysis revealed that

$$\begin{aligned}
&\mathrm{Var}\left[X_k^{(m)}\right] \sim \frac{\mathrm{Var}[X_t]}{m^{\beta}}, \qquad \beta = 2-2H,\ 0.5 \le H \le 1\\
&\lim_{m\to\infty} R_{X_k^{(m)}}(k) = R_{X_t}(k)
\end{aligned} \tag{5.6}$$

which is consistent with the definition of statistical self-similarity and long-range dependence [Sta02, WPRT03]. The equations show that statistical properties of the process are preserved (apart from a scaling factor) across time scales. The term H denotes the Hurst parameter and is used to measure the persistence of a process. By persistence it is meant the range of dependence or "memory", which is inherent in the time series making up the process. An asymptotic second-order self-similar process process is called a Short-Range Dependent (SRD) process for H-values close to 0.5. The dependence of the process is increasingly larger for H-values close to 1. This type of process is called Long-Range Dependent (LRD) [PW00]. Although it is possible to define long-range processes that are not self-similar or self-similar processes that are not long-range dependent, typical traffic models assume the range $0.5 < H < 1$ where Long-Range Dependent (LRD) processes are self-similar and vice versa [Sta02].

For a SRD process the autocovariance function, $C(\tau)$, decays geometrically fast:

$$C(k) \sim c_1 a^k, \quad \text{as } k \to \infty, \quad 0 \leq c_1 < \infty,\ 0 < a < 1. \tag{5.7}$$

where $\sim$ means "asymptotic to". On the other hand, $C(\tau)$ decays hyperbolicly slow for a LRD process:

$$C(k) \sim c_2 k^{-b}, \quad \text{as } k \to \infty, \quad 0 \leq c_2 < \infty,\ 0 < b < 1. \tag{5.8}$$

The consequence is that for a SRD process the autocovariance function is summable, whereas for a LRD process it is not: $\sum_{k=-\infty}^{-\infty} C(k) = \infty$ [WPRT03]. This is the phenomenon of persisting correlations mentioned in the previous section in conjunction with sample independence.

The paper of Leland et al. was followed by others that provided ample evidence that self-similarity is present in various types of network traffic. In addition, some of the papers [PKC96, WTSW97, CB97] investigated the origins of self-similarity. Informally, the results put forward suggest that self-similarity is caused by traffic sources and human behavior, "think time", when both factors exhibit very large degrees of variability. Such variability can be best characterized by *heavy-tailed* distributions. Heavy-tailed distributions are a subset of a larger class of *subexponential* or *long-tailed* distributions. Long-tailed distributions decay slower than exponential distributions. The term tail refers to large x-values for which the corresponding CCDF value is very small, but non-negligible. Heavy-tailed distributions are a more restrictive subclass, since they require infinite variance. A random variable X has a heavy-tailed distribution if:

$$\lim_{x \to \infty} \overline{F}_X(x) = \lim_{x \to \infty} P[X > x] = cx^{-\alpha}, \quad 0 < \alpha < 2,\ c > 0 \tag{5.9}$$

When the *tail index* α is $0 < \alpha < 1$, the heavy-tailed distribution has infinite mean in addition to infinite variance. This is in contrast to the larger class of long-tailed distributions that posses finite moments [PW00].

The Pareto distribution is a good example of heavy-tailed distribution. The lognormal and Weibull distributions are subexponential, but not heavy-tailed. In particular, the Weibull distribution has finite variance [PW00]. Paxson and Floyd provide proof that the lognormal distribution is not heavy-tailed [PF95]. Gaussian or Gamma and exponential distributions are called *light-tailed* distributions and are not part of the subexponential class [PW00]. For light-tailed distributions the CCDF-values in the tail are negligible.

5.3 Exploratory Data Analysis

The first step in the modeling methodology employed is to identify a distribution family $F_X(x,\theta)$. This is done through an Exploratory Data Analysis (EDA) approach that combines graphs of the data (e.g., histograms and distribution plots) and summary statistics (e.g., mean, median and standard deviation) [DS86, MB03].

The histograms and distribution plots are the main EDA tools. Using them, the EDA user is aided in recognizing a family of distributions that provides good match for the data. However, the summary statistics provide some numerical guidance in case of uncertainty. Even so, this procedure is highly subjective.

After the unknown parameters of the distribution family are estimated, the candidate distribution is fully specified. The next step is to use formal numerical methods to asses the quality of the fit, as described in Section 5.5.

5.3.1 Summary Statistics

Five different types of statistics are used to summarize a random sample: maximum, minimum, mean, median and standard deviation. All definitions in this section assume a random sample $X_1, X_2, \ldots, X_n$ of length $n > 1$. The corresponding order statistics are $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$.

Definition 5.3. The $X_{(n)} = \max[X_1, \ldots, X_n]$ and $X_{(1)} = \min[X_1, \ldots, X_n]$ statistics select from the random sample the largest and smallest value, respectively. □

The difference $X_{(n)} - X_{(1)}$ defines the range of the data sample.

Definition 5.4. The sample mean $\overline{X}$ is defined as

$$\hat{\mu} = \overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{5.10}$$

□

$\overline{X}$ is an unbiased estimator of the first population moment, that is of the expected value $E[X]$. For a symmetric distribution the actual value of $\overline{X}$ represents the "center" of the data range. For a skewed distribution, the median statistic is a more appropriate representation of centricity.

Definition 5.5. If the sample size is odd, $n = 2k+1$, the sample median is the middle order statistic $X_{(k+1)}$. If the sample size is even, $n = 2k$, the sample median is the average of the two middle order statistics

$$\text{median} = \frac{X_{(k)} + X_{(k+1)}}{2} \tag{5.11}$$

□

Definition 5.6. The sample standard deviation is defined as

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2} \tag{5.12}$$

□

5.3.2 Histogram Plots

A histogram plot is a graph of tabulated frequencies for a univariate data sample $x_1, x_2, \ldots, x_n$ of length n. If the frequencies are normalized such that the area below the histogram is equal to one, then the histogram can be viewed as rough estimate of the probability density function.

In order to build a histogram one must begin by dividing the range r of the data into a number of m contiguous bins. Each bin i covers a portion of length[2] L of the data range. The boundaries of the bin i are denoted by b_i and b_{i+1}. Next, the data values are sorted and placed into bins that correspond to their value. The number of entries in each bin represents the frequency f_i of the bin i. To obtain the probability of each bin, the frequencies f_i are normalized such that the probability p_i of bin i is:

$$p_i = \frac{f_i}{n} \tag{5.13}$$

Then, according to [LK00]

$$p_i \approx P[b_i < X \le b_{i+1}] = \int_{b_i}^{b_{i+1}} f_X(x)\,\mathrm{d}x = f_X(y)\,L \quad \text{for some } y \in (b_i, b_{i+1}) \tag{5.14}$$

An important question is how to choose the bin length L or equivalently the number of bins m. Histograms using bins that are too wide fail to reveal specific characteristics of the data such as multi-modality (i.e., mixture of distributions) or impulses at the origin. These are called undersmoothed histograms. On the other hand, if the bin width

[2]Bins of equal length are assumed.

is too small the histogram is likely have a jagged appearance that could complicate the identification of the underlaying distribution or even worse, it may present false evidence of multi-modality. In this case the histogram is called an oversmoothed histogram. Research into optimal bin width has lead to the thumb rules [Wan97, VR99] presented in Table 5.1.

Name	Bin width
Sturges' formula	$L = r/(1+\log_2 n)$
Scott's rule	$L = 3.49\hat{\sigma}\, n^{-1/3}$
Friedman-Diaconis	$L = 2\,(q_{.75} - q_{.25})\, n^{-1/3}$

Table 5.1: Various rules for choosing histogram bin width

The terms $\hat{\sigma}$, $q_{.75}$ and $q_{.25}$ denote the estimated standard deviation, the 0.75-quantile, and the 0.25-quantile, respectively. Figure 5.1 shows an example of how the histogram of specific data can look like when the bin width is too large, too small and when it is chosen using the Friedman-Diaconis method.

The rules in Table 5.1 work well in many situations. Unfortunately, none of them is a panacea. In fact, for some distributions it is necessary to manually adjust the number of bins in order to obtain a smooth histogram [VR99].

5.3.3 EDF Plots

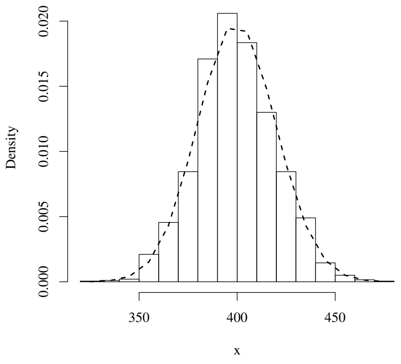
The Empirical Distribution Function (EDF) $\mathscr{F}_n(x)$[3] of a random sample is an approximative representation of the true CDF for population from which the sample is drawn.

Definition 5.7. Given a random sample $X_1, X_2, \ldots, X_n$ of length n drawn from a distribution $F_X(x)$, denote the corresponding order statistics by $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$. Then, the EDF $\mathscr{F}_n(x)$ is defined as

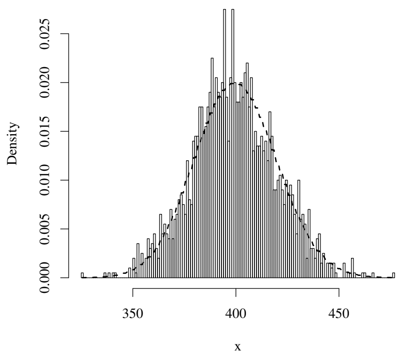
$$\mathscr{F}_n(x) \triangleq \begin{cases} 0 & x < X_{(1)} \\ \frac{i}{n} & X_{(i)} \leq x < X_{(i+1)} \\ 1 & X_{(n)} \leq x. \end{cases} \tag{5.15}$$

□

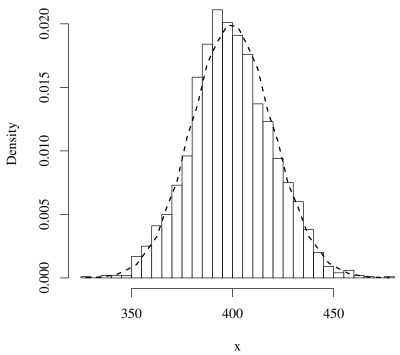
[3]Normally, the notation $F_n(x)$ is used to denote an EDF. However, this notation could conflict here with the notation used for a CDF and is therefore written using a calligraphic letter: $\mathscr{F}_n(x)$.



(a) Undersmoothed



(b) Oversmoothed



(c) Friedman-Diaconis

Figure 5.1: Poisson distribution with $\lambda = 400$: histogram for 2000 samples and superimposed density function.

For large samples, $\mathscr{F}_n(x)$ converges uniformly to the population $F_X(x)$ for all x-values [DS86].

Corollary 5.2. Assuming a EDF $\mathscr{F}_n(x)$

$$\overline{\mathscr{F}}_n = 1 - \mathscr{F}_n \tag{5.16}$$

is the corresponding Complementary EDF (CEDF). □

The histogram, the EDF and the CEDF are complementary views of the sample distribution. In particular, CEDF plots with logarithmic xy-axes are used regularly in the thesis to identify heavy-tailed distributions and to asses the fit of estimated distributions. The terminology used for CCDF and CEDF plots is to denote as “body” the values of $\overline{\mathscr{F}}_n(x)$ for $x \leq \xi$ and as “tail” the values of $\overline{\mathscr{F}}_n(x)$ for $x > \xi$. The point ξ on the x-axis is in general dictated by the type of data analysis performed.

5.4 Parameter Estimation

Parameter estimation is the second step of the modeling methodology used in this thesis. It is assumed that a distribution family $F_X(x;\theta)$ with a set of unknown parameters θ has been identified as described in Section 5.3. The next step is to define point *estimators* $\hat{\Theta} = \mathscr{E}(X_1, X_2, \ldots, X_n)$. Point *estimates* $\hat{\theta}$ are obtained by replacing the random variables in $\hat{\Theta}$ with observed values.

The optimality of point estimators is decided by concepts such as bias, efficiency, consistency and sufficiency. An unbiased estimator is one for which $E[\hat{\Theta}] = \theta$ [LM86]. Furthermore, an estimator $\hat{\Theta}_1$ is more efficient than an estimator $\hat{\Theta}_2$ if $\text{Var}\left[\hat{\Theta}_1\right] < \text{Var}\left[\hat{\Theta}_2\right]$. By consistency it is meant that a sequence of estimators converges towards the “true” value of the parameter. Sufficiency is concerned with the amount of information intrinsic to the sample, which is lost or kept when a particular estimator is used [MGB74, LM86]. These are large topics outside the scope of this thesis. It is sufficient to mention that maximum likelihood estimators are in general at least as good as other estimators for large sample size. However, the equations that appear in the course of using the method can be non-linear and difficult to solve. In this case numerical solutions are required [MGB74, LM86, MR99]. Similar problems appear when the method is used with mixture distributions. Therefore, a secondary method, denoted *minimum-absolute-error*, is introduced as well. In addition for providing point estimates, the minimum-absolute-error method is used as goodness-of-fit measure as described in Section 5.5.

5.4.1 Maximum Likelihood Method

The Maximum Likelihood (ML) method is based on the concept of *likelihood function*, which is defined as the joint PDF of a number of random variables [MGB74].

Definition 5.8. Given n random variables $X_1, X_2, \ldots, X_n$ drawn from a distribution $F_X(x;\theta)$, and the corresponding observed values $x_1, x_2, \ldots, x_n$, the likelihood function $\mathscr{L}(\theta)$ is defined as

$$\mathscr{L}(\theta) \triangleq f_{X_1,X_2,\ldots,X_n}(x_1, x_2, \ldots, x_n) \tag{5.17}$$

which is the joint distribution of $X_1, X_2, \ldots, X_n$. □

Corollary 5.3. For a random sample $X_1, X_2, \ldots, X_n$ with common distribution $F_X(x;\theta)$

$$\mathscr{L}(\theta) = \prod_{i=1}^{n} f_X(x_i;\theta), \tag{5.18}$$

which follows from the definition of a random sample. □

Intuitively, the likelihood function $\mathscr{L}(\theta)$ for a random sample drawn from a discrete PDF is the probability that the random sample will assume the observed values [MR99]:

$$\mathscr{L}(\theta) = P[X_1 = x_1;\theta]\,P[X_2 = x_2;\theta]\ldots P[X_n = x_n;\theta] \tag{5.19}$$

It becomes evident that the optimal $\hat{\theta}$ is the value that maximizes $\mathscr{L}(\theta)$. This idea can be applied in a similar manner to random samples from a continuous PDF. Assuming certain regularity conditions [MGB74], the solution $\hat{\Theta} = \mathscr{E}(X_1, X_2, \ldots, X_n)$ to the equation

$$\frac{d\mathscr{L}(\theta)}{d\theta} = 0 \tag{5.20}$$

is the ML estimator. When the random variables are replaced with the actual observed values, one obtains the ML estimate $\hat{\theta} = \mathscr{E}(x_1, x_2, \ldots, x_n)$. Sometimes it is easier to solve the equation

$$\frac{d\ln[\mathscr{L}(\theta)]}{d\theta} = 0 \tag{5.21}$$

instead of Equation 5.20 [MGB74].

5.4.2 Minimum-Absolute-Error Method

The minimum-absolute-error method seeks to find an estimate $\hat{\theta}$ that minimizes the difference between the EDF, $\mathscr{F}_n(x)$, and the estimated CDF, $\hat{F}_X(x;\hat{\theta})$, over all x.

Definition 5.9. For a data sample $x_1, x_2, \ldots, x_n$, the difference between $\mathscr{F}_n(x)$ and $\hat{F}_X(x;\hat{\theta})$ is defined as the cumulative absolute error $\varepsilon(\theta)$, such that

$$\varepsilon(\theta) \triangleq \sum_{i=1}^{n} \left|\hat{F}_X(x_i;\hat{\theta}) - \mathscr{F}_n(x_i)\right| \tag{5.22}$$

The estimate $\hat{\theta}$ is the θ-value that minimizes $\varepsilon(\theta)$. □.

Since this method relies on the EDF, $\hat{\theta}$ cannot be solved for analytically. Numerical algorithms to obtain solutions for it are discussed in Section 5.7.

5.5 Goodness-of-Fit

This section describes the final step of the modeling methodology. It is assumed that a candidate distribution, $\hat{F}_X(x;\hat{\theta})$, has been fitted to the random sample $X_1, X_2, \ldots, X_n$ by using the methods presented in Section 5.3 and Section 5.4. The focus here is on quantifying the quality of the fit. In statistics, this usually involves a goodness-of-fit test. The goodness-of-fit test is a hypothesis test [LK00]:

$$H_0: \text{ The random sample } X_1, X_2, \ldots, X_n \text{ is drawn from the distribution } \hat{F}_X(x;\hat{\theta}) \tag{5.23}$$

where H_0 is the null hypothesis. The alternative hypothesis, H_1, is composite, which means that it simply states that H_0 is false without specifying an alternative distribution [DS86]. To test the hypothesis, a test statistic and a *significance level* η are required. The statistic acts on a random sample and selects the outcome of the test. The significance level is the a priori chosen probability for a Type 1 error that is, for rejecting the null hypothesis when it is, in fact, true.

The best known goodness-of-fit tests are Chi-Squared (χ^2), Kolmogorov-Smirnov, and the Anderson-Darling tests. However, for very large sample size these tests generally reject H_0 (Type 1 error) [Ber94, LK00, BCNN01]. Therefore, a different approach is used, one where the hypothesis test is avoided.

Definition 5.10. A goodness-of-fit measure is defined to be a statistic that describes the discrepancy between a fitted distribution and the data used for the fit. Given a random sample $X_1, X_2, \ldots, X_n$, and two candidate distributions, $\hat{F}_X(x;\hat{\theta}_1)$ and $\hat{G}_X(x;\hat{\theta}_2)$, the goodness-of-fit measure provides a discrepancy (error) estimate for each candidate distribution. The distribution having the smaller error estimate is, according to the goodness-of-fit measure, a better fit. □

The cumulative absolute error $\varepsilon(\theta)$ from Definition 5.9 is a good example of a goodness-of-fit measure.

A goodness-of-fit measure called error-percentage measure ($E_{\%}$) was introduced in [EIP05] and used later in [Erm05, IEP06]. The method is based on the Probability Integral Transform (PIT) [LK00, BCNN01]. The PIT method works as follows. Given a *continuous* random variable R with CDF $F_X(x)$, then

$$F_X(R) = P[X \leq R] = Y \tag{5.24}$$

with $P[Y \leq y] \overset{d}{=} U[0,1]$, where $U[0,1]$ denotes the uniform distribution between zero and one[4]. The algorithm to compute $E_{\%}$ is shown below.

Algorithm 1 Calculate Error Percentage

```
Fit a distribution $\hat{F}_X(x;\hat{\theta})$ to the random sample $X_1, X_2, \ldots, X_n$
Obtain the order statistics $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$
Transform the random sample with PIT: $\hat{U}_i = \hat{F}_X(X_{(i)};\hat{\theta}), \quad i = 1, \ldots, n$
$E_{\%} = 100 \frac{\sum_{i=1}^{n} |U_i - \hat{U}_i|}{n E_{max}}$, where $U_i = \frac{i}{n} \overset{d}{=} U[0,1]$
**return** $E_{\%}$
```

If the distribution $\hat{F}$ is a perfect fit, then the PIT transforms the random sample to a uniform distribution, $U[0,1]$. However, since perfect fittings rarely occur in reality, the transformed distribution, $\hat{U}$, only approximates the uniform distribution. The discrepancies between U and $\hat{U}$ are computed and their average is normalized to the highest possible error E_{max} for the distribution $U[0,1]$ where [Erm05],

$$\begin{aligned} E_{max} &= \int_0^1 \sup\{U(x), 1 - U(x)\}\ \mathrm{d}x = \int_0^{1/2} [1 - U(x)]\ \mathrm{d}x + \int_{1/2}^1 U(x)\ \mathrm{d}x \\ &= \left[x - \frac{x^2}{2}\right]_0^{1/2} + \left[\frac{x^2}{2}\right]_{1/2}^1 = \frac{3}{4} \end{aligned} \tag{5.25}$$

[4]The symbol $\overset{d}{=}$ denotes equality in distribution.

Range	$0 \leq E_{\%} < 2$	$2 \leq E_{\%} < 4$	$4 \leq E_{\%} < 6$	$E_{\%} \geq 6$
Quality	Excellent	Good	Acceptable	Poor (unacceptable)

Table 5.2: Quality-of-fit mapping

$E_{\%}$ is expressed in the form of a percentage. The criteria used here to accept a candidate distribution is that $E_{\%} < 6$. This value will be called the *accepted error percentage*. The accepted error percentage was decided experimentally by observing that most distributions that provide a visually acceptable fit in both body and tail have $E_{\%} < 6$. Table 5.2 presents a mapping between various $E_{\%}$ ranges and qualitative statements about the fit.

In essence, the $E_{\%}$-measure is based on the minimum-absolute-error method. To prove this, assume the random variables in the expression for $\hat{U}_i$ are replaced with observed values $x_1, x_2, \ldots, x_n$. It follows that $\hat{U}_i$ becomes an EDF $\mathscr{F}_n(i)$, according to Definition 5.7. Furthermore, $U_i = F_X(i)$, since $U_i \overset{d}{=} U[0,1]$, by definition. The sum used in the expression for $E_{\%}$ becomes therefore

$$\sum_{i=1}^{n} |F_X(i) - \mathscr{F}_n(x_i)| \tag{5.26}$$

which is equivalent to the sum in Equation 5.22.

By finding the $\hat{\theta}$ that minimizes the expression for $E_{\%}$ one can simultaneously obtain the minimum-absolute-error estimate and a goodness-of-fit measure for it. This is the approach used here and it is referred to as the $E_{\%}$-method.

The main disadvantage of the $E_{\%}$-method is that it cannot be used with discrete distributions. The reason is that, according to [DJ50], when the PIT method is applied to a discontinuous distribution, the transformed variable is not uniformly distributed. However, most distributions considered in this thesis are or can be approximated with continuous distributions without sacrificing accuracy.

5.6 Finite Mixture Distributions

Sometimes a single PDF cannot accurately describe the distribution of the random variables of interest and at least two probability distributions must be mixed. Typically, one component of the mixture accounts for the main body of the empirical distribution and a different one describes the behavior in the tail. The crux of the problem becomes to find a way to combine the two distributions in a meaningful way. The method used here is based on finite mixture distributions as described in [TSM85].

Definition 5.11. A mixture distribution $F_X(x)$ with two components has the following density and distribution function, respectively:

$$f_X(x) = \pi g_1(x) + (1-\pi)g_2(x) \tag{5.27}$$

$$F_X(x) = \pi G_1(x) + (1-\pi)G_2(x) \tag{5.28}$$

where $g_1(x)$ and $g_2(x)$ are PDFs, $G_1(x)$ and $G_2(x)$ are CDFs and π is a constant called *mixing weight*. The mixing weight π is selected such that $0 < \pi < 1$ and it decides how much each component is allowed to influence the distribution $F_X(x)$. □

It is assumed that a random sample, $X_1, X_2, \ldots, X_n$ of the data has been collected. The first step is to construct an empirical distribution, $\mathscr{F}_n(x)$, from the random sample.

The next step is to identify a distribution family $G_1(\cdot)$ that matches the body of the data, preferably the tail as well. Then, the parameters of the distribution are estimated, yielding a specific distribution function $\hat{G}_1(x;\hat{\theta}_1)$. $\hat{G}_1(x;\hat{\theta}_1)$ is visually compared to the true distribution to asses the fit in the tail. If the fit is good, then the goodness-of-fit measure $E_{\%}$ is computed as explained in Section 5.5. Otherwise, it is necessary to find the *cutoff point* x_c and the corresponding *cutoff quantile*, q_c, where $\hat{G}_1$ diverges from the true distribution. The probability mass between q_c and 1 is used to identify the distribution family, $\hat{G}_2(\cdot)$ that matches the tail. The parameters of the new distribution must be estimated as well, yielding $\hat{\theta}_2$. Then, a finite mixture distribution is assembled:

$$\hat{F}(x;\hat{\theta}) = \pi\hat{G}_1(x;\hat{\theta}_1) + (1-\pi)\hat{G}_2(x;\hat{\theta}_2) \tag{5.29}$$

where $\pi = q_c$. Since the single distributions, $\hat{G}_1$ and $\hat{G}_2$, are now combined in a finite mixture, the parameters $\hat{\theta}_1$ and $\hat{\theta}_2$ must be recomputed[5]. Their original values may be used as a starting point. An optimal value for π must be computed as well. The parameter set $\hat{\theta}$ in $F(x;\hat{\theta})$ is the set containing the parameters for both distributions and π, i.e., $\hat{\theta} = \{\hat{\theta}_1, \hat{\theta}_2, \pi\}$. Numerical methods for computing the set of optimal parameters θ are presented in Section 5.7.

It is often the case that a mixture distribution (in particular one with only two components) still cannot describe the data accurately enough. This may be improved by increasing the number of components in the mixture distribution at the expense of an increase in the number of parameters. However, a different approach was used here.

Typically, the major discrepancies between the estimated distribution and the true one appear either in the body or in the tail. If, for example, the discrepancies appear in the tail, one can attempt to improve the model accuracy by adjusting the values of the distribution parameters. However, this is likely to decrease the accuracy of the

[5]Recall that G_1 was estimated using the entire probability mass.

model in the body. Similarly, attempts to increase the accuracy in the body may lead to (higher) discrepancies in the tail. Thus, a trade-off is required, accuracy in the body versus accuracy in the tail. Accordingly, a decision must be taken on which part of the distribution (body or tail) is more important to model accurately.

For example, in the case of transfer rates the tail of the distribution models high rates of traffic (bursts) that occur rarely. On the other hand, the body of the distribution models the "average" size of transfer rates. Because of our interest in workload characterization, the distribution tail is deemed more important than the body. This is because bursts of traffic have a greater impact on the workload than the typical traffic. Interarrival times, session durations and packet sizes are all treated in a similar manner. It is the "worst-case" scenario that dictates which part of the distribution is more important.

Although the generation of random variates from mixture distributions is not within the scope of this thesis, a short overview is provided in Appendix B for reference.

5.7 Methodology Review

The goal of this section is to bring together the methods discussed previously into a formal process that can be used to model properties of network traffic. As mentioned before, it is assumed that the property of interest has been measured (sampled) n times. The values $x_1, x_2, \ldots, x_n$ resulting from the n measurements are assumed to be the result of a random sample $X_1, X_2, \ldots, X_n$. The complete process for building the statistical models is presented in Algorithm 2 on page 61.

Step 15–16 may be confusing since no criteria has been provided on how to decide between selecting a different quantile and starting over. Here, the quantile was changed in increments of 0.05 to either sides of the original value. If that did not result in any improvement, the decision was to start over.

The process presented here was implemented using the statistical software package `R` [R D05]. `R` is an interpreted computer language with syntax similar to `S` and `S-PLUS`. The software package contains in addition to the language, a run-time environment with graphics, a debugger and a large library of functions.

As mentioned in Section 5.4.2, the minimum-absolute-error method and indirectly the $E_{\%}$-method rely on numerical optimization for finding a minimum. Also ML estimation requires numerical optimization in many cases where no closed form ML estimators exist. In the thesis the `R`-functions `optimize()` and `optim()` have been used for numerical optimization.

The function `optimize()` performs optimization in one dimension. The underlaying algorithm is, according to the `R` documentation, "...a combination of golden section

Algorithm 2 Methodology for statistical modeling

```
1:  Use EDA visual tools (i.e., histogram, EDF and CEDF plots) to explore the data.
    The summary statistics provide hints about range, skewness and spread
2:  Select a distribution family $G_1$, which appears to provide a good fit
3:  Estimate the unknown parameters $\theta_1$ using ML-estimation to obtain a candidate
    distribution $\hat{G}_{1X}(x, \hat{\theta}_1)$
4:  Compare the plots of $\hat{g}_{1X}(x; \hat{\theta}_1)$, $\hat{G}_{1X}(x; \hat{\theta}_1)$, and $\overline{\hat{G}_{1X}}(x; \hat{\theta}_1)$ to the histogram, EDF,
    and CEDF plots obtained in Step 1
5:  if high visual discrepancy then
6:      Go back to Step 1
7:  end if
8:  Compute $E_{\%}$ for $\hat{G}_{1X}(x; \hat{\theta}_1)$ using $x_1, x_2, \dots, x_n$.
9:  if $E_{\%} < 6$ then
10:     return $E_{\%}$ and $\hat{G}_{1X}(x, \hat{\theta}_1)$
11: end if
12: Identify the cutoff quantile $q_c$
13: Fit a distribution $G_2(\cdot)$ to the probability mass $(1 - q_c)$ as outlined in Step 1–8
14: if $E_{\%} > 6$ then
15:     Either go back to Step 12 and select a different quantile $q_c$ or,
16:     Go back to Step 1. This is equivalent to starting over. Try using a different
        distribution family $G_1(\cdot)$
17: end if
18: Assemble the mixture distribution $F(\cdot) = \pi G_1(\cdot) + (1 - \pi) G_2(\cdot)$
19: Estimate the unknown parameters $\theta = \{\theta_1, \theta_2, \pi\}$ using $E_{\%}$ method. Use the esti-
    mated values from previous steps as initial values
20: if $E_{\%} < 6$ then
21:     return $E_{\%}$ and $F_X(x; \theta)$
22: else
23:     Go back to step 1
24: end if
```

search and successive parabolic interpolation. Convergence is never much slower than that for a Fibonacci search. If 'f' has a continuous second derivative which is positive at the minimum (which is not at 'lower' or 'upper'), then convergence is superlinear, and usually of the order of about 1.324.", and a reference [Bre73] is provided. More information about golden search and parabolic interpolation can be found in [PTVF92].

General purpose multi-dimensional optimization is performed by the `optim()` function. The function has support for several optimization algorithms. In this thesis the default algorithm, Nelder-Mead [PTVF92, LRWE98, MF04], is used primarily. The algorithm does not require any derivatives, being quite stable although not extremely efficient in terms of number of iterations.

When the Nelder-Mead algorithm fails to converge to a solution, the L-BFGS-B [BLNZ94] algorithm is used instead. This algorithm requires a lower and an upper bound for each variable. The thumb rule used to provide the bounds is to allow variables with initial values $m_0 \geq 1$ a range of $0.2m_0$ between the upper and lower bound. For variables with initial values $m_0 < 1$ the range between bounds was $0.1m_0$. This thumb rule was designed empirically and is by no means optimal in any way. In fact, the bounds needed often additional adjustment to obtain convergence.

Chapter 6

Gnutella Traffic Characteristics

Two sets of Gnutella measurements were performed at BTH resulting in two different link-layer packet traces. The first set of measurements was done in the early development phase of the BTH measurement infrastructure (May 2004), and the second set of measurements was performed one year later (July 2005). During the time that passed between the two measurements, the Gnutella system was subject to several changes in terms of extensions to the protocol specification. New features such as dynamic query, download mesh, PFS and UDP file transfers were adopted by a majority of servents, thus profoundly changing the characteristics of the Gnutella network traffic. The older measurement data is therefore less relevant in describing Gnutella traffic characteristics. Consequently, the decision was to abandon it in favor of the new measurement data from July 2005. The statistical analysis presented here was performed exclusively on the new packet trace.

New results concerning traffic characteristics and traffic models for Gnutella signaling traffic as well as insights from the statistical analysis of the data are reported in this chapter. The environment in which the measurements took place and overall trace statistics are described in Section 6.1. The subsequent three sessions describe statistics and models for sessions, messages and IP datagrams.

In order to keep the mathematical formulas brief we use the following conventions. CDFs are denoted by capital letters and PDFs by lower case letters, as in Table 6.1. The parameters are as follows: μ and σ are the mean and standard deviation, while α, β and κ are the shape, scale and location parameters. For the uniform distribution, a and b are the lower and upper boundary, respectively, of the range of x-values for which the distribution is valid. In particular, the parameter a is equivalent to a location parameter, while $(b-a)$ is equivalent of a scale parameter [LK00]. A quick review of the distributions used in the thesis is available in Appendix A. All logarithmic EDF-plots use $\log_{10}$-transformations for both axes.

Uniform	$u_X(x;a,b)$	$U_X(x;a,b)$
Poisson	$\mathrm{po}_X(x;\mu)$	$\mathrm{PO}_X(x;\mu)$
Exponential	$\exp_X(x;\mu)$	$\mathrm{EXP}_X(x;\mu)$
Normal	$\mathrm{n}_X(x;\mu,\sigma)$	$\mathrm{N}_X(x;\mu,\sigma)$
Log-normal	$\ln_X(x;\alpha,\beta)$	$\mathrm{LN}_X(x;\alpha,\beta)$
Pareto	$\mathrm{pa}_X(x;\alpha,\kappa,\beta)$[1]	$\mathrm{PA}_X(x;\alpha,\kappa,\beta)$

Table 6.1: Model notation

6.1 Environment and Packet-Trace Statistics

The results presented here were obtained from an 11-days long link-layer packet trace collected from the BTH network with the methods described in Chapter 5. The `gtk-gnutella` servent at BTH was configured to run as ultrapeer and to maintain 32–40 connections to other ultrapeers and ≈ 100 connections to leaf nodes. The number of connections is a vendor preconfigured value, which is close to the suggested values [Fis03a, SR02]. Although `gtk-gnutella` is capable of operation over UDP, this functionality was turned off. Consequently, the ultrapeer used only TCP for its traffic. No other applications, with the exception of an SSH daemon, were running on the ultrapeer for the duration of the measurements. One SSH connection was used to remotely check on the status of the measurements and the amount of free disk space. The SSH connection was idle for most of the time. The firewall was turned off during the measurements.

The total amount of PCAP data collected with `tcpdump` is approximately 33 GB. The PCAP data generated approximately 45 GB log files. The recorded traffic contains 234 million IP datagrams. The log files show 604 thousand Gnutella sessions that were used to exchange 267 million Gnutella messages. A total of 423 thousand sessions (70%) were unable to perform a successful Gnutella handshake. The main reasons for the unsuccessful handshakes are filled-up connection queues[2] and refusal to accept uncompressed connections. The remaining sessions consist of 181 thousand sessions where both peers used compression, 22 where one of the peers used compression and 10 uncompressed sessions.

[1]This is a Generalized Pareto Distribution. More details are available in Appendix A.

[2]Code 409: "Vendor would exceed 60% of our slots".

6.2 Session Characteristics

A Gnutella session is defined to be the set of Gnutella messages exchanged over a TCP connection between two directly connected peers that have successfully completed the Gnutella handshake. The session lasts until the TCP connection is closed by either FIN or RST TCP segments. The session duration is computed as the time duration between the instant when the first handshake message (CLI_HSK) is recorded (at link layer) until the measured time of the last Gnutella message on the same TCP connection. The session is not considered closed until both sides have sent FIN (or RST) segments.

An incoming session is defined as a session for which the CLI_HSK message was *received* by the ultrapeer at BTH. Outgoing sessions are sessions for which the CLI_HSK message was *sent* by the ultrapeer at BTH. Tables 6.2 and 6.3 show duration (in seconds), number of exchanged messages and bytes for incoming and outgoing sessions, respectively. Table 6.4 shows the same statistics when no distinction is made between incoming and outgoing sessions.

In the column denoted "Samples" the first number shows the number of valid Gnutella sessions that is used to compute the statistics. A Gnutella session is considered valid (in the sense that it is used to compute session statistics) if the Gnutella handshake was completed successfully and at least one Gnutella message was transfered between the two hosts participating in the session. The number in parenthesis is the total number of observed sessions, valid and invalid. In the case of the BTH ultrapeer, only 30% of the all sessions were valid (31.5% and 13.4% when considering only incoming and outgoing sessions, respectively).

The tables show that outgoing sessions transfer about 40 times more data than incoming sessions. Furthermore, by comparing the mean and median values for messages and bytes it can be observed that a few incoming sessions transfer the majority of incoming data. This can be partly accounted for by the hierarchy inherent in Gnutella: ultrapeers are bound to transfer more data than their leaf nodes. In addition, most incoming sessions have very short duration ($<$ 1 second), which can be observed by comparing the mean and median duration values for incoming sessions. This translates in little data being exchanged.

Many of the short connections were terminated with the reception of a BYE mes-

Type	Max	Min	Mean	Median	Stddev	Samples
Duration (s)	767553 (8.9 days)	0.03	517.30	0.86	6780.99	173711 (551168)
Messages	7561532 (7.6M)	4	585.18	11	22580.99	173711 (551168)
Bytes	535336627 (535.3M)	780	53059	1356	2034418	173711 (551168)

Table 6.2: Incoming session statistics

Type	Max	Min	Mean	Median	Stddev	Samples
Duration (s)	470422 (5.4 days)	0.12	3949.86	2459.10	11170.80	7094 (52904)
Messages	2644660 (2.6M)	6	23145.15	15716.50	58627.75	7094 (52904)
Bytes	182279191 (182.3M)	1574	2173564	1457360	4458468	7094 (52904)

Table 6.3: Outgoing session statistics

Type	Max	Min	Mean	Median	Stddev	Samples
Duration (s)	767553 (8.9 days)	0.03	651.98	0.87	7036.85	180805 (604072)
Messages	7561532 (7.6M)	4	1470.34	11	25375.64	180805 (604072)
Bytes	535336627 (535.3M)	780	136258	1357	2219411	180805 (604072)

Table 6.4: Incoming and outgoing session statistics

sage with code 200 `Node Bumped`. A plausible explanation for this is peer bandwidth management. Many Gnutella software packages allow the user to specify several bandwidth limits: for the peer as a whole, for signaling traffic, for downloads and uploads, and for each peer connection. The bandwidth cap can be implemented either by dropping non-essential messages or, in a more aggressive form, by dropping entire sessions (i.e., TCP connections). The hypothesis is that the short sessions are a result of the aggressive form of bandwidth management.

The discussion points out that peers are highly heterogenous in terms of duration and volume of traffic. This assertion is also supported by the standard deviation values shown above. These findings confirm results already presented in [SGG01, SW02, AG04]. Following the taxonomy used in [AG04, BC02], the sessions can be divided into “mice” and “elephants”, that is into sessions carrying small amounts of data and sessions responsible for large volumes of traffic. Sessions can also be classified by their duration into “dragonflies”, that is very short sessions, and “tortoises”, sessions with very long duration. The interpretation of these findings is that in the long run some of the peers will effectively act as “servers”, whereas the remaining peers act as “clients”. The possible consequences of this development were already discussed in Section 2.5: content availability and distribution suffers if the few “servers” available no longer offer their services.

6.2.1 Session Interarrival and Interdeparture Times

The statistics and models for session interarrival and interdeparture times are shown in Table 6.5 and Table 6.6. Interarrival times can be modeled by the lognormal distribution, which is subexponential. In contrast, session interdeparture times require a mixture distribution with a heavy-tailed component (i.e., the Pareto distribution) to pro-

DIR	Max	Min	Mean	Median	Stddev	Samples
IN	1119.01	4.05e-6	5.47	2.20	20.38	173710 (551167)
OUT	5192.62	0.20e-3	133.99	71.78	210.34	7093 (52903)

Table 6.5: Session interarrival and interdeparture times statistics (sec)

DIR	Model	$E_{\%}$
IN	$\mathrm{LN}_X(x; 0.71, 1.08)$	3.0%
OUT	$0.77\,\mathrm{EXP}_X(x; 0.01) + 0.23\,\mathrm{PA}_X(x; 0.7, 0, 132.9)$	3.3%

Table 6.6: Session interarrival and interdeparture times (sec)

vide an acceptable fit. The quality of the fit is shown in the CCDF plots in Figure 6.1.

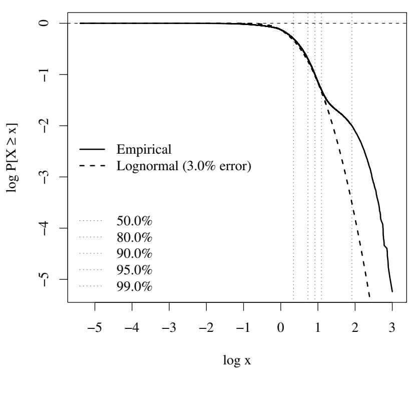
An interesting characteristic was observed when all session interarrival times were considered, that is even those for invalid sessions. It turns out that the set of all interarrival times is exponentially distributed with parameter $\lambda = 0.58$, as shown in Figure 6.2(a). The session arrival rate was analyzed to verify that this is not a measurement error. It is well-known that exponentially distributed interarrival times imply a Poisson arrival rate [Kle75]. As it can be observed from Figure 6.2(b), a Poisson distribution $\mathrm{PO}_X(x; 0.58)$ fits well, at least visually. Unfortunately, no $E_{\%}$ measure can be provided since the method does not work with discrete distributions. However, the EDF of the data should leave little doubt that the data is indeed Poisson distributed. The EDF is plotted without log-scaled axes, since most of the data, 99.9% of the probability mass, is clustered around the values $0, 1, \ldots, 4$.

The same relation does not hold for outgoing traffic which is modeled by a mixture distribution $0.88\,\mathrm{LN}_X(x; -2.32, 1.41) + 0.12\,\mathrm{EXP}_X(x; 0.008)$ with 1.1% error, as observed in Figure 6.3.

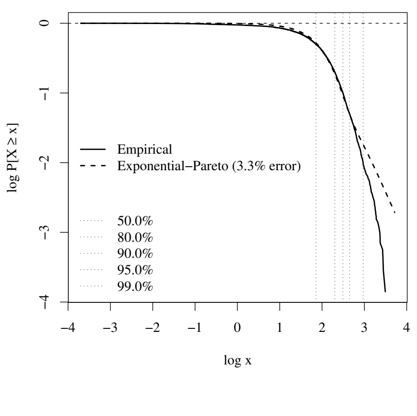
Gnutella sessions are created when servents establish a number of peer connections, which guarantees enough connectivity to find desired resources with high probability. For example, the BTH ultrapeer maintains on average 130–140 peer connections. If the number of connections drops below a specific threshold (dependent on the Gnutella implementation), the servent attempts to open more connections. Although the user can manually open connections to a specific peer, this rarely happens. Thus, one can

Statistic	Model	$E_{\%}$
Interarrival times (sec)	$\mathrm{EXP}_X(x; 0.58)$	1.7%
Rate (session/second)	$\mathrm{PO}_X(x; 0.58)$	N/A

Table 6.7: Gnutella (valid and invalid) session interarrival times



(a) Incoming



(b) Outgoing

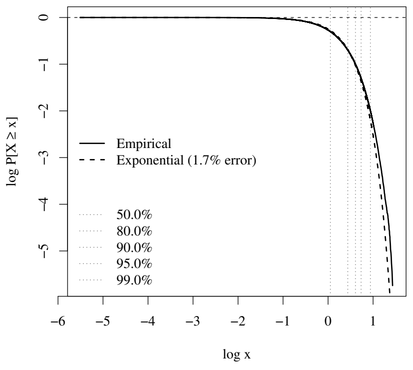
Figure 6.1: Gnutella session interarrival and interdeparture times (sec)

rule out human intervention and proceeding by assuming session initiations are fully automatic.

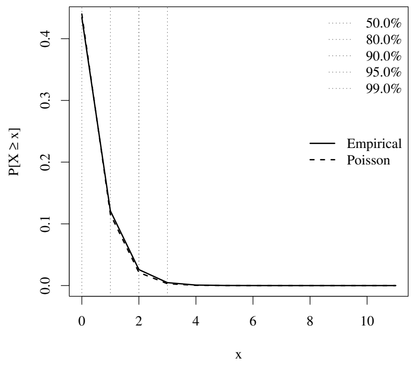
A possible explanation for the appearance of the exponential distribution is as follows. A session initiation is defined by the arrival of a CLI_HSK message, which is the first part of the Gnutella three-way handshake. Consequently, the model for incoming sessions describes the arrival of a mixture of CLI_HSK messages from different sources. Following the arguments from [SW86, CR02, CCLS03], a model where the arrivals are generated by a number of point processes is assumed. Then according to [CM65, CR02], the superposition of the arrival point processes will converge, under some mild assumptions, to a Poisson distribution as the number of sources increases.

6.2.2 Session Size and Duration

The session size and duration models are reported in Table 6.8. It should be noted that the session duration statistic has a very complex CCDF, which cannot be modeled with only two distributions. This is the only model reported here that uses a mixture of three distributions. Alternatively, the upper 5% of the tail can be modeled with a Pareto distribution.



(a) Interarrival times (sec)



(b) Incoming session rate (sessions/sec)

Figure 6.2: Gnutella (valid and invalid) session interarrival times and incoming session rate

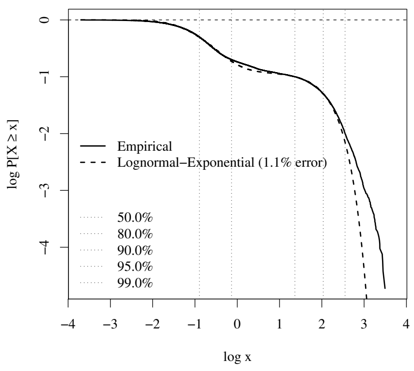


Figure 6.3: Gnutella (valid and invalid) session interdeparture times (sec)

Statistic	Model	$E_{\%}$
Session size (bytes)	$0.69\,\mathrm{N}_X(x;1356,5.9)+0.31\,\mathrm{LN}_X(x;9.0,3.17)$	4.7%
Session duration (sec)	$0.57\,\mathrm{N}_X(x;0.85,0.07)+$ $0.33\,\mathrm{LN}_X(x;0.37,0.96)+$ $0.10\,\mathrm{U}_X(x;18.45,2460)$	2.3%
Session duration, upper 5% (sec)	$\mathrm{PA}_X(1.1,1800,1870.4$	2.4%

Table 6.8: Session size and duration models

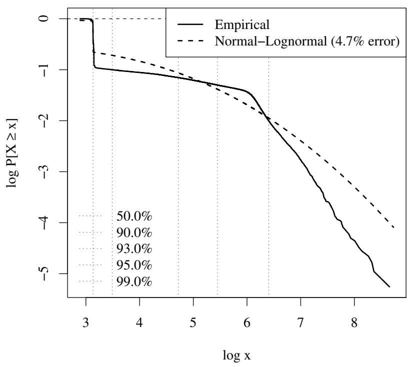
Most of the observed session sizes (64.8%) lie in the range 1300–1400 bytes, and out of the remaining 35.2%, 9.6% are smaller than 1300 bytes and 25.6% are larger than 1400 bytes.

6.3 Message Characteristics

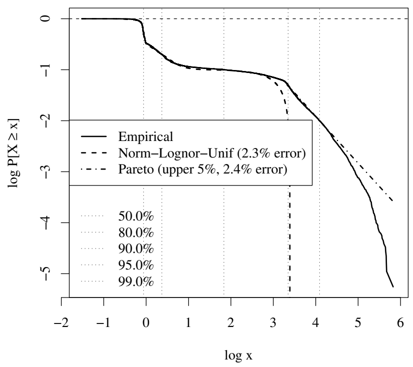
In this section message statistics are reported for each Gnutella message type. The type UNKNOWN denotes messages with a valid Gnutella header, but with unrecognized message type. These messages are either experimental or corrupted. The type ALL is used for statistics computed over all messages, irrespective of type. Only models for aggregated message streams (i.e., message type ALL) are presented

Table 6.9 shows interarrival times for messages received by the BTH ultrapeer and Table 6.10 shows interdeparture times for messages sent by the BTH ultrapeer. Although the PCAP timestamps have microsecond resolution [Tor05], the times presented have only 100 μs precision. This is due to memory limitations in the postprocessing software.

Summing over the number of samples for each message type does not add up to the value shown in the number of samples for message type ALL. The reason is the analysis software which ignores messages that generate negative interarrival/interdeparture times. Negative times appear because the application flow reassembly handles several (typically more than a hundred) connections at the same time. On each connection the timestamp for arriving packets is monotonically increasing. However, the interarrival/interdeparture statistics presented here are computed across all connections. To ensure monotonically increasing timestamps even in this case, new messages from arbitrary connections are stored in a buffer, where they are sorted by timestamp. The size of the buffer is limited to 500,000 entries due to memory management issues. By summing the entries in the "Mean" column in Table 6.17 it can be observed that, on average, there are 280 incoming and outgoing messages per second. This means that the buffer can store about 30 minutes of average traffic and much less during traffic bursts. If there are delayed messages due to TCP retransmissions or other events, they



(a) Session size (bytes)



(b) Session duration (sec)

Figure 6.4: Gnutella session size and duration

reach the buffer too late and are discarded.

The large interarrival and interdeparture times in handshake messages (CLI_HSK, SER_HSK, FIN_HSK) in Table 6.9 and Table 6.10 occur because once a servent reaches the preset amount of connections, it no longer accepts or attempts to open new connections until one or more of the existing connections is closed. This behavior also explains the large interarrival and interdeparture times for BYE messages.

It is interesting to see that interarrival times are exponentially distributed. An argument similar to the one in Section 6.2.1 can be used to explain the cause for it. However, the arrival process is not a pure Poisson process, but rather a compound Poisson process [Ros92, CM65, Kle75] since simultaneous message arrivals do occur. To see why that happens, recall that before the messages can be extracted from the TCP flows, those flows pass through a decompression layer. Typically, a single TCP segment carries several Gnutella messages. All of them receive the same timestamp, since they traveled in bulk all the way from the source. Models for the bulk-size distributions are provided in Table 6.14 and Table 6.15

Message interdeparture times have an interesting distribution. As it can be observed in Table 6.12, approximately 73.9% of the probability mass is clustered around the values 0.0001–0.0005. The remaining 26.1% of the probability mass can be modeled

Type	Max	Min	Mean	Median	Stddev	Samples
CLI_HSK	28.4591	0.0001	1.7246	1.1256	1.8644	551148
SER_HSK	5185.0490	0.0001	19.6294	0.2090	92.1849	48432
FIN_HSK	1118.9920	0.0001	5.3165	2.1942	19.4800	178783
PING	13.5871	0.0001	0.2762	0.1931	0.2726	3457169
PONG	2.2624	0.0001	0.1404	0.0979	0.1383	9086918
QUERY	1.4514	0.0001	0.0343	0.0240	0.0340	59010007
QUERY_HIT	19.2778	0.0001	0.1842	0.0976	0.2661	6932327
QRP	50.0632	0.0001	2.0475	1.0534	2.8707	478451
HSEP	1780.4420	0.0003	6.1560	4.3834	8.4758	154742
PUSH	40.1396	0.0001	0.0677	0.0405	0.1157	24934450
BYE	1119.5930	0.0001	5.9160	2.3591	22.3494	160695
VENDOR	30.8037	0.0001	0.4346	0.2207	0.5993	9669915
UNKNOWN	51576.8600	3.0680	2075.3190	6.9379	9298.3600	35
ALL	9.8299	0.0001	0.02436	0.0169	0.0243	114663084

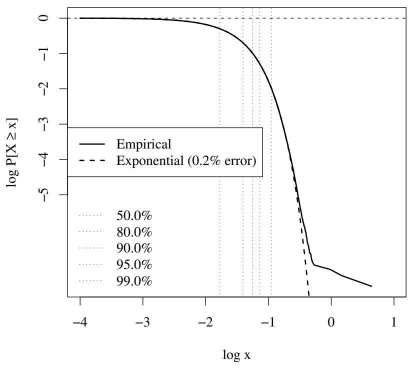
Table 6.9: Message interarrival time statistics (sec)

Type	Max	Min	Mean	Median	Stddev	Samples
CLI_HSK	5189.2340	0.0002	17.9655	0.1273	88.8506	52902
SER_HSK	28.4595	0.0003	1.7298	1.1287	1.8712	549456
FIN_HSK	5185.5150	0.0006	28.4784	0.3305	110.2372	33373
PING	20.5910	0.0001	1.3773	0.5077	2.1342	694550
PONG	2.7215	0.0001	0.1573	0.1012	0.1682	34639367
QUERY	12.1151	0.0001	0.0295	0.0003	0.0541	70066326
QUERY_HIT	19.2818	0.0001	0.2188	0.1285	0.2885	6309719
QRP	603.3599	0.0001	2.6350	0.0004	19.8572	680103
HSEP	358.3067	0.0001	2.5020	1.4089	5.8293	384084
PUSH	76.5303	0.0001	0.0429	0.0003	0.1713	38105019
BYE	3849.4550	0.0001	134.8121	77.2090	187.7784	7033
VENDOR	64.6689	0.0001	1.8253	1.1124	2.4838	525269
UNKNOWN	N/A	N/A	N/A	N/A	N/A	1
ALL	1.5450	0.0001	0.0178	0.0003	0.0353	152047214

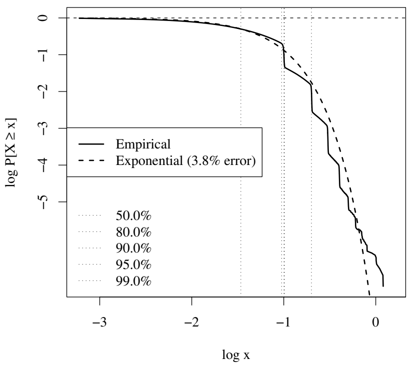
Table 6.10: Message interdeparture time statistics (sec)

DIR	Message	Model	$E_{\%}$
IN	`ALL`	$\mathrm{EXP}_X(x; 40.96)$	0.16%
OUT	`ALL`	$0.261\,\mathrm{EXP}_X(x; 20.23)$ (see Table 6.12 for the body)	3.8%

Table 6.11: Models for message interarrival and interdeparture times (sec)



(a) Message interarrival times



(b) Message interdeparture times, upper 26.1%

Figure 6.5: Message interarrival and interdeparture times (sec)

Interdeparture times	0.0001	0.0002	0.0003	0.0004	0.0005
Probability	0.024	0.515	0.155	0.033	0.012

Table 6.12: Probability mass points for message interdeparture times (sec)

Type	Max	Min	Mean	Median	Stddev	Samples
CLI_HSK	696	22	336.91	328	65.69	604072
SER_HSK	2835	23	386.83	369	145.69	597896
FIN_HSK	505	23	107.92	76	88.55	212162
PING	34	23	25.48	23	3.88	4151799
PONG	464	37	74.96	61	38.68	43727188
QUERY	376	26	70.17	55	46.40	129078986
QUERY_HIT	39161	58	590.28	358	1223.58	13242329
QRP	4124	29	608.60	540	596.70	1158596
HSEP	191	47	70.39	71	28.15	538834
PUSH	49	49	49.00	49	0.00	63040718
BYE	148	35	40.02	37	15.84	167726
VENDOR	177	31	36.45	33	19.51	10195389
UNKNOWN	43	23	23.53	23	3.24	38
ALL	39161	22	93.45	49	303.26	266715733

Table 6.13: Message size statistics (bytes)

by a exponential distribution ($\lambda = 20.23$) with 3.8% error.

Table 6.13 shows the message size statistics for each Gnutella message type. In contrast to the other tables, messages are not classified by direction (incoming or outgoing). The rationale is that message size is independent of message direction. It can be observed that, on average, QUERY_HIT and QRP messages have the largest size. They are tightly followed by handshake messages, where the capability headers account for most of the data. It is interesting to notice that the maximum size of QUERY_HIT messages is 39 KB, which is an order of magnitude larger than the 4 KB specified by [KM02].

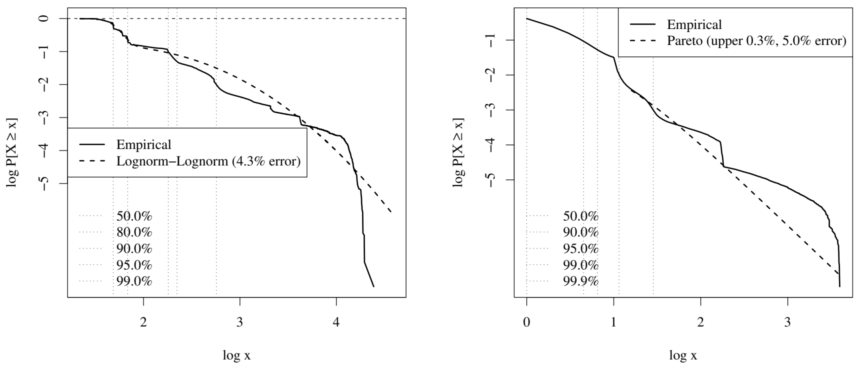
The model for the message bulk size is presented in Table 6.14 and Table 6.15. Bulks of size 1–15 use 99.7% of the probability mass. The remaining 0.3% of the probability mass is modeled with a Pareto distribution.

DIR	Message	Model	$E_{\%}$
IN/OUT	`ALL`	$0.81\,\mathrm{LN}_X(x;3.94,0.23)+0.19\,\mathrm{LN}_X(x;5.14,1.24)$	4.3%
IN/OUT	Bulk size	$0.003\,\mathrm{PA}_X(x;0.42,15,9.6)$	5.0%

Table 6.14: Message size (bytes) and bulk size distribution

Bulk size	1	2	3	4	5
Probability	0.586	0.173	0.082	0.049	0.031
Bulk size	6	7	8	9	10
Probability	0.020	0.012	0.008	0.005	0.003
Bulk size	11	12	13	14	15
Probability	0.019	0.005	0.002	0.001	0.001

Table 6.15: Probability mass points for message bulk size



(a) Message size distribution

(b) Message bulk size distribution (upper 0.3%)

Figure 6.6: Gnutella message size (bytes) and bulk distribution

Type	Max	Min	Mean	Median	Stddev	Samples
CLI_HSK	349.3015	0	0.0308	0	1.0412	604072
SER_HSK	52.2645	0	0.0032	0	0.1350	597896
FIN_HSK	68.6295	0	0.0057	0	0.2838	212162
PING	251.2914	0	0.0273	0	0.6309	4151799
PONG	2355.8650	0	0.0077	0	0.5881	43727188
QUERY	2355.8650	0	0.0035	0	1.3271	129078986
QUERY_HIT	480.8159	0	0.0243	0	1.0260	13242329
QRP	753.1904	0	0.1883	0	1.6019	1158596
HSEP	74.0482	0	0.0017	0	0.2186	538834
PUSH	135.5155	0	0.0023	0	0.2017	63040718
BYE	148.7292	0	0.0386	0	0.5194	167726
VENDOR	391.3439	0	0.0117	0	0.2451	10195389
UNKNOWN	1.0418	0	0.2995	0	0.4294	38
ALL	2355.8650	0	0.0065	0	0.9968	266715733

Table 6.16: Message duration statistics (seconds)

The message duration statistic can be useful to infer waiting times at application layer, when a message is divided across two or more TCP segments. The statistic is defined as the time difference between the first and last TCP segments that were used to transport the message. When a message uses only one TCP segment the time duration for that specific message is zero.

From the median column in Table 6.16 it can be observed that at least 50% of the messages require just one TCP segment. The PONG and QUERY_HIT message rows contain extreme values for the maximum duration, 2355.9 seconds ($\approx$ 39 minutes). These values are most likely the result of malfunctioning or experimental Gnutella servents.

6.4 Transfer Rate Characteristics

This section reports on transfer rates in bytes/second and in messages/second for each Gnutella message types. All statistics are computed over 950,568 samples. The number of samples is equal to the time duration ($\approx$ 11 days) in seconds for the available measurement data. Models are reported only for aggregate message flows (i.e., type ALL messages). As it can be observed in Table 6.19 both incoming and outgoing transfer rates are heavy-tailed. In terms of specific message types, QUERY and QUERY_HIT messages dominate incoming and outgoing streams, both in terms of average message rate as well as average byte rates. This is to be expected since the GNet is used primarily for searching for files.

DIR	Max	Min	Mean	Median	Stddev
IN	6471	0	120.63	111	84
OUT	4164	0	159.96	153	61

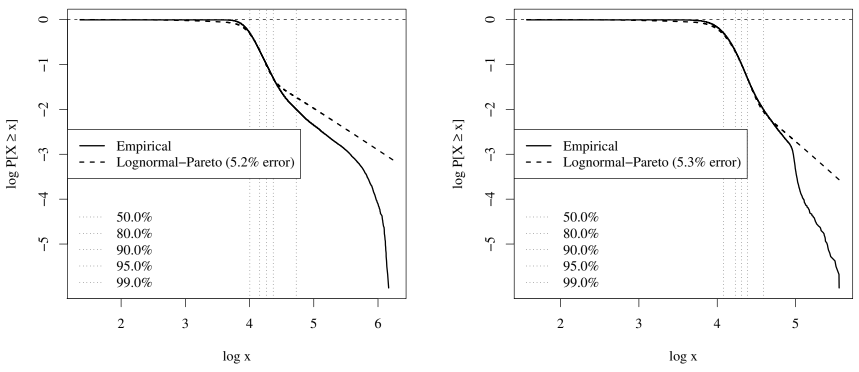
Table 6.17: Gnutella (ALL) message rate (msg/sec) statistics

DIR	Max	Min	Mean	Median	Stddev
IN	1745341	0	12883	10113	24287
OUT	370825	0	13338	12062	7624

Table 6.18: Gnutella (ALL) byte rate (bytes/sec) statistics

DIR	Model	$E_{\%}$
IN	$0.76\,\mathrm{LN}_X(x;9.26,0.37)+0.23\,\mathrm{PA}_X(x;1.06,0,4003)$	5.2%
OUT	$0.81\,\mathrm{LN}_X(x;9.43,0.39)+0.19\,\mathrm{PA}_X(x;0.63,0,3704)$	5.3%

Table 6.19: Gnutella (ALL) byte rate (bytes/sec) modeling results



(a) Incoming (b) Outgoing

Figure 6.7: Gnutella (ALL) byte rates (bytes/sec) models

Type	DIR	Max	Min	Mean	Median	Stddev
CLI_HSK	IN	12	0	0.58	0	0.79
CLI_HSK	OUT	30	0	0.06	0	0.56
SER_HSK	IN	20	0	0.05	0	0.48
SER_HSK	OUT	12	0	0.58	0	0.79
FIN_HSK	IN	9	0	0.19	0	0.46
FIN_HSK	OUT	18	0	0.04	0	0.34
PING	IN	72	0	3.64	3	1.94
PING	OUT	17	0	0.73	0	1.56
PONG	IN	130	0	9.56	9	4.33
PONG	OUT	433	0	36.44	36	19.12
QUERY	IN	347	0	62.08	60	19.64
QUERY	OUT	875	0	73.71	69	34.08
QUERY_HIT	IN	531	0	7.29	5	9.82
QUERY_HIT	OUT	272	0	6.64	5	7.39
QRP	IN	45	0	0.50	0	0.98
QRP	OUT	283	0	0.72	0	7.18
HSEP	IN	20	0	0.16	0	0.41
HSEP	OUT	23	0	0.40	0	0.68
PUSH	IN	1068	0	26.23	23	19.34
PUSH	OUT	4091	0	40.09	32	37.32
BYE	IN	40	0	0.17	0	0.43
BYE	OUT	118	0	0.01	0	0.15
VENDOR	IN	6385	0	10.17	1	76.17
VENDOR	OUT	24	0	0.55	0	0.80
UNKNOWN	IN	1	0	0.00	0	0.01
UNKNOWN	OUT	1	0	0.00	0	0.00

Table 6.20: Message rate (msg/sec) statistics

Type	DIR	Max	Min	Mean	Median	Stddev
CLI_HSK	IN	4126	0	187	0	258
CLI_HSK	OUT	14519	0	27	0	273
SER_HSK	IN	12507	0	31	0	289
SER_HSK	OUT	4001	0	212	0	306
FIN_HSK	IN	982	0	15	0	42
FIN_HSK	OUT	4474	0	9	0	94
PING	IN	1665	0	92	92	50
PING	OUT	503	0	19	0	45
PONG	IN	17043	0	1213	1173	541
PONG	OUT	26050	0	2235	2162	1179
QUERY	IN	24101	0	4441	4317	1426
QUERY	OUT	46424	0	5088	4702	2511
QUERY_HIT	IN	1736791	0	4868	1912	23917
QUERY_HIT	OUT	360235	0	3355	1837	5229
QRP	IN	47340	0	389	0	1408
QRP	OUT	152820	0	353	0	3660
HSEP	IN	940	0	8	0	21
HSEP	OUT	2185	0	32	0	58
PUSH	IN	52332	0	1285	1127	948
PUSH	OUT	200459	0	1964	1568	1829
BYE	IN	1720	0	6	0	16
BYE	OUT	4956	0	1	0	11
VENDOR	IN	210702	0	347	33	2514
VENDOR	OUT	2197	0	44	0	81
UNKNOWN	IN	23	0	0	0	0.1
UNKNOWN	OUT	43	0	0	0	0.1

Table 6.21: Message byte rate (bytes/sec) statistics

6.5 Traffic Characteristics at IP Layer

Table 6.22 provides the summary statistics for the IP byte rates. It is interesting to note that the mean and median IP byte rates are very similar to the corresponding statistics for Gnutella byte rates shown in Table 6.18. These values alone indicate that the compression of Gnutella messages does not yield large gains. However, if one takes into consideration the maximum and standard deviation values it can be observed that the compression removes much of the burstiness from the application layer, leading to smoother traffic patterns. This effect is visible if one compares Figure 6.9 to Figure 6.10, especially in the plots for incoming traffic.

In Table 6.22 it can be observed that the incoming and outgoing IP byte rates are quite similar. The statistical models shown in Table 6.23 are further evidence to that. It should be observed that a heavy-tailed component is present in each model. The tail of the incoming byte rate CEDF displayed in Figure 6.8 shows clear signs of decay by power law. Indeed, the Pareto distribution provides a good fit for the upper 30% of the CEDF, as shown in Figure 6.8(b).

DIR	Max	Min	Mean	Median	Stddev
IN	249522	0	11536	10961	4075
OUT	176986	0	12668	12037	5722

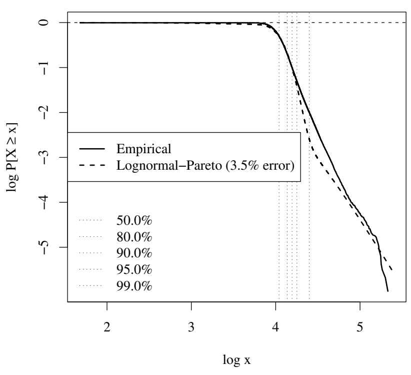
Table 6.22: IP layer byte rate (bytes/sec) statistics

DIR	Model	$E_{\%}$
IN	$0.89\,\mathrm{LN}_X(x;9.33,0.26)+0.11\,\mathrm{PA}_X(x;0.32,0,2774)$	3.5%
IN (upper 30%)	$\mathrm{PA}_X(x;0.27,12812,2180)$	2.2%
OUT	$0.86\,\mathrm{LN}_X(x;9.45,0.27)+0.14\,\mathrm{PA}_X(x;0.87,0,1662)$	2.3%

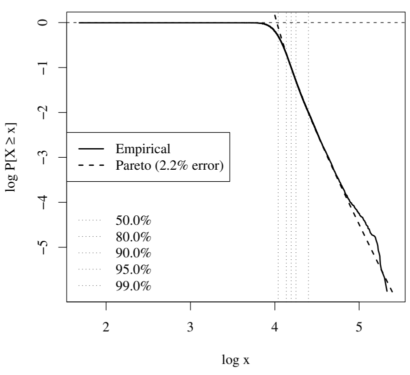
Table 6.23: Models for IP layer byte rates (bytes/sec)

The upper 52% of the IP datagram size distribution can be modeled by a Pareto distribution defined in Table 6.24. The CCDF plot for it is shown in Figure 6.11. Probabilities for the datagram sizes corresponding to the lower 48% of the distribution are shown in Table 6.25. It can be observed that 46.7% of the probability mass is accounted for by IP datagrams with size 40 bytes and 52 bytes, respectively. The 40-bytes datagrams correspond to TCP segments with no data and no options.

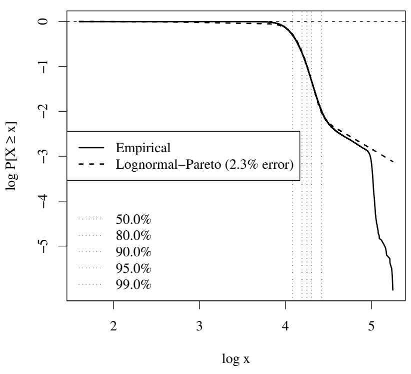
The interarrival and interdeparture times statistics are displayed in Table 6.26. It is interesting to note that interarrival times for IP datagrams follow an exponential distribution. This is similar to the case of interarrival times for valid and invalid sessions as well as to the case of interarrival times for Gnutella messages (type ALL). Just as



(a) Incoming

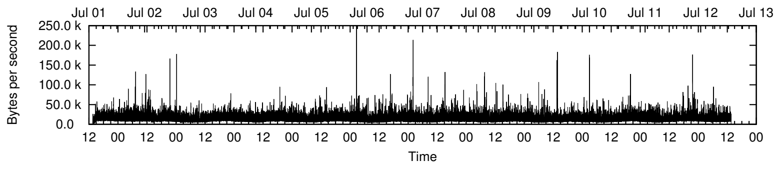


(b) Incoming (upper 30%)

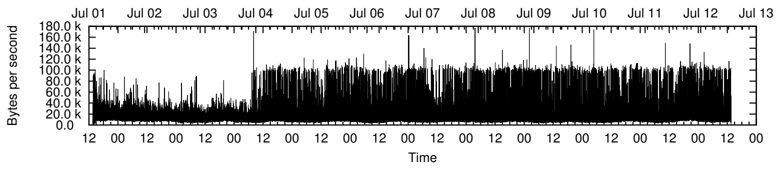


(c) Outgoing

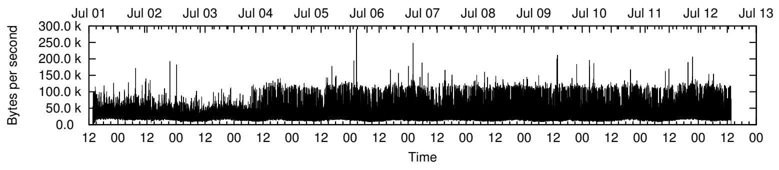
Figure 6.8: IP layer byte rates (bytes/sec)



(a) Incoming IP byte rate

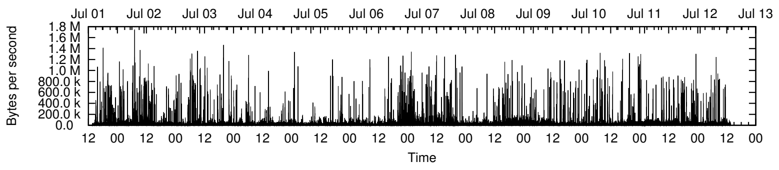


(b) Outgoing IP byte rate

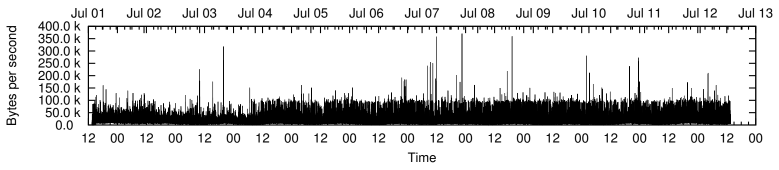


(c) Incoming and Outgoing IP byte rate

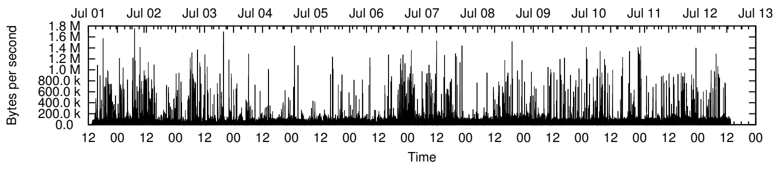
Figure 6.9: Gnutella transfer rates at IP layer (bytes/sec)



(a) Incoming Gnutella byte rate



(b) Outgoing Gnutella byte rate



(c) Incoming and outgoing Gnutella byte rate

Figure 6.10: Gnutella transfer rates at application layer (bytes/sec)

DIR	Model	$E_{\%}$
IN/OUT	$0.52\,\mathrm{PA}_X(x; 0.53, 60, 50.55)$	3.3%

Table 6.24: Model for IP datagram size (bytes)

Datagram size	0–39	40	41	42	43	44
Probability	0.000	0.209	0.000	0.001	0.000	0.000
Datagram size	45	46	47	48	49	50
Probability	0.000	0.000	0.008	0.006	0.000	0.000
Datagram size	51	52	53	54	55	
Probability	0.000	0.258	0.003	0.001	0.000	

Table 6.25: Probability mass points for IP datagram size (bytes)

in those cases, it is conjectured that superposition of point processes is responsible for this phenomenon. The CCDF plots for interarrival and interdeparture times are shown in Figure 6.12.

DIR	Max	Min	Mean	Median	Stddev
IN	11.36	3e-6	8.85e-3	5.75e-3	9.72e-3
OUT	22.05	7e-6	8.24e-3	0.49e-3	14.22e-3

Table 6.26: IP datagram interarrival and interdeparture times statistics (sec)

DIR	Message	Model	$E_{\%}$
IN	IP datagrams	$\mathrm{EXP}_X(x; 118.76)$	0.8%
OUT	IP datagrams	$0.5\,\mathrm{LN}_X(x; -8.45, 0.26) + 0.5\,\mathrm{EXP}_X(x; 61.29)$	1.2%

Table 6.27: Interarrival and interdeparture times models for IP datagrams (sec)

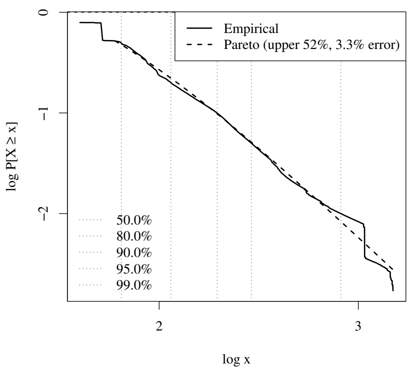
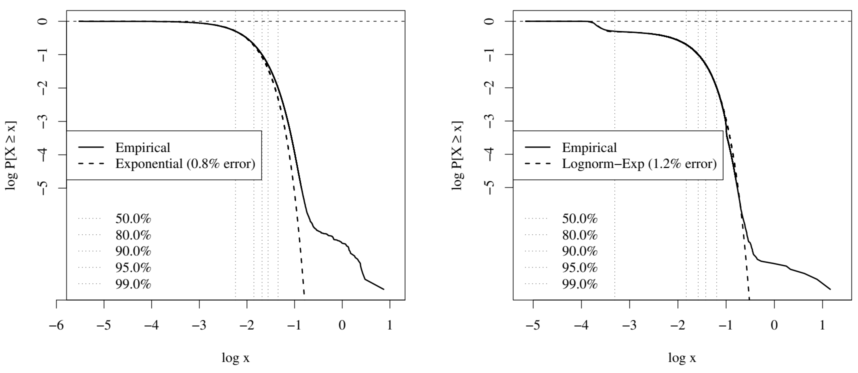


Figure 6.11: IP datagram size (bytes)



(a) Interarrival times

(b) Interdeparture times

Figure 6.12: IP datagram interarrival and interdeparture times (sec)

Chapter 7

Conclusions and Future Work

Increasingly wide availability of low-priced computer hardware and software is enabling more and more advanced user services. In particular, multimedia distribution services are emerging as the new killer applications. Non-interactive multimedia services such as software, music and video distribution place high demands on the amount of bandwidth required for data transport. Interactive multimedia services as for example Internet telephony, video conferencing and live TV, have additional constraints in terms of packet loss and delay jitter.

The accumulation of computing resources at the edge of the network paved the way for new ideas related to content search and distribution. Napster's demise marked a paradigm shift from services based on the client-server architecture to new P2P services. Contrary to services based on the client-server architecture, P2P services are distributed across all nodes in the overlay, rather than being restricted to a few server nodes.

All these factors need to be taken into consideration in the design of future networks with support for traffic engineering and QoS. For existing networks, it becomes imperative to understand how they are affected by the new services in order to achieve efficient network management.

The work presented here is focused on characteristics and statistical models for Gnutella network traffic. The emphasis for the characteristics has been on accuracy and detail, while for the traffic models the emphasis has been on analytical tractability and ease of simulation. To the author's best knowledge this is the first work on Gnutella that presents statistics down to message level.

Another important contributions of the thesis is the design and implementation of a modular measurement infrastructure for P2P traffic. Using the measurement infrastructure, Gnutella traffic from the BTH network was recorded as a link-layer packet trace. Decoding and analysis of the packet trace yielded statistical characteristics and models for sessions, flows and messages, which are the major contributions of this thesis.

Although the packet trace provided much insight in the patterns of Gnutella network traffic, there are several other questions that warrant further investigation. Foremost, analysis of the degree of long-range dependence is necessary in order to determine the amount of correlation that occurs in the traffic. This issue is relevant in the light of results presented in [CR02, CCLS03], which indicate that certain types of traffic aggregations tend towards a Poisson distribution when the number of sources increases. A Gnutella ultrapeer can be viewed as an application layer router that aggregates traffic flows from many peers. If the Poisson assumptions mentioned previously hold, then the theory predicts that statistical multiplexing gains will occur. It would be very interesting to measure the extent to which that happens, if any.

The models developed here are going to be used to generate synthetic traffic in a simulator. The idea is to tweak some of the Gnutella messages to carry QoS information about peer connections (e.g., throughput, RTT, packet loss). Thus, a QoS overlay network will be built on top of Gnutella. Then, the overlay network can be used to perform QoS routing, which would enable high-quality interactive multimedia services. The real challenge is to ensure that the QoS routing runs smoothly, with little overhead compared to the payload data.

Appendix A

Probability Distributions

This is a very short review of the PDFs and CDFs for distributions used in this thesis. The review is based on information presented in [LK00, Gha05, JKB94].

A.1 Uniform Distribution, U[a,b]

$$F_X(x;a,b) = \begin{cases} 0 & \text{if } x < a \\ \dfrac{x-a}{b-a} & \text{if } a \le x \le b \\ 1 & \text{if } b < x \end{cases} \tag{A.1}$$

$$f_X(x;a,b) = \begin{cases} \dfrac{1}{b-a} & \text{if } a \le x \le b \\ 0 & \text{otherwise} \end{cases} \tag{A.2}$$

The special case, $U[0,1]$ is equivalent to

$$F_X(x;0,1) = \begin{cases} 0 & \text{if } x < a \\ x & \text{if } 0 \le x \le 1 \\ 1 & \text{if } 1 < x \end{cases} \tag{A.3}$$

$$f_X(x;0,1) = \begin{cases} 1 & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{A.4}$$

A.2 Poisson Distribution, $\mathrm{PO}[\lambda]$

$$F_X(x;\lambda) = \begin{cases} 0 & \text{if } x < 0 \\ e^{-\lambda} \sum_{i=0}^{[x]} \dfrac{\lambda^i}{i!} & \text{otherwise} \end{cases} \tag{A.5}$$

where $[x]$ is the largest integer such that $[x] \leq x$.

$$f_X(x;\lambda) = \begin{cases} \dfrac{e^{-\lambda}\lambda^x}{x!} & \text{if } x \in \mathbb{N} \\ 0 & \text{otherwise} \end{cases} \tag{A.6}$$

A.3 Exponential Distribution, $\mathrm{EXP}[\lambda]$

$$F_X(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & \text{if } 0 \leq x \\ 0 & \text{otherwise} \end{cases} \tag{A.7}$$

$$f_X(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{if } 0 \leq x \\ 0 & \text{otherwise} \end{cases} \tag{A.8}$$

A.4 Normal Distribution, $\mathrm{N}[\mu,\sigma^2]$

$$f_X(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2} \right) \quad \text{for all } x \in \mathbb{R} \tag{A.9}$$

There is no closed form available for $F_X(x;\mu,\sigma^2)$. The values must be estimated numerically.

A.5 Lognormal Distribution, $\mathrm{LN}[\mu,\sigma^2]$

$$f_X(x;\mu,\sigma^2) = \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(\ln[x]-\mu)^2}{2\sigma^2} \right) \quad \text{for all } x \in \mathbb{R} \tag{A.10}$$

There is no closed form available for $F_X(x;\mu,\sigma^2)$. The values must be estimated numerically.

A.6 Classical and Generalized Pareto Distributions

The classical Pareto distribution, used for example in [PF95], is defined as

$$F_X(x; a, \kappa) = 1 - \left(\frac{\kappa}{x}\right)^a \tag{A.11}$$

where a is the shape parameter and κ the location parameter. This is called a Pareto distribution of the first kind [JKB94]. The corresponding probability density function is

$$f_X(x; a, \kappa) = a\kappa^a x^{-a-1} \tag{A.12}$$

The symbol a is used instead of α to avoid confusion with the shape parameter in the generalized Pareto distribution.

A generalized Pareto distribution [Col01] is defined as

$$F_X(x; \alpha, \kappa, \beta) = 1 - \left[1 + \frac{\alpha(x - \kappa)}{\beta}\right]^{-\frac{1}{\alpha}} \tag{A.13}$$

where α is the shape parameter, κ is the location parameter, and β is the scale parameter. The corresponding density function is

$$f_X(x; \alpha, \kappa, \beta) = \frac{1}{\alpha}\left[1 + \frac{\alpha(x - \kappa)}{\beta}\right]^{-\frac{1}{\alpha}-1} \tag{A.14}$$

Clearly, for $\beta = \alpha\kappa$ and $\alpha = 1/a$, the generalized Pareto distribution is equivalent to the classical Pareto distribution.

Appendix B

Random Variate Generation from Finite Mixture Distributions

Using the PIT method from Section 5.5 one can easily generate random variates provided the analytical expression for a $F_X^{-1}(x)$ can be obtained. If that is not possible, numerical methods are available in many cases.

In the case of finite mixture distributions with two components,

$$F_X(x) = \pi G_1(x) + (1-\pi)G_2(x), \tag{B.1}$$

if $G_1^{-1}(x)$ and $G_2^{-1}(x)$ are more easily obtained than $F_X^{-1}(x)$, then [LK00] offers a different method to obtain random variates. The method requires the mixture combination to be convex. According to [Rud87] "A set E in a vector space V is said to be *convex* if it has the following geometric property: Whenever $x \in E$, $y \in E$, and $0 < t < 1$, the point

$$z_t = (1-t)x + ty \tag{B.2}$$

also lies in E.". A convex combination of k points x_i $i = 1, \ldots, k$ has the form shown below [BV04]:

$$\sum_{i=1}^{k} \theta_i x_i \quad \text{with} \quad \sum_{i=1}^{k} \theta_i = 1 \tag{B.3}$$

Given the way in which the finite mixture distributions $F_X(x)$ are built in this thesis (see Section 5.6) and the assumption that $G_1(x)$ and $G_2(x)$ are well-behaved distributions, then we can assume that $F_X(x)$ is a convex combination (i.e., $F_X(x)$ is a convex set).

To generate variates from the convex mixture a variable $R_1 \sim U[0,1]$ is required to select the component distribution: if $R_1 \leq \pi$ select $G_1(x)$, otherwise select $G_2(x)$. A secondary variable $\hat{R}_2 \sim U[0,1]$ will be used to get the variate from the selected component distribution through the PIT method.

Appendix C

Log File Format

The log files produced by the application flow reassembly engine described in Chapter 4.3.3 describe one Gnutella message per line. Each line is divided into a number of fields separated by a space character. The number of fields is message dependent. However, the first nine fields, which are enumerated below, appear in each message. An excerpt from a Gnutella log file is shown in Figure C.1.

1. Timestamp when the message was recorded at link layer
2. Source IP address and port number separated by colon. In the example below they have been replaced by `src:sport` to protect user anonymity
3. Destination IP address and port number separated by colon. In the example below they have been replaced by `dst:dport` to protect user anonymity
4. Message duration. This field is zero if the whole message arrived in one TCP segment. Otherwise it is computed as the difference between the first and the last TCP segment that carried the message
5. Gnutella message type
6. Global user identity. For handshake messages it is set to the string "GUID"
7. Message size in bytes, including Gnutella header
8. TTL value
9. Hops value

```
# timestamp source destination duration type GUID size TTL hops variable_size_msg_dependent
1120224868.015149 src:sport dst:dport 0.000000 QHD ad77c399f6bc368bf1caf06e1e5a2 260 5 2 1 68.35.140.188:6346 392
1120224868.015343 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.015521 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.015708 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.015885 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.016280 src:sport dst:dport 0.000000 PONG 6891e074e084169794e7a2385c2cc2 54 1 7 ip:port 0 0
1120224868.016280 src:sport dst:dport 0.000000 PONG 6891e074e084169794e7a2385c2cc2 61 3 5 ip:port 56 357389
1120224868.016280 src:sport dst:dport 0.000000 PONG 6891e074e084169794e7a2385c2cc2 61 1 7 ip:port 62 3616621
1120224868.016458 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.016639 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.016805 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.016997 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.017398 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.017592 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.017836 src:sport dst:dport 0.000000 QUERY df0c3a39fb75834045a3cc6bca00c318 112 1 2 39044
1120224868.019315 src:sport dst:dport 0.000000 SER_HSK GUID 627 0 0 503 Full
1120224868.040922 src:sport dst:dport 0.000000 CLI_HSK GUID 483 0 0
1120224868.044415 src:sport dst:dport 0.000000 CLI_HSK GUID 483 0 0
1120224867.839559 src:sport dst:dport 0.208864 SER_HSK GUID 638 0 0 200 OK
1120224868.049039 src:sport dst:dport 0.000000 FIN_HSK GUID 50 0 0 200 OK
1120224868.085769 src:sport dst:dport 0.000000 CLI_HSK GUID 485 0 0
1120224868.112756 src:sport dst:dport 0.000000 CLI_HSK GUID 483 0 0
1120224868.116255 src:sport dst:dport 0.000000 CLI_HSK GUID 484 0 0
1120224868.135865 src:sport dst:dport 0.000000 QUERY 8764e8b531c1f67f2bfb129d51f9b617 101 2 2 38912
1120224868.156855 src:sport dst:dport 0.000000 PING 00000000 23 7 0
1120224868.176215 src:sport dst:dport 0.000000 CLI_HSK GUID 324 0 0
1120224868.177083 src:sport dst:dport 0.000000 SER_HSK GUID 506 0 0 200 OK
1120224868.177591 src:sport dst:dport 0.000000 VEND 20e0290186b02cebd1f0e6b57a497c 129 1 0
1120224868.177654 src:sport dst:dport 0.000000 QRP b9425fa094da61705a06f16c67dd7cb2 29 1 0 reset 16384 2
1120224868.177654 src:sport dst:dport 0.000000 QRP 41e55fa0a6a09503e65cb656db99970 196 1 0 patch 1 1 0x1 4
1120224868.177654 src:sport dst:dport 0.000000 PING ebdf5fa05bf1c01796ff6a2383bb32a 30 4 0
1120224868.177654 src:sport dst:dport 0.000000 VEND 00000000 169 1 0
1120224868.183475 src:sport dst:dport 0.000000 CLI_HSK GUID 485 0 0
1120224868.268282 src:sport dst:dport 0.000000 SER_HSK GUID 615 0 0 200 OK
1120224868.268977 src:sport dst:dport 0.000000 FIN_HSK GUID 383 0 0 409 Vendor would exceed 60% of our slots
1120224868.274402 src:sport dst:dport 0.000000 SER_HSK GUID 472 0 0 503 Full
```

Figure C.1: Excerpt from Gnutella log file

Appendix D

Acronyms

AOL	America Online
API	Application Programming Interface
BTH	Blekinge Institute of Technology
CAN	Content-Addressable Network
CCDF	Complementary CDF
CD	Compact Disc
CDF	Cumulative Distribution Function
CEDF	Complementary EDF
CGI	Common Gateway Interface
CIDR	Classless Inter-Domain Routing
DHT	Distributed Hash Table
DiffServ	Differentiated Services
DNS	Domain Name System
EDA	Exploratory Data Analysis
EDF	Empirical Distribution Function
FTP	File Transfer Protocol
F2F	Firewall-to-Firewall
GGEP	Gnutella Generic Extension Protocol
GNet	Gnutella Network
GUID	Globally Unique ID
GWC	Gnutella Web Cache
HSEP	Horizon Size Estimation Protocol
HTTP	HyperText Transfer Protocol
HUGE	Hash/URN Gnutella Extensions
IANA	Internet Assigned Numbers Authority
IntServ	Integrated Services
IP	Internet Protocol
IRC	Internet Relay Chat
ISP	Internet Service Provider
LAN	Local Area Network
LN	Leaf Node

LRD	Long-Range Dependent
MD5	Message-Digest algorithm 5
ML	Maximum Likelihood
MPLS	Multiprotocol Label Switching
NAT	Network Address Translator
NFS	Network File System
OS	Operating System
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PARQ	Passive/Active Remote Queueing
PC	Personal Computer
PDF	Probability Density Function
PFS	Partial File Sharing
PIT	Probability Integral Transform
QoS	Quality of Service
QRP	Query Routing Protocol
RUDP	Reliable UDP File Transfer
RTT	Round Trip Time
SHA-1	Secure Hash Algorithm One
SMTP	Simple Message Transfer Protocol
SRD	Short-Range Dependent
TCP	Transport Control Protocol
THEX	Tree Hash EXchange format
TTL	Time to Live
UDP	User Datagram Protocol
UHC	UDP Host Cache
UP	Ultrapeer
URL	Universal Resource Locator
URN	Uniform Resource Names
VLSI	Very Large-Scale Integration
VPN	Virtual Private Network

Bibliography

[AG04] Nadia Ben Azzouna and Fabrice Guillemin. Experimental analysis of the impact of peer-to-peer applications on traffic in commercial ip networks. *European Transactions on Telecommunications: Special Issue on P2P Networking and P2P Services*, 2004.

[AH00] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000. http://www.firstmonday.org/issues/issue5_10/adar/index.html.

[And01] David G. Andersen. Resilient overlay networks. Master's thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 2001.

[BBC06] BBC. BBC iMP, March 2006.
http://www.bbc.co.uk/imp.

[BC02] Nevil Brownlee and K. C. Claffy. Understanding internet traffic streams: Dragonflies and tortoises. In *IEEE Communications Magazine*, pages 110–117. October 2002.

[BCNN01] Jerry Banks, John S. Carson II, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07548, U.S.A, 3rd edition, 2001. ISBN: 0-13-088702-1.

[Ber94] Jan Beran. *Statistics for Long-Memory Processes*, volume 61 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, 1994. ISBN: 0-412-04901-5.

[Ber06a] UC Berkeley. The oceanstore project, March 2006.
http://oceanstore.cs.berkeley.edu.

[Ber06b] UC Berkeley. SETI@Home, March 2006.
http://setiathome.ssl.berkeley.edu.

[BLC90] Tim Berners-Lee and Robert Cailiau. Worldwideweb: Proposal for a hypertext project, November 1990.
http://www.w3.org/Proposal.

[BLNZ94] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. Technical Report NAM-08, Northwestern University, Evanston, IL, USA, May 1994.

[BMQD04] Sam Berlin, Andrew Mickish, Julian Qian, and Sam Darwin. *Multicast in Gnutella. Document Revision 2*. The Gnutella Developer Forum (GDF), November 2004.
http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/LAN Multicast.

[Bre73] Richard P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1973. ISBN 0-13-022335-2.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN: 0-521-83378-7.

[CB97] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[Cc05] Gerald Combs and contributors. Ethereal: A network protocol analyzer.
http://www.ethereal.com, November 2005.

[CCLS03] Jin Cao, William S. Cleveland, Dong Lin, and Don X. Sun. *Nonlinear Estimation and Classification*, volume 171 of *Lecture Notes in Statistics*, chapter Internet Traffic Tends Toward Poisson and Independent as the Load Increases, pages 83–110. Springer-Verlag, 2003. ISBN: 0-387-95471-6.

[CCR04] Miguel Castro, Manuel Costa, and Antony Rowstron. Peer-to-peer overlays: structured, unstructured, or both? Technical report, Microsoft Research, Cambridge, UK, July 2004.

[Cli03] Clip2. *The Annotated Gnutella Protocol Specification v0.4*. The Gnutella Developer Forum (GDF), 1.8th edition, July 2003. http://groups.yahoo.com/group/the_gdf/files/Development.

[CM65] D. R. Cox and H. D. Miller. *The Theory of Stochastic Processes*. Chapman & Hall, London, UK, 1965. ISBN: 0-412-15170-7.

[CM03] Justin Chapweske and Gordon Mohr. Tree hash exchange format (THEX). Web page, March 2003. http://www.open-content.net/specs/draft-jchapweske-thex-02.html.

[Coh03] Bran Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer to Peer Systems*, June 2003.

[Col01] Stuart Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer Series in Statistics. Springer-Verlag, London, UK, 2001. ISBN: 1-85233-459-2.

[Com00] Douglas E. Comer. *Internetworking with TCP/IP: Principles, Protocols, and Architectures*, volume 1. Prentice Hall, Upper Saddle River, NJ, USA, 4th edition, 2000. ISBN: 0-13-018380-6.

[Com06] Skype Communications. Skype, March 2006. http://www.skype.com.

[CR02] Jin Cao and Kavita Ramanan. A poisson limit for bbuffer overflow probabilities. In *Proceedings of IEEE Infocom 2002*, number 1, pages 994–1003, June 2002.

[CRB03] Yatin Chawathe, Sylvia Ratnasamy, and Lee Breslau. Making gnutella-like p2p systems scalable. In *Proc. ACM SIGCOMM '03*, August 2003.

[CTB98] Mark E. Crovella, Murad S. Taqqu, and Azer Bestavros. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, chapter 1: Heavy-Tailed Probability Distributions in the World Wide Web, pages 3–25. Birkäuser, Boston, USA, 1998. ISBN: 0-8176-3951-9.

[Div06] DivX Inc. DivX, December 2006. http://www.divx.com.

[DJ50] F. N. David and N. L. Johnson. The probability integral transform when the variable is discontinuous. *Biometrika*, 37(1–2):42–49, June 1950.

[DS86] Ralph B. D'Agostino and Michael A. Stephens. *Goodness-of-Fit Techniques*, volume 68 of *STATISTICS: textbooks and monographs*. Marcel Dekker, Inc., 1986. ISBN: 0-8247-7487-6.

[EIP05] David Erman, Dragos Ilie, and Adrian Popescu. Bittorrent session characteristics and models. In Demetres Kouvatsos, editor, *Proceedings of the 3rd International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks*, pages P30/1–P30/10, Ilkley, West Yorkshire, UK, July 2005.

[EIPN04] David Erman, Dragos Ilie, Adrian Popescu, and Arne A. Nilsson. Measurement and analysis of bittorrent signaling traffic. In Olav Østerbø and Harald Pettersen, editors, *Proceeding of the 17th Nordic Teletraffic Seminar (NTS17)*, pages 279–290, Fornebu, Norway, August 2004.

[ELL06] B. Even, Y. Li, and D. J. Leith. Evaluating the performance of tcp stacks for high-speed networks. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks*, Nara, Japan, February 2006.

[Enc05] Wikipedia Encyclopedia. Gnutella, December 2005.
http://en.wikipedia.org/wiki/Gnutella.

[Erm05] David Erman. *BitTorrent Traffic Measurements and Models*. Licentiate dissertation, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, October 2005. ISBN: 91-7295-071-4.

[ES05a] Jörg Eberspächer and Rüdiger Schollmeier. *Peer-to-Peer Systems and Applications*, Chapter 2. What is This "Peer-to-Peer" About?, pages 9–16. Number 3485 in Lecture Notes in Computer Science. Springer-Verlag, Berling, Heidelberg, Germany, 2005. ISBN: 3-540-29192-X.

[ES05b] Jörg Eberspächer and Rüdiger Schollmeier. *Peer-to-Peer Systems and Applications*, Chapter 5. First and Second Generation of Peer-to-Peer Systems, pages 9–16. Number 3485 in Lecture Notes in Computer Science. Springer-Verlag, Berling, Heidelberg, Germany, 2005. ISBN: 3-540-29192-X.

[Fis03a] Adam A. Fisk. *Gnutella Dynamic Query Protocol*. LimeWire LLC, 0.1 edition, May 2003.
http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/search/Dynamic Querying/.

[Fis03b] Adam A. Fisk. *Gnutella Ultrapeer Query Routing*. Lime Wire LLC, 0.1 edition, May 2003.
http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/search/Ultrapeer QRP.

[FM03] Michael J. Freedman and David Maziéres. Sloppy hashing and self-organizing clusters. In *Proceedings of 2nd Intl. Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, February 2003.

[GA05] Jean-loup Gailly and Mark Adler. zlib, December 2005.
http://www.gzip.org/zlib.

[GDF05] The GDF. Gnutella protocol development, December 2005.
http://www.the-gdf.org.

[GDS$^+$03] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19)*, Bolton Landing, NY, USA, October 2003.

[Gha05] Saeed Ghahramani. *Fundamentals of Probability with Stochastic Processes*. Pearson Prentice Hall, Upper Saddle River, NJ, USA, 3rd edition, 2005. ISBN: 0-13-145340-8.

[Gnu05] The Gnutella Developer Forum (GDF). *Reliable UDP (RUDP) File Transfer Spec 1.0*, February 2005.
http://groups.yahoo.com/group/the_gdf/files/Proposals/Pending Proposals/F2F/.

[Har68] Garett Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.
http://www.dieoff.com/page95.htm.

[HB70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communication of the ACM*, Volume 13(Number 7):p. 422–426, July 1970. ISSN:0001-0782.

[HP01] Mark Handley and Vern Paxon. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceeding of the 10th USENIX Security Symposium*, Washington, D.C., USA, August 2001. USENIX Association.

[IEP06] Dragos Ilie, David Erman, and Adrian Popescu. Transfer rate models for gnutella signaling traffic. In *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW'06)*, Guadeloupe, French Caribbean, February 2006.

[IEPN04a] Dragos Ilie, David Erman, Adrian Popescu, and Arne A. Nilsson. Measurement and analysis of gnutella signaling traffic. In Veljko Milutinović, editor, *Advances in the Internet Technology: Concepts and Systems*, pages Proceedings of IPSI–2004, Stockholm, Sweden, September 2004.

[IEPN04b] Dragos Ilie, David Erman, Adrian Popescu, and Arne A. Nilsson. Traffic measurements of p2p systems. In *Proceedings of the 2nd Swedish National Computer Networking Workshop (SNCNW'04)*, pages 25–29, Karlstad, Sweden, November 2004.

[Ins06] Fraunhofer Institute. Audio & multimedia MPEG layer-3, January 2006. http://www.iis.fraunhofer.de/amm/techinf/layer3/index.html.

[IWS05] IWS. Internet world stats, December 2005. http://www.internetworldstats.com.

[JKB94] Norman L. Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous Univariate Distributions, Vol. 1.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc., 2nd edition, 1994. ISBN: 0-471-58495-9.

[JLM05] Van Jacobsen, C. Leres, and S. McCanne. Tcpdump. http://www.tcpdump.org, August 2005.

[KBB$^+$03] Thomas Karagiannis, Andre Broido, Nevil Brownlee, K. Claffy, and Michalis Faloutsos. File-sharing in the internet: A characterization of p2p traffic in the backbone. Technical report, University of California, Riverside, November 2003.

[KBB$^+$04] Thomas Karagiannis, Andre Broido, Nevil Brownlee, KC Claffy, and Michalis Faloutsos. Is p2p dying or just hiding? In *IEEE Globecom 2004 - Global Internet and Next Generation Networks*, Dallas, TX, USA, December 2004.

[KBFC04] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. Transport layer identification of p2p traffic. In *Internet Measurement Conference (IMC)*, Taormina, Sicily, Italy, October 2004.

[KK77] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks, performance evaluation and optimization. In *Computer Networks*, volume 1, pages 155–174, January 1977.

[Kle75] Leonard Kleinrock. *Queueing Systems Volume 1: Theory*. John Wiley & Sons, Inc., 1975. ISBN: 0-471-49110-1.

[KM02] Tor Klingberg and Raphael Manfredi. *Gnutella 0.6*. The Gnutella Developer Forum (GDF), 200206-draft edition, June 2002. http://groups.yahoo.com/group/the_gdf/files/Development/.

[KR01] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice*. Addison Wesley, 2001. ISBN: 0-201-71088-9.

[Lei03] D. J. Leith. Linux implementation issues in high-speed networks. Technical report, Hamilton Institute, Ireland, 2003.

[LK00] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 3rd edition, 2000. ISBN: 0-07-059292-6.

[LM86] Richard J. Larsen and Morris L. Marx. *An Introduction to Mathematical Statistics and Its Applications*. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA, 2nd edition, 1986. ISBN: 0-13-487174-X.

[LM97] T. V. Lakshman and Upamanyu Madhow. The performance of tcp/ip for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, June 1997.

[LM04] Zhi Li and Prashant Mohapatra. Qron: Qos-aware routing in overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):29–40, January 2004.

[LMMR06] Christoph Lampert, Pascal Massimino, Michael Militzer, and Peter Ross. XviD, December 2006.
http://www.xvid.org.

[LMS05] P. Leach, M. Mealling, and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*, July 2005. RFC 4122.

[LRWE98] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Wright Paul E. Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.

[LTWW94] Will E. Leeland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[Man03a] Raphael Manfredi. *Gnutella Traffic Compression*. The Gnutella Developer Forum (GDF), January 2003.
http://groups.yahoo.com/group/the_gdf/files/Development.

[Man03b] Raphael Manfredi. *Passive/Active Remote Queueing (PARQ), Version 1.0.a*. The Gnutella Developer Forum (GDF), May 2003.
http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/QUEUE/.

[Man06] Raphael Manfredi. `gtk-gnutella`, March 2006.
http://gtk-gnutella.sourceforge.net.

[MB03] John Maindonald and John Braun. *Data Analysis and Graphics using R: An Example-based Approach*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2003. ISBN: 0-521-81336-0.

[MF04] John H. Mathews and Kurtis K. Funk. *Numerical Methods using Matlab*, chapter 8: Numerical Optimization, pages 430–436. Prentice-Hall, Inc., 4th edition, 2004. ISBN: 0-13-065248-2.

[MGB74] Alexander M. Mood, Franklin A. Graybill, and Duane C. Boes. *Introduction to the Theory of Statistics*. McGraw-Hill, 3rd edition, 1974. ISBN: 0-07-085465-3.

[MKL$^+$03] Dejan S. Milojicic, Van Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto, July 2003.
http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf.

[MM03] Petar Maymounkov and David Maziéres. Rateless codes and big downloads. In *Proceedings of IPTPS '03*, Berkeley, CA, 2003.

[Moh02] G. Mohr. *Hash/URN Gnutella Extensions (HUGE) v0.94*. The Gnutella Developer Forum (GDF), April 2002.
http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/HUGE/.

[MR99] Douglas C. Montgomery and George C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Inc., 2nd edition, 1999. ISBN: 0-471-17027-5.

[Odl03] Andrew M. Odlyzko. Internet traffic growth: Source and implications. In B. B. Dingel, W. Weiershausen, and A. K. Dutta, editors, *Proceedings of SPIE*, volume 5247 of *Optical Transmission Systems and Equipment for WDM Networking II*, pages 1–15, Orlando, FL, USA, August 2003.

[Orl03] Andrew Orlowski. ”I poisoned P2P networks for the RIAA” — whistle-blower. In *The Register*. January 2003.
http://www.theregister.co.uk/2003/01/17/i_poisoned_p2p_networks.

[Ost05] Shawn Ostermann. Tcptrace.
http://www.tcptrace.org, August 2005.

[Par04] Andrew Parker. The true picture of peer-to-peer file sharing. Presentation, CacheLogic, Cambridge, England, July 2004.
http://www.cachelogic.com/research/p2p2004.php.

[Par05] Andrew Parker. Peer-to-peer in 2005. Presentation, CacheLogic, Cambridge, England, August 2005.
http://www.cachelogic.com/research/p2p2005.php.

[Pax94] Vern Paxson. Empirically derived analytic models for wide-area tcp connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1994.

[PF95] Vern Paxson and Sally Floyd. Wide area traffic: The failure of the poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[PKC96] Kihong Park, Gitae Kim, and Mark Crovella. On the relationship between file sizes, transport protocol, and self-similar network traffic. In *Proceedings of IEEE International Conference on Network Protocols (ICNP’96)*, pages 171–180, October 1996.

[PN98] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., January 1998.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vellering, and Brain P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, October 1992. ISBN: 0-521-43108-5.

[PV02] Atilla Pásztor and Darryl Veitch. Pc-based precision timing without gps. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems SIGMETRICS '02*, volume 30, pages 1–10, June 2002.

[PW00] Kihong Park and Walter Willinger. *Self-Similar Network Traffic and Performance Evaluation*, chapter 1: Self-Similar Network Traffic: An Overview, pages 1–38. 2000. ISBN: 0-471-31974-0.

[R D05] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0
http://www.R-project.org.

[RFH$^+$01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, pages 161–172, San Diego, CA, August 2001. ACM Press.

[Rit01] Jordan Ritter. *Why Gnutella Can't Scale. No, Really.*, February 2001.
http://www.darkridge.com-/˜jpr5-/doc-/gnutella.html.

[Roh02a] Christopher Rohrs. *Query Routing for the Gnutella Network*. Lime Wire LLC, 1.0 edition, May 2002.
http://groups.yahoo.com/group/the_gdf/files/Development.

[Roh02b] Christopher Rohrs. *SACHRIFC: Simple Flow Control for Gnutella*. Lime Wire LLC, March 2002.

[Ros92] Sheldon M. Ross. *Applied Probability Models With Optimization Applications*. Dover Books On Mathematics. Dover Publications, 1992. ISBN: 0-486-67314-6.

[Rud87] Walter Rudin. *Real and Complex Analysis*. Mathematics Series. McGraw-Hill, 1987. ISBN: 0-07-100276-6.

[Sch01] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the IEEE 2001 International Conference on Peer-to-Peer Computing (P2P2001)*, Linköping, Sweden, August 2001.

[Sch04] Thomas Schürger. *Horizon size estimation on the Gnutella network v0.2*, March 2004.
http://www.menden.org/gnutella/hsep.html.

[SFK05] Pyda Srisuresh, Bryan Ford, and Dan Kegel. *State of Peer-to-Peer (P2P) communication across Network Address Translators (NATs).* Internet Draft, October 2005. draft-srisuresh-behave-p2p-state-01.txt.

[SGG01] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical Report UW-CSE-01-06-02, Deparment of Computer Science and Engineering, University of Washington, Seattle, WA, USA, July 2001.

[SGG02] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, January 2002.

[SM05] Burkhard Stiller and Jan Mischke. *Peer-to-Peer Systems and Applications*, Chapter 17. Peer-to-Peer Search and Scalability, pages 269–288. Number 3485 in Lecture Notes in Computer Science. Springer-Verlag, Berling, Heidelberg, Germany, 2005. ISBN: 3-540-29192-X.

[SMK$^+$01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[sof06] `ns-2`, 2006.
http://nsnam.isi.edu/nsnam/index.php/Main_Page.

[SR02] Anurag Singla and Christopher Rohrs. *Ultrapeers: Another Step Towards Gnutella Scalability.* Lime Wire LLC, 1.0 edition, November 2002.
http://groups.yahoo.com/group/the_gdf/files/Development.

[SSBK04] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy Katz. OverQoS: An overlay based architecture for enhancing internet qos. In *Proceedings at 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[Sta02] William Stallings. *High-Speed Networks and Internets: Peformance and Quality of Service*. Prentice Hall, Upper Saddle River, NJ 07458, USA, 2nd edition, 2002. ISBN: 0-13-032221-0.

[SW86] Kotikalapudi Sriram and Ward Whitt. Characterizing superposition arrival processes in packet multiplexers for voice and data. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):833–846, September 1986.

[SW02] Subhabrata Sen and Jia Wang. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 137–150, Marseille, France, November 2002.

[Tha01] Sumeet Thadani. *Metadata Extension*. The Gnutella Developer Forum (GDF), 2001.
http://www.the-gdf.org/wiki/index.php?title=Metadata_Extension.

[Tho02] Jason Thomas. *Gnutella Generic Extension Protocol (GGEP)*, February 2002.
http://rfc-gnutella.sourceforge.net/src/GnutellaGenericExtensionProto-col.0.51.html.

[Tor05] Linus Torvalds. `do_gettimeofday()`. Linux 2.6.14 Kernel Sources, 2005. arch/i386/kernel/time.c.

[TSM85] D. M. Titterington, A. F. M. Smith, and U. E Makov. *Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons, Inc., 1985. ISBN: 0-471-90763-4.

[Var06] András Varga. `OMNet++`, March 2006.
http://www.omnetpp.org.

[VR99] William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S-PLUS*. Statistics and Computing Series. Springer-Verlag, 3rd edition, 1999. ISBN: 0-387-98825-4.

[Wan97] M. P. Wand. Data-based choice of histogram bin width. *The American Statistician*, 51:59–64, 1997.

[Wan00] Zheng Wang. *Internet QoS: Architectures and Mechanisms*. Morgan Kaufman Publishers, October 2000. ISBN: 1-55860-608-4.

[Wik05] Wikipedia. World wide web, December 2005. http://en.wikipedia.org/wiki/World_Wide_Web.

[WPRT03] Walter Willinger, Vern Paxson, Rolf H. Riedi, and Murad S. Taqqu. *Theory and Applications of Long-Range Dependence*, chapter : Long-Range Dependence and Data Network Traffic, pages 373–407. Birkäuser, 2003. ISBN: 0-8176-4168-8.

[WS95] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated: The Implementation*, volume 2 of *Professional Computing Series*. Addison Wesley, 1995. ISBN: 0-201-63354-X.

[WTSW97] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, February 1997.

[XHG03] Kaixin Xu, Xiaoyan Hong, and Mario Gerla. Landmark routing in ad hoc networks with mobile backbones. *Journal of Parallel and Distributed Computing (JPDC)*, 63(2):110–122, February 2003.

[Zak97] Robert H'obbes' Zakon. *RFC 2235: Hobbes' Internet Timeline*. IETF, November 1997. Category: Informational.

[Zak05] Robert H'obbes' Zakon. Hobbes' internet timeline v8.1, December 2005. http://www.zakon.org/robert/internet/timeline.

[ZDHJ02] Ben Y. Zhao, Yitao Duan, Ling Huang, and Anthony D. Joseph. Brocade: Landmark routing on overlay networks. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, March 2002.