ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Philipp Jardas

P2P Filesharing Systems: Real World NetFlow Traffic Characterization

Bachelor's Thesis BA-2004-01
November 2003 to February 2004

Supervisors: Arno Wagner, Thomas Dübendorfer
Professor: Bernhard Plattner

Abstract

During the last few years peer-to-peer filesharing networks have grown significantly and are today a source of considerable amounts of traffic. Some studies even claim that filesharing is today's top bandwidth-consuming application, and has already overtaken the World Wide Web. Since filesharing applications are not restricted to specific ports there is yet no method known to reliably identify filesharing traffic in NetFlow data logfiles.

This Bachelor's Thesis develops and implements some experimental approaches to the identification of filesharing traffic based on NetFlow data. First today's major filesharing network implementations were observed in operation to determine the main characteristics of the generated traffic. This knowledge was used to devise possible approaches for the detection of filesharing traffic within the NetFlow data. The most promising approaches were implemented and validated using NetFlow data gathered at SWITCH, a medium sized Swiss backbone provider.

The thesis shows that identification of filesharing traffic is non-trivial. Failed approaches are discussed so they can be avoided in future work on this topic. Nevertheless the thesis shows that rather accurate concepts exist and concludes with suggestions for promising subjects of future research.

Author

Philipp Jardas
Bülachstrasse 11f
8057 Zürich
Switzerland
<philipp@jardas.de>

School

Institut für technische Informatik und Kommunikationsnetze
Departement Informationstechnologie und Elektrotechnik
Eidgenössische Technische Hochschule (ETH) Zürich

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
Swiss Federal Institute of Technology (ETH) Zurich

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Distributed Denial of Service (DDoS) attacks are a threat to Internet services ever since the widely published attacks on ebay.com and amazon.com in 2000. ETH itself was the target of such an attack 6 months before these commercial sites where hit. ETH suffered repeated complete loss of Internet connectivity ranging from minutes to hours in duration. Massively distributed DDoS attacks have the potential to cause major disruption of Internet functionality up to and including severely decreasing backbone availability.

So far, peer-to-peer (P2P) systems were not involved in DDoS attacks. However, as their user community and herewith their share in the use of the network bandwidth is constantly rising, there is a non negligible probability that they could be used as a DDoS attack platform.

1.2 About DDoSVax

In the joint ETH/SWITCH research project DDoSVax[1] abstract Internet traffic data (Cisco NetFlow) is collected at all border gateway routers operated by SWITCH. This data contains information about which Internet hosts were connected to which others and how much data was exchanged over which protocols.

1.3 Bachelor's Thesis Task

The aim of this thesis is to characterize the netflow traffic patterns caused by presently used P2P filesharing applications. The approach will involve measurements on P2P applications, as well as the identification and analysis of P2P traffic in abstracted traffic data (Cisco Netflow) of a Swiss Backbone provider (SWITCH). Since not all P2P system use fixed TCP/UDP ports, P2P traffic identification may be non-trivial.

This task is split into the following subtasks:

1.3.1 Related Work

Information about current P2P systems has to be gathered. Among others the number of P2P users, usage frequency, and typical size of shared files are of interest.

[1] see `http://www.tik.ee.ethz.ch/~ddosvax/`

The systems should be characterized by the different ways they work. About four systems will be analyzed in detail in this thesis.

In case that there exist any recent traffic measurements on P2P systems they must be taken into consideration.

1.3.2 Analysis of P2P Traffic

For each one of the more closely analyzed P2P systems characteristic traffic patterns that distinguish their traffic from other TCP or UDP traffic will be defined. Network monitors and analyzers as well as self-engineered tools will be used to observe the behaviour of the systems.

Further more, traffic patterns for P2P systems that show up in the Cisco Netflow traffic data gathered in the DDoSVax project will be defined and investigated.

By analyzing real Internet traffic, an estimation on the number of (active) users of P2P systems should be made and a rough number on the network bandwidth consumed by P2P traffic should be given.

1.3.3 Timeframe

The bachelor's thesis will start on November 10th, 2003 and will be finished twelve weeks later by February 9th, 2004. This is the first Bachelor's thesis written at ETH.

1.3.4 Supervisors

Arno Wagner (wagner@tik.ee.ethz.ch)
Thomas Dübendorfer (duebendorfer@tik.ee.ethz.ch)

Chapter 2

Peer-To-Peer Systems Overview

In this chapter a global overview of the most important filesharing systems up to now is being given. The basic underlying principles and structures are being examined and compared.

2.1 Definition

Peer-to-peer: Process whereby computers can trade information between each other without having to pass the information through a centrally controlled server as with email programs. [EDi]

File sharing: A method of allowing one server to give the same file to many different end users. [GVi]

Throughout this thesis the term *peer-to-peer* relates strictly to peer-to-peer filesharing applications.

Note that there are two common pitfalls when discussing P2P networks: First, one has to distinguish between a P2P *network* or *system* and a *client* that connects to these networks. Often these two parts have the same names—like eDonkey, which describes both client and underlying network. On the contrary many clients are able to connect to more than one network or have different names than the network. KaZaA, for instance, is a client that connects to the FastTrack network.

Second, the use of the terms *user* and *client* has to be defined. A *client* is a program that connects to one or more P2P networks. A *user* is a human or process attending one or more clients per computer. So the client acts as an interface between human and network.

2.2 Conjoint Features

All P2P filesharing systems share some common features that clearly distinguish them from other content delivery systems like WWW or FTP. The most remarkable property of P2P systems lies in the renunciation of the classical client–server approach towards the concept of *servents* (fusion of *serv*er and cli*ent*), which utilizes the fact that every client connected to the specific P2P network automatically serves as a server which is connected to other servents, creating an overlay network above the transport layer.

This results in clients having many inbound connections, as well as outgoing connections.

FastTrack	3356	36.16%	
eDonkey	1663	17.92%	
BitTorrent	1500	16.16%	*(estimated)*
iMesh	1098	11.84%	
Overnet	1001	10.97%	
MP2P	250	2.70%	
DirectConnect	210	2.26%	
Gnutella	202	2.17%	

Table 2.1: User statistics of P2P networks. Concurrent users connected, averaged over January 2004 (in thousands). Data gathered from [Slya, Men]

2.3 User Statistics

It is generally very difficult to determine the exact amount of users on a specific network, due to the decentralized approach of P2P networks. It is, however, relatively easy to create statistics about the usage of filesharing clients, since most clients have some kind of usage reporting features included. From the number of clients connected to the network you can infer an estimate number of connected users. Given that an increasing number of clients have the ability to connect to more than one network and due to usage fluctuations these figures indeed are rough estimates, but they serve well when it comes to determining relative sizes of distinct P2P networks.

Table 2.1 shows some recent user data gathered by slyck.com. Note that this is by no means a complete list of all existing filesharing protocols, it only contains P2P systems that gather user data and report it to slyck.com. BitTorrent, for instance, surely is one of the biggest filesharing communites, but the nature of both its protocol and its client do not allow a user statistic to be generated. However, one of the biggest BitTorrent sites Suprnova.org has recently published statistics about the trackers this site knows (estimatedly 60%) and places the number of BitTorrent users in the region between one and two million. [Men] iMesh has started in 1999 with its own network—thus it is listed separately— but has jumped onto the FastTrack train and uses it exclusively by now, so its users can be added to the FastTrack users.

From table 2.1 one can easily select the biggest networks, namely FastTrack, eDonkey and Overnet. These three networks contain 90% of all filesharing users, thus generating approximately the same fraction of traffic. So these will be the networks that will be covered in this short survey of P2P systems. Additionally, we will look at BitTorrent because it surely is one of the big traffic generating networks and Gnutella because it is somehow “the mother” of P2P filesharing networks and still a prototype for most existing protocols. Napster and Audiogalaxy, however, will not be covered herein simply because their technologies are completely outdated and the network traffic generated by them is neglectable.

2.4 P2P Networks

Each of the networks will be discussed the same way: First a little bit of history about the development of the network, then a specification of the underlying network structure followed by an outline of user behaviour, common clients and the file types that are typically shared on this network, finally some remarks about pros and cons, scalability and prospects of further development. The order in which the networks are being covered is roughly chronological.

Many commercial clients are being funded by third-party software that is be-

ing distributed with the client and installed automatically with or without the user's knowledge. Due to the nature of these programs they are often referred to as *adware* or *spyware*. This additional software can cause serious harm and severe privacy violations and is very much deprecated by the community. Slyck.com about Grokster: "We were disappointed when we loaded this client (...). In total, we ended up with 3 ugly modifications to Internet Explorer, 6 third party programs installed and 8 new icons pointing to casinos left on the desktop." [slyb]

2.4.1 Gnutella

History

After Napster had been closed down by American authorities on July 11, 2001 [Mar01] the newly created P2P community had to research into alternatives that would avoid the major handicap of Napster: its server based file indexing method. Every connected client had to register all its shared files with on of about 50 central servers, which would then in turn answer all search queries as a central omniscient authority, which was both Napster's biggest advantage and its biggest downside, since it was possible for the authorities to shut down the whole service by just switching off the central servers that were directly controlled by Napster, Inc.

Having learnt from this the open source community developed Gnutella, a new protocol that abided the basic principle of the Internet itself: decentralization. No centralized structure, not even the slightest, was allowed to encroach into the new protocol.

Network overview

The basic principle behind Gnutella is broadcasting. Every servent maintains TCP connections to a number of other servents thus creating a web like overlay network. A servent issuing a query sends it to every servent it is connected to, which searches its local database for the file. If the file is being found, the queried servent answers with further information about the file, whereupon the first servent commences a direct connection to the servent hosting the file and requesting a download. If the queried servent does not hold the required file in stock, it forwards the message to any other connected servents, thus spreading the message around the network. If a certain number of hops is reached without finding the file, the query is being dropped. Since every servent replies to the host the query was received from, a query reply will take the same way back to the issuer as the query has gone the first way.

The file download itself is being conducted via the HTTP protocol, which is normally used for the WWW. Contained within the query reply is a port number on which the server can be connected to. The client now directly connects to the server on this port sending a HTTP GET request, which—if everything works fine—will be answered by the requested file.

The network protocol description provided here is based on the Gnutella Protocol Specification v0.4 [Cli00] and the draft v0.6 [gnu02].

Clients and File Characterization

The Gnutella protocol is publicly available, hence numerous clients for all conceivable operating systems have been written over time. The most commonly used are: [Ber03][Slya]

BearShare is a Gnutella client for Windows containing the latest features of Gnutella clients. [Bea]

Limewire is a Java based, open source Gnutella client that has the advantage of being capable of running on many platforms. [Lim]

Shareaza P2P is the founder of the Gnutella2 network and also connects to eDonkey and BitTorrent. [Shab]

Gnucleus is a very simple open source client with high popularity. [Gnu]

Morpheus first started as MusicCity, then switched name to Morpheus and network to FastTrack, then to Gnutella. Now it incorporates Gnutella, FastTrack, eDonkey, Overnet and iMesh. [Mor]

Since Gnutella was quite the only filesharing network for quite a while, its file distribution varied from smallest (less than 1MB) to big files (less than 1GB). Since files are not being sliced but have to be downloaded as a whole, the filesize is limited: You can only download from one servent per file which takes a lot of time for big files. From once having been the *one* P2P network Gnutella's user figures decayed with the emergence of newer, faster protocols. As of January 2004 the network contained only roughly 185,000 users, which is not even 7% of the current number of FastTrack users.

Disquisition and Prospects

Despite the disadvantage of only one download per file the one thing that doomed Gnutella from the beginning was its lack of scalability. Ritter showed in February 2001, less than a year after the first public release of the first Gnutella client, that "the application was an incredible burden on modern networks and would probably never scale" [Rit01] by calculating that the network overhead generated by a single 18 Byte query could easily reach orders of 800MB or more.

Additionally the Gnutella protocol relies on the decent behavior of all connected clients. There are, however, many ways a client can download heaps of files without sharing, thus resulting in the possibility that one single client can consume up to 80% of the total network resources.

In an attempt to reduce the network overhead generated by queries [gnu02] introduced higher level nodes called *ultrapeers*, powerful nodes that interconnect with few other ultrapeers but many non-ultrapeer nodes (called *leaf nodes*). The ultrapeer's function is to work as cache for the leaf nodes. Every leaf composes a compressed hash table of its shared files and sends it to its ultrapeer who then can answer search queries on behalf of its attached leafes, reducing network traffic efficiently. [Ber03, HHH+02]

Within the last year the *Gnutella2 Developers' Network* has created a new protocol called Gnutella2 [GDN] which at last will be a real P2P network that is not only focused on filesharing but on peer-to-peer applications in general. The filesharing client Shareaza was the first client to implement the protocol and has proven quite successful, mainly because of the fact that it has the capability to connect to as much as four different P2P networks (eDonkey, BitTorrent, Gnutella and Gnutella2).

2.4.2 eDonkey

History

In 2002 MetaMachine [eDo, Ove] developed an alternative protocol to the radical Gnutella approach of complete decentralization. He incorporated a scheme just like Gnutella's ultrapeers and called his new network eDonkey2000.

Network Overview

The basic eDonkey network is being built by hosts running eDonkey servers that interconnect with each other. A client connects to a single server and uploads a hash table of its shared files. The rest of the protocol is rather similar to the Gnutella v0.6 protocol with the exception that users *must* connect to a server in order to join the network. This way the eDonkey network becomes very efficient. Anyone may set up a server so the network stays decentralized while maintaining scalability—to a certain degree.

Files are not only searched by filename but also by a *md5 hash* thus enabling the servers to supply clients with sources that have the same file with a different filename.

The biggest improvement, however, was the introduction of chunkwise downloads. Every shared file is divided into small bits that are shared independently and can be downloaded in any order. So as soon as a client has downloaded a chunk of the big file it is immediately shared for other users to download it. This way even huge files can be downloaded efficiently since the client can simultaneously download from many sources and the number of sources per chunk is being increased because clients don't have to download the whole file before sharing it.

Clients and File Characterization

The most common clients are:

eDonkey is the original client developed by MetaMachine. [eDo]

eMule is the open source variant, which enjoys the most popularity. [eMu]

iMesh connects to the eDonkey network as well. [iMe]

Morpheus is also able to connect to the eDonkey network. [Mor]

The eDonkey protocol was mainly developed to better support sharing of large files by introducing chunked sharing, hence the network is mostly populated with large files like videos and games. Lately filesizes have clearly exceeded the 1GB mark with DVD and game rips with filesizes of up to 5GB.

Disquisition and Prospects

Like many open source "attacks" on proprietary software the totally anarchic approach of Gnutella proved to be too radical. eDonkey, however, incorporated the idea of hierarchical nodes from the beginning and still is one of the largest networks up to now and clearly the favorite network for downloading large files like movies or games.

2.4.3 Overnet

History

Overnet is a complete overhaul of eDonkey that was performed by its owner MetaMachine to overcome the limitations of eDonkey due to its centralized approach.

Network Overview

Overnet is completely decentralized, i.e. it has no hierarchy whatsoever but follows the XOR metric principle introduced with Kademlia [MM02], which dramatically reduces routing overhead and search latency.

Both eDonkey and Overnet use the same file transfer protocol based on HTTP, so users of both networks can exchange files with each other, which is one of the important points that led to the success of the two networks.

Clients and File Characterization

The Overnet network was actually developed as a successor for the eDonkey network. The latter, however, was not performing as bad as anticipated so both eDonkey users and eDonkey server administrators were reluctant to abandon the running eDonkey system. Thus the Overnet network had to fight against low participation in the beginning. This changed with the introduction of the eDonkey/Overnet hybrid, which is able to connect to both networks, hence pushing Overnet's usage statistic to new heights.

The clients connecting to Overnet are the same as those who are capable of connecting to eDonkey.

Since the Overnet and eDonkey networks are mostly accessed by the same clients their file characterization is basically the same: videos, games, full albums.

Disquisition and Prospects

Due to the development the Overnet network is somehow an extension of the existing eDonkey network. With a community that is becoming constantly larger the eDonkey network will reach its scalability limits some other day, making users gradually switch over to the Overnet network.

2.4.4 FastTrack

History

In March 2001 some Amsterdam programmers created another alternative to the non-scalable Gnutella which tried to reduce overhead search traffic by the introduction of *supernodes*. Thus a new network was born: FastTrack. It's userbase has exploded from 20'000 users in April 2001 up to four million by now. [kazb, Slya]

The FastTrack network is being copyrighted by *Sharman Networks* who also develop and maintain the main client KaZaA.

Network Overview

Basically the FastTrack network is very similar to Gnutella. It enhances the functionality of the common non-hierarchical overlay network by introducing *supernodes*, similar to Gnutella's *ultrapeers*. However, supernodes are not special servers that have to be configured separately, but common clients that have a high bandwidth Internet connection and amble system resources. These supernodes are assigned dynamically and in a decentralized fashion, so no single authority has control over them. This way the FastTrack network gains scalability without the drawback of being vulnerable to legal actions.

FastTrack supports chunked downloads as well as file hashes providing advanced search and download performance.

Clients and File Characterization

There exist mainly five clients that connect to the FastTrack network: [Slya]

KaZaA Media Desktop (KMD) is the standard FastTrack client developed by *Sharman Networks*. It provides medium search and download performance due to restrictions to the free version. The program is loaded with ad- and spyware and is commonly known to harm systems severely. [KaZa]

Grokster is actually KMD with a different logo and even more ad- and spyware. [Gro]

KaZaa Lite K++ is an open source modification of KMD that aims to unlock the full potential of the FastTrack network while simultaneously removing any malware contained. [Kli]

iMesh is one of the older P2P clients around. It first connected to its own network, which proved not to be effective, then tapped FastTrack. Recently the iMesh network has gained many users making this client even more popular. [iMe]

Morpheus: The "resurrection" of the famous client now also connects to FastTrack. [Mor]

Files commonly found on FastTrack include music files, albums, music videos as well as applications and adult material. However, FastTrack suffers of many malfunctioning or malevolent applications and the quality of shared material is in general rather low.

2.4.5 BitTorrent

History

Leaving the classical concepts of a coherent searchable network ashore Bram Cohen presented BitTorrent in July 2001, actually designed as a means of publishing files in the quickest way possible to as many clients as possible by distributing the upload burden to the downloading clients. It has, however, been gratefully adopted by the filesharing community and has rapidly grown in popularity during the last years. Recent statistics put BitTorrent's userbase far beyond one million, making it one of the larger filesharing communities. [Men]

Network Overview

The publisher of a file to be shared sets up a BitTorrent *tracker*. Then he creates a *torrent* file containing the tracker URI as well as information about the shared file like file size, hash and chunk information. This small (few 100K) file is being published via any means, traditionally via websites.

A user intending to download the file opens the torrent with his BitTorrent client which then connects and registers to the tracker. The later stores information about all peers holding the file or parts of it and provides each client with a random list of peers holding the file. Clients then mutually exchange file chunks using a variation of tit-for-tat for choosing the hosts to upload to. [Coh03]

Clients and File Characterization

There are four common clients for BitTorrent: [Slya]

BitTorrent is the client Bram Cohen, the developer of BitTorrent, has written himself. [Bitb]

TheShadow's Experimental BT is the more popular enhanced version enabling the user to tweak the behavior of the client. [Shaa]

BitComet is a neat graphical interface that provides a lot more comfort when downloading multiple files. [Bita]

Shareaza P2P is a hybrid that can handle torrents. [Shab]

BitTorrent is clearly designed for the distribution of large files, especially because the user can not conveniently search within the client itself. Furthermore the lifetime of a file is very short (usually below one month, opposed to years in classical filesharing networks) because people who have downloaded the complete file eventually close their clients leaving no seeds to download from.

The main type of files shared with BitTorrent are recent game and movie releases as well as TV shows and mangas.

Disquisition and Prospects

BitTorrent clearly is the prime filesharing system when it comes to download rates which exceed these of classical filesharing systems by far. Since the torrent retrieval process is not as intuitive as searching for a file from within the client, many users have reservations—if they know of BitTorrent at all. It is still an insiders' tip that becomes more popular day by day.

Chapter 3

Comprehensive Analysis

In this chapter the most important file sharing networks will be described in detail providing information needed to identify the traffic generated by them in Cisco NetFlow data.

3.1 About Cisco NetFlow

Cisco NetFlow[1] was developed and patented at Cisco Systems in 1996 and is now the primary network accounting technology in the industry [Cisa]. It regards network traffic not as a heap of single packets but rather as a collection of *flows*, each flow describing one half of a TCP connection. The data types stored in the logfiles used by DDoSVax are listed in table 3.1 on the following page. There is no information about the data being sent in the packages like, for instance, HTTP–headers. Not even the size of the single packets is known, only the total number of packets in a flow and their total size. Hence the identification of filesharing traffic within NetFlow logfiles is anything but trivial.

NetFlow starts a new logfile entry for existing connections every fifteen minutes. That means one *flow* lasts at most fifteen minutes, after that the current entry will be closed and a new flow entry will be started. It is quite easy to mend these fragments, since only the beginning one will have a SYN flag set.

3.2 Methodology

Two kinds of traffic have to be distinguished: *Network* traffic and *file transfer* traffic. *Network* traffic is being generated by the clients and supernodes while maintaining the P2P network. This includes initial handshakes, ping/pong messages, search queries and replies and all other messages that are not file transfer related. *File transfer* traffic is generated by actual file transfers. File transfer traffic exceeds network traffic by several orders of magnitude, so to measure the fraction of the total bandwidth used by filesharing applications one can constrain the measurements to file transfer traffic.

Network traffic is generally hard to detect by conventional means: Classification by port numbers is obsolete because most filesharing networks allow clients to randomly choose any ports to accept connections so as to avoid being blocked by firewalls. Even packet insight would not help further since the information transmitted is being encrypted.

[1] see [Cisb]

Source IP Address and port: The IP address and TCP or UDP port number of the host the packets are originating from. Note that this not necessarily has to be the host that initiated the TCP connection.

Destination IP Address and port: The address and port of the host receiving the packets.

Packet Count: The total number of packets observed in this flow, including initial TCP handshake and zero-length packets like ACK packets.

Layer 3 Protocol: The transport layer protocol having sent the packets, for instance TCP, UDP or ICMP.

TCP Flags: Any TCP flags that have been set in any packet during the flow.

Start and End Time: The time when the first packet and the last packet respectively of this flow were recorded.

Additional Routing Information: Routing and peering information that is of little interest to the identification of P2P traffic.

Table 3.1: Cisco NetFlow logfile format

As for file transfer traffic, if the contents of the TCP packets transmitted are known, it is very easy to detect since most filesharing networks use HTTP with custom headers (e.g. `X-Kazaa-Username` in FastTrack) for transferring files, as shown in [SGD+02]. However, NetFlow does not provide any details about the content delivered so other ways of detecting file transmission traffic have to be developed. Due to the randomness of the ports used it is actually impossible to know whether a connection is a WWW download from a webserver with an odd port number or a filesharing download.

3.2.1 Approaches

A first approach used in this thesis is to try to detect P2P handshake patterns among NetFlow data. A handshake occurs every time a client newly connects to the network so by detecting handshake sequences, which normally have distinct patterns, a set of active hosts in this network could be created. If one of these known hosts starts a connection to another known host, one can assume that one is witnessing a filesharing transfer.

A second approach is to use commonly known ports of the filesharing systems (like **`TCP/1214`** for FastTrack) to determine a set of active clients on the net. Many hosts still use these well-known ports so every client connected to our observed network is bound to connect to some of these hosts over time. Thus detecting connections to port **1214** would render a set of active FastTrack clients in a specific period of time which would then in turn enable us to identify transfer traffic using other ports.

3.2.2 Hardware and Software

Most common clients run on Microsoft's Windows operating systems series, so one of the testing computers was a machine running Windows XP. ETH limits traffic on filesharing ports which could prevent realistic observation, so a branch line to the ISP SWITCH was being used to circumvent ETH's gateways. To protect the

Windows computer from the evil world and vice versa a second computer was inserted running Debian Linux with a firewall and ethereal for sniffing.

The clients installed on the Windows computer were:

- Kazaa lite K++ v2.4.3 build 1
- Kazaa Media Desktop v2.6
- eDonkey v0.52
- eMule v0.30e
- Overnet v0.52
- Morpheus v4.0.53.139
- Shareaza v1.8.10.8
- BearShare v4.3.5.4
- LimeWire v3.6.15
- BitTorrent v3.3
- Shadow's Experimental BitTorrent v5.8.8
- BitComet v0.41 beta
- iMesh v.4.2 build 140

3.3 Assayed Networks

As was shown in section 2.3 on page 4, about 90% of the whole P2P filesharing traffic is being generated by only four systems (FastTrack, eDonkey, Overnet and BitTorrent). To estimate the total traffic generated by filesharing it appears to be sufficient to identify these four systems within the Netflow data.

To simplify the description of the processes the following conventions are being used:

- *Client* always denotes the client program connecting to the network while *server* denotes the host the client is connecting to (ultrapeer, supernode, eDonkey server etc.), may it be for searching or file transfer.
- *Upward* direction ($\longrightarrow$) is from client to server, *downward* ($\longleftarrow$) from server to client.
- All packet lengths are Bytes of the IP total length including the IP header.
- Within tables describing connections TCP flags are denoted as **S** (SYN), **P** (PUSH) and **A** (ACK).

One problem that is common to all decentralized networks is that for bootstrapping the client needs to have a list of hosts that are part of the network. This set of host is generated by caching host addresses from previous sessions, by advertisement within the network and—for the first session—from lists that can be found on the WWW.

3.3.1 Gnutella

Basics

The Gnutella protocol is publicly available [gnu02] and well understood. However, the implementation of the different user agents varies, so only the most important can be covered herein, which would be BearShare and LimeWire. The default Gnutella port is **`TCP/6346`**.

GWebCache

For bootstrapping the Gnutella community has developed a web-based distributed host caching system called GWebCache or GWC. A list of currently connected ultrapeers can be retrieved by simply querying one of the many GWC servers, which will additionally return a set of several other GWC servers. The client has a few addresses of GWC servers hardcoded; after the first query it maintains a cache of GWC servers. For a detailed description see [GWe]. The main benefit of this method is the general availability of webservers (remember, GWC is web-based) opposed to the low availability of Gnutella clients.

There are many GWC servers, the ones queried during startup are fairly random and fluctuation of servers is medium to low. This might suggest that one could determine a set of GWC servers and treat every host issuing queries to one of them as a Gnutella client. However, most of these servers are running as virtual hosts on large webservers with one IP for many virtual domains, thus making it impossible to unerringly pin down GWC servers.

BearShare

Out of the set of known hosts nine are simultaneously probed with **`SYN`** packets with a timeout of approximately one second and two retries. If the connection succeeds the two hosts shake hands with a HTTP-like protocol. During the first packet sent to the ultrapeer variable length data is being sent, like a list of known hosts and an **`Accept-Encoding`** header. The server reply also varies in length due to varying headers.

Observed values for many BearShare connects are listed in table 3.2.

No	Dir	Flags	Length	Comments
1	$\longrightarrow$	**S**	48	
2	$\longleftarrow$	**S A**	48	TCP handshake
3	$\longrightarrow$	**A**	40	
4	$\longrightarrow$	**P A**	450-491 (avg. 459)	
5	$\longleftarrow$	**P A**	500-761 (avg. 661)	

Table 3.2: Example Gnutella handshake (BearShare)

By default BearShare keeps open connections to two ultrapeers with continuous traffic because Gnutella is broadcast based. This query traffic causes approximately 1600 Bytes in 21 packets per minute in either direction.

LimeWire

Basically LimeWire's startup process is very similar to that of BearShare, with some little differences. Instead of trying nine known hosts with **`SYN`** packets, LimeWire probes as many as 20 hosts. Prior to this the client may connect to GWebCache servers in order to retrieve a list of active Gnutella ultrapeers. It keeps querying GWC servers until it receives a sensible answer.

In other respects LimeWire behaves practically identical to BearShare, except it keeps alive four connections to ultrapeers by default as opposed to BearShare's default two.

3.3.2 eDonkey

Basics

The eDonkey protocol is proprietary, but good efforts have been made to reverse engineer it. The most comprehensive analysis probably is [Kli03], on which this section is mainly based on besides my own sniffing observations.

The two clients eDonkey and eMule behave very similar during startup as far as NetFlow relevant data is concerned.

Startup

Out of its set of known servers the client chooses two and simultaneously[2] probes them with **SYN** packets, three times with increasing timeouts of 3, 6 and 12 seconds. Then the next two servers are tried and so on, until a connection has been established. Unsuccessful connections thus result in three packets sent with a total of 144 bytes within approximately 21 seconds.

Upon a successful TCP handshake the eDonkey handshake follows: The client logs in with the server with a <**hello server**> message, which includes several headers, for instance the client's IP and port number, username, client type and version and compression options. This packet's size is variable and lies somewhere between 100 and 160 bytes.

The server now replies with a <**ID change**> message which assigns the client a client ID and can (and normally does) append variable numbers of headers of variable size, for instance, eMule's extensions to the protocol. This packet's size can range approximately between 100 and 800 bytes.

Now the client would normally request a server list, this packet weighs 46 bytes and is being answered by the server with a list of known servers, which has an approximate size of 160 bytes plus 6 bytes per included server info.

Table 3.3 summarizes these findings using an example connection.

No	Dir	Flags	Length	Comments	
1	⟶	**S**	48		
2	⟵	**S A**	48	TCP handshake	
3	⟶	**A**	40		
4	⟶	**P A**	108	<**hello server**>	range 100-160
5	⟵	**A**	40		
6	⟵	**P A**	654	<**ID change**>	range 100-800
7	⟶	**P A**	46	<**get server list**>	
8	⟵	**A**	40		
9	⟵	**P A**	551	<**server list**>	here: 65 servers
10	⟶	**A**	40		

Table 3.3: Example eDonkey2000 handshake

Ping/Pong

After successfully commencing the handshake the connection is open for search queries and control traffic, in particular ping challenges. In eDonkey these are

[2] In fact the second **SYN** is 0.5 seconds delayed.

called <**server status**> and are always initiated by the server. In the message included are the number of users currently connected to the server and the number of files these users share. Both fields are 4 bytes long, so the total packet size is always 54 bytes.

The client simply replies with an **ACK** packet. The <**server status**> message cycle is summarized in table 3.4. By default the server sends this message unsolicited every 410 seconds.

No	Dir	Flags	Length	Comments
1	⟵	**P A**	54	<**server status**>
2	⟶	**A**	40	

Table 3.4: Typical eDonkey2000 ping

Total Handshake

Assuming that no other traffic than the handshake and subsequent <**server status**> messages are being exchanged, one can calculate an estimation for the first flow generated by an eDonkey client startup. One <**server status**> every 410 seconds yields two cycles during the first flow of 15 minutes. Thus the first flow would have 7 packets with 370-430 bytes upwards and 7 packets with 556-1856 bytes downwards, assuming a server list of 10-100 entries.

3.3.3 FastTrack

Basics

Due to its proprietary nature the FastTrack protocol is not yet fully understood. The available reverse engineering documents [giF] prove unsatisfactory in explaining the ongoings during a FastTrack session. Attempts at cracking the FastTrack protocol have been made but have failed to break the encryption. Therefore I tried to detect traffic patterns during the initial handshake with Ethereal, which lead to quite usable results.

Originally FastTrack used port **1214**, but because many internet service providers have filtered this port the clients are now free to use any unfiltered port, even port **80** if necessary.

Handshake

To start up, a client first needs to know some IP addresses and port numbers of supernodes which are hardcoded into the source code of the client.

KaZaA, when started by the user, issues five UDP packets (40B each) to five different hosts out of its hardcoded and cached hosts list. These packets serve as pings to determine whether the host is willing to accept FastTrack connections on this port.

The client then tries to establish TCP connections in the order of the recieval of the ping replies. A typical handshake is shown in table 3.5 on the next page. The total handshake results in 8 packets (404B) upward and 7 packets (2065B) downward. This pattern occurs every time a client connects to a server, which occurs either at client startup or reconnection after losing the connection.

During startup the client opens the website `http://desktop.kazaa.com/` within the client window displaying news about the network and ads. Additionally several advertisement webservers are connected, including `ad.(countrycode).doubleclick.net`, `adfarm.mediaplex.com`, `kupgrade.bullguard.com`,

No	Dir	Flags	Length	Comments
1	⟶	**S**	48	
2	⟵	**S A**	48	TCP handshake
3	⟶	**A**	40	
4	⟶	**P A**	52	
5	⟵	**P A**	54	
6	⟶	**P A**	65	
7	⟵	**A**	1500	
8	⟵	**P A**	338	
9	⟶	**A**	40	
10	⟶	**P A**	68	
11	⟵	**A**	40	
12	⟶	**P A**	51	
13	⟵	**A**	40	
14	⟵	**P A**	45	
15	⟶	**A**	40	

Table 3.5: Typical FastTrack handshake

`BAdKey.adsolu.com` and `cydoor.com`. All these connections occur on server port **`80`**. Each single website may not hint to a Kazaa client, however the four of them coinciding is a unmistakable indication for a Kazaa client starting up.

Ping/Pong

After the handshake the connection is kept open with ping messages every 120 seconds as shown in table 3.6. Note that the direction in which the ping is being sent is not fixed upwards. The server can send ping messages as well, which renders them a bit more difficult to detect. A complete ping message sent by the client has 2 packets (81B) upward and 1 packet (41B) downward.

No	Dir	Flags	Length	Comments
1	⟶	**P A**	41	**`Ping?`**
2	⟵	**P A**	41	**`Pong!`**
3	⟶	**A**	40	

Table 3.6: Typical FastTrack ping

Total Handshake

NetFlow terminates a flow entry after fifteen minutes, which results in a handshake and seven ping/pong exchanges if the connection stays idle. This results in 22 packets (971B) upward and 14 packets (2352B) downward for the first flow and (assuming 7 ping/pong exchanges per fifteen minutes) 14 packets (567B) upward and 7 packets (287B) downward during consecutive flows, always assuming that the client initiates all ping challenges.

3.3.4 Overnet

Basics

The Overnet protocol, based on Kademlia [MM02], is proprietary and relatively young, so there is no comprehensive reverse engineering known yet. One important fact is easily observable: The Overnet protocol makes extensive use of UDP as

opposed to most other filesharing systems. Since the program used by DDoSVax for sniffing and generating NetFlow data does not aggregate UDP streams[3], this can be a major benefit when it comes to identifying Overnet. The standard Overnet port is **`3575/UDP`**.

Handshake

During startup the Overnet client opens websites on both `sda.edonkey2000.com` and `home.overnet.com` (which are on the same server, but nevertheless two connections are established).

Analysis of the UDP "flows" was done using the tool **`udp_flows`**[4].

The actual connection process starts with three UDP packets of length 53 bytes to three different hosts known from previous sessions ore hardcoded into the client. After five seconds without receiving a reply these packets are supposed to have timed out and another three packets are sent. However, these are not resent to the previous hosts but to a new group of hosts. This is being repeated until a reply has been recieved, which can take rather long (average during measurements was around one minute resulting in as much as 150 UDP packets sent without a reply).

A host responding to one these requests is answering with a packet of variable length around 538 bytes. This is reckoned to be the retrieval of a list of active hosts, similar to the GWebCache. There is no further communication between these two hosts.

Now client behaviour changes to sending a packet of length 32 to three new hosts. A host responding replies with two packets with 34 bytes and 30 bytes respectively.

After this setup connections become more random. There are, however, some patterns that can be seen rather often. These are summarized in table 3.7, including frequently occuring startup patterns.

upstream		downstream	
Packets	**Bytes**	**Packets**	**Bytes**
1	53	0	0
1	32	2	64
1	47	0	0
1	47	2	507
9	471	4	597
10	524	4	597
2	143	2	98
2	186	2-3	179-350+

Table 3.7: Typical Overnet UDP patterns

In general a host running Overnet has UDP "connections" to rather many hosts. Excluding unsuccessful connections attempts my test client has had connections to an average of 80 hosts during the first four minutes.

3.3.5 BitTorrent

BitTorrent has no overlying network, thus clients do not have to register with it. Communication between tracker and clients, as well as between clients, is based on HTTP, although not on port **`80`**. Normally trackers would listen on port **`6969`**, clients on the port range **`6881-6900`**, although about 90% of the traffic on these

[3] There *are* no UDP streams. Every UDP packet is a single flow.

[4] See appendix B on page 39 for details.

ports is generated on ports **6881-6886**, as shown in figure 3.1. The amount of trackers covered by Suprnova [sup] is only around 2000 [Men], but fluctuation is heavy and the same problem as described in section 3.3.1 on page 14 arises, namely the problem that most domain websites reside on virtual hosts that can not be distinguished, because they use the same IP.

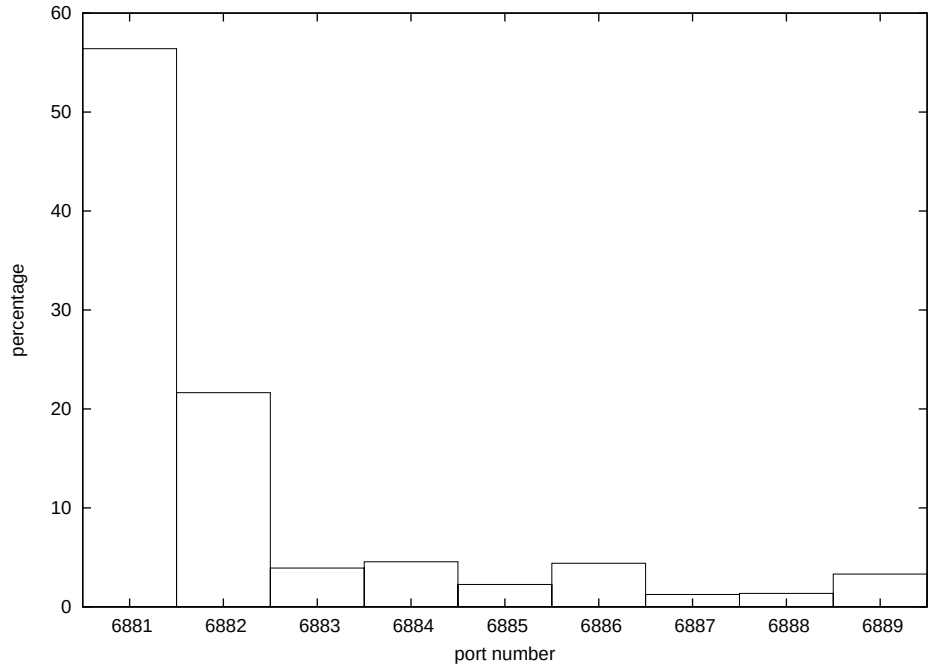


Figure 3.1: BitTorrent port usage statistics extracted from the data used for the graphics in appendix A on page 27.

It seems that there is no reliable approach on identifying BitTorrent traffic than observing traffic on it's default ports.

Chapter 4

Identification in Real World Data

Using the results from the previous chapter we will develop and verify methods for the identification of filesharing traffic within NetFlow data.

4.1 Examination of Real World Traffic

To gain a better understanding of the traffic patterns generated by filesharing systems, I have started my real world analysis with the observation of this traffic. This was accomplished by assuming that a major amount of traffic on the default ports of the filesharing networks is generated by exactly these networks. So looking at the traffic patterns occurring on these ports might lead to a better understanding of the networks' traffic patterns.

I randomly picked out any day, namely November 21, 2003. There is nothing special about this date[1] and the traffic boost of the worms Blaster and Sobig.F date back more than a month. [WD03b, WD03a] Thus the traffic observed on this day should be rather average.

The calculations were conducted on a dual processor Athlon MP 2800+ with 2GB RAM running Debian Linux. The NetFlow logfiles are split into hours, for each hour the first 100 million TCP flows[2] are categorized using the port numbers shown in table 4.1. Note that for reference WWW traffic is also being observed, as well as the total traffic itself.

BitTorrent	6881-6889 (clients), 6969 (tracker)
eDonkey	4661, 4662
FastTrack	1214
Gnutella	6346
WWW	80, 443 (https), 8080 (proxy)

Table 4.1: Default port numbers for the observed services.

Processing the first 100 million flows of an hour takes about 40 minutes, hence there is no aggregation of correlating flows in both directions nor a combination of consecutive 15 minute bits. The complete collection of sheets generated can be found in appendix A on page 27.

[1] After researching a bit to consolidate this I can assure you that the most important event on November 21 probably was North Carolina's ratification of the Constitution in 1789, making it the twelfth state in the Union. [LoC]

[2] All other types like UDP or ICMP are not considered.

4.2 Identification via Traffic Patterns

If client startup patterns were distinctly defined with unvarying packet lengths and if clients would not produce any query traffic during the first 15 minutes after startup we would clearly see high peaks in the graphs in appendix A on page 27.

The highest peaks are always found at 48 bytes, which are clients unsuccessfully trying to establish connections with **SYN** packets, and at 40 bytes, which are **RST** or **ZeroWindow** packets indicating a host rejecting an incoming TCP handshake request.

The FastTrack startup is rather strict concerning packet lengths, so this will be the network we will begin our identification attempts with. Recall from section 3.3.3 on page 17 the startup pattern including the first 15 minutes' ping messages: 22 packets with 971 bytes upwards and 14 packets with 2352 bytes downward.

4.2.1 Example: FastTrack Startups

I will depict the steps in obtaining the number of FastTrack clients connecting to the network. We will use a logfile slice consisting of approximately 19 million flows, converted to a human readable cooked format using **netflow_to_text**. Extraction of data is done using **grep**, **egrep** and **wc -l**.

First count the number of TCP flows exactly matching our criteria of 971 bytes in 22 packets and 2352 bytes in 14 packets respectively. Result: 4 flows upstream and 2 flows downstream. Dissatisfactory.

In an attempt to broaden the results we try to include clients using custom headers that would change the startup pattern. Adding a custom header adds from 10 to 100 bytes so we can assume that no packets become bigger than 1500 bytes, thus the total number of packets will remain the same. Extracting TCP flows with 900-1100 bytes in 22 packets and 2200-2499 bytes in 14 packets yields 240 upstream and 465 downstream flows.

Analyzing the ports used shows that 43 upstream and 114 downstream flows connect to the WWW ports 80 and 443. Checking the IPs with **wget** shows that every one of these servers actually has a running webserver running on ports 80 and 443 respectively, which renders them unlikely to have running filesharing clients on these ports. This leaves 197 up- and 351 downstreams left.

Further analysis reveals that 82 up- and 24 downstream connections connect to one single host, mostly on port 6667, which is a well-known port used by Internet Relay Chat (IRC). A short research shows that this host is indeed publicly known to have an Undernet IRC server running on port 6667, so its flows can be omitted.

We only want to extract hosts that stay on for at least 14 minutes, because only then the full volley of both handshake and seven ping/pong exchanges can be observed, resulting in the postulated number of packets. So we can dismiss flows with a duration of less than 14 minutes (840 seconds). Result: disappointing zero flows for both up- and downstream. The longest upstream is around 400 seconds, down less than 200 seconds.

This result is quite surprising but nevertheless reproducible using other logfiles. Analyzing the complete logfile shows that only about 19 thousand flows out of our 19 million flows have a length greater than 800 seconds, which is itself rather astonishing. Since the recording router is supposed to terminate flows after 15 minutes (900 seconds) one would reasonably expect a large portion of flows with an exact—or at least approximate—length of 900 seconds. However, only two thousand flows end between 890 and 910 seconds. A possible explanation is the fact that the router records the time of the first and last packet in this flow. If a packet would exceed the limit of 900 seconds, it starts a new flow. Yet the preced-

ing packet could have arrived quite some time before the one causing the router to start a new flow. Thus the total time of the flow is unlikely to be 900 seconds but less. Funny enough there are a few flows that even exceed 1000 seconds, the top value observable seems to be 1800 seconds, which is twice the limit. I have no sensible explanation for this, except maybe a faulty software implementation in the router.

The other filesharing networks lead to even worse results due to the larger variation of the handshake flow lengths.

4.2.2 Summary

Altogether the approach of identifying startup patterns seems not to work without developing algorithms using heuristic observation, which would definitely not fit into this Bachelor's thesis.

4.3 Identification via Default Ports

Filesharing applications create a considerable amount of traffic, so many ISPs, companies and universities are restricting the use of default filesharing ports or block them altogether. So resourceful developers make newer clients try random ports until they find a suitable one. If everything else fails, they may even use port **`80`** which is basically never blocked. But by default the clients still use their inherent default ports, which works fine for hosts not firewalled.

4.3.1 Connections per Hour

It is hard to come up with statistics of the port usage of clients, since most known research was based on data gathered on specific ports. Let us cheekily assume for now that about half of the clients are using default ports.

A standard client opens and maintains connections to one to five other clients or servers for query traffic. These connections can theoretically last very long, from several hours up to weeks. More fluctuation in connections can be observed during file transfers. These should occur regularly—after all, the main purpose of filesharing is sharing files—and imply short connections lasting seconds to minutes to one to twenty or even more hosts.

Assuming that clients always have at least one download or upload running, one can assume that each client connected to a filesharing network has at least ten connections to unique hosts each hour, out of which presumably five use default ports.

4.3.2 Algorithms

For a short description of programs used please refer to appendix B on page 39.

Active Hosts

The algorithm for retrieving a set of active hosts is described below.

1. Extract a slice of a logfile with **`netflow_extract`**.
2. Convert the extracted log to parseable format using **`netflow_to_text`**.
3. Initialize a hashtable with IP addresses versus number of flows.
4. Iterate over the log:

- Filter flows using a default port with **`egrep`**.
- Increment the hashtable entries for both source and destination address.

5. Drop hosts having less than five connections.
6. Write the list of IP addresses to a file.

Flows within the Network

The algorithm for finding all flows within a network using the list of active hosts from the previous section:

1. Read the list of hosts.
2. Initialize a hashtable with keys {smaller address, greater address}.
3. Iterate over the logfile:
   - Check whether the source *and* destination IP address are listed in the set of known hosts.
   - Add number of bytes and packets to the coresponding entry in the hashtable.
4. Output all flows to a file.

4.3.3 Verification

For proof-of-concept these two algorithms were implemented in **`perl`**. Applying the algorithms to a one hour logfile we obtain table 4.2 showing the fraction of bytes consumed by each filesharing network with varying thresholds.

	Threshold					
Network	**1**	**3**	**5**	**10**	**20**	**obs.**
BitTorrent	2.99	2.92	2.74	1.59	0.98	3.83
eDonkey	3.07	2.62	2.24	1.25	0.38	2.16
FastTrack	0.30	0.22	0.20	0.14	0.10	0.38
Gnutella	0.11	0.06	0.05	0.04	0.00	0.09

Table 4.2: Bandwith statistics created with **`hosts`** and **`flows`**. Values are percent of total bytes transferred. **Obs.** shows observed values as found in appendix A on page 27.

It can be seen that the calculated values are very close to those observed in appendix A on page 27. However, the figures for FastTrack seem unreasonably low, presuming it confines more than 4 million simultaneous users. One possible explanation for FastTrack to be harder to detect than the other networks is the demographics of FastTrack. Its users are mainly home or business users that often reside behind firewalls and/or network address translation (NAT). A firewall blocking port **`TCP/1214`** causes the client to search for other open ports, eventually evading to port **`80`**. A NAT box causes the port visible to the outer world to be another than the one visible to the private network. So both cases render the port to be randomly chosen and not trivially recognizable.

En contraire, most BitTorrent and eDonkey users are users more proficient with computers, often accessing the Internet via DSL, cable modem or broadband lines without blocking firewalls or NAT. Hence these networks show a higher percentage of clients using default ports.

Chapter 5

Conclusion and Outlook

5.1 Results and Future Research

This thesis was not intended to solve the problem of identifying filesharing traffic. The main goal was—as the title suggests—the characterization of today's filesharing networks and the traffic being generated by them. A secondary goal was to survey the current P2P filesharing usage and to document technical details of the most important and promising systems.

The survey and comprehensive analysis of filesharing systems precisely describe the ongoings during client startup and can be a useful resource when analyzing traffic with information about single packets being sent, or at least the first few packets.

The identification of filesharing traffic itself turned out to be more complex than assumed. The identification using startup patterns failed mostly, but could be enhanced by developing heuristic approaches utilizing long term observations. Additionally the startup patterns of clients not covered in this report would have to be revised for including statistical deviations of flow lengths also measurable by long term observations. A refinement to the startup pattern recognition could also include observing the advertisement webservers that are normally connected during startup.

Identification via observing hosts communicating on default ports, however, proved to be rather accurate for some of the systems while still demanding development for other systems. Advancement could be achieved by observing longer timeframes to gain a larger set of hosts connected to a network. When observing larger timeframes a timeout mechanism for expunging outdated clients would have to be implemented.

One could try to determine all active GWebCache servers or BitTorrent trackers, which would achieve very accurate results, if accomplished. The same applies for instance for eDonkey servers.

By developing a custom client that constantly connects and disconnects to certain networks a large number of active hosts could be actively retrieved in short time. Yet, large numbers of hosts can only be accessed when sharing or downloading files which puts legal manacles onto this approach.

The whole process of large scale identification of filesharing traffic would be a whole lot easier if NetFlow data would contain more information about single packets being sent. But as an accounting system it was never meant to measure much more than throughput and is designed to create as little burden on routers as possible, so there must not be expected too much improvement in this direction.

No attempt has been made on identifying Kademlia or Overnet traffic. For this one could create a module aggregating TCP "connections" and apply a pat-

tern search using the results found in this thesis. This approach appears to be quite promising, given that the NetFlow logfiles used by DDoSVax contain non-aggregated single UDP packets. One could even simulate the network with stateful models of clients based on observed data.

By combining traffic patterns of hybrid clients that connect to more than one network a further refinement could be achieved.

5.2 Acknowledgments

First of all I would like to thank my supervisors Arno Wagner and Thomas Dübendorfer for their open door—you don't need weekly meetings if you can just drop by twice a week whenever you have a question.

A great resource for information about P2P systems is `slyck.com` [Slya], including its forums.

Big thumbs-up to all people answering my many questions and spending lots of time chatting about pros and cons of the filesharing protocols in several webforums, newsgroups and on IRC.

Thanks to my folks on ICQ who made me remember that there still is a world outside *The Dungeon*[1]. And thanks to my fellow students in *The Dungeon* who made me think that this place is not half as far below the surface as it seems.

Last but not least thanks a lot to Graziella Spizzi for proof-reading this report *over night*, Lukas Ruf for his great LaTeX templates [Ruf] and my computer for not causing a petty average by conking out one day before submission.[2]

[1] *The Dungeon* is the nickname of one of the student working rooms, because it is located in the basement and has an ambiente close to that of its nickname.

[2] The last third of this sentence is ironic. The rest is not.

Appendix A

Real World Traffic Sheets

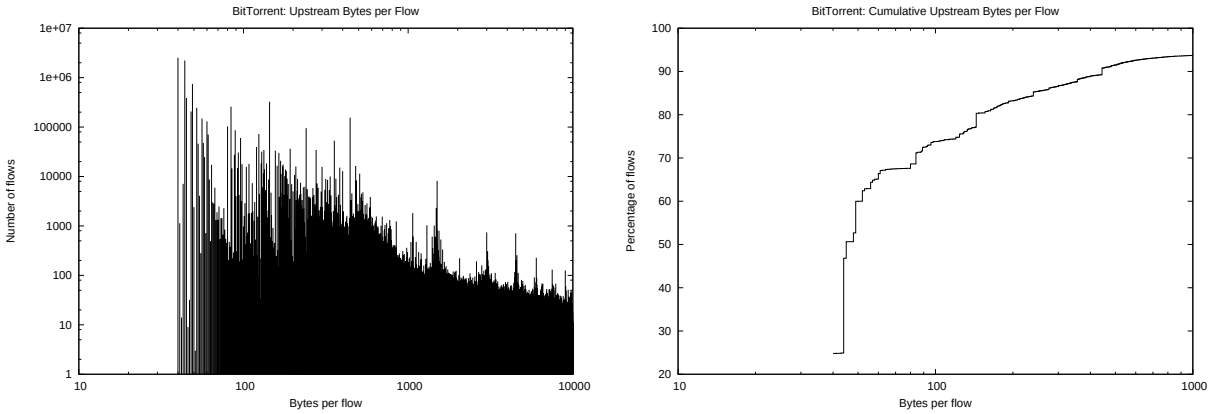
- Only TCP traffic is being observed, so percentages actually represent the fraction of the total TCP traffic, not the total overall traffic.
- Only approximately the first one hundred million flows of each hour have been observed.
- Cumulative sheets are read like this: “xx percent of the network’s traffic has less or equal this number of packets/bytes.”
- The host using one of the default ports is considered the server; *upstream* means from client to server, *downstream* from server to client.
- A small part of the traffic observed is non-filesharing traffic that incidentally uses one of the filesharing networks’ ports.
- An even smaller fraction is transit traffic and is counted twice, though this should not have major influence on the appearance of the charts.

Network	Bytes [G]	rel. [%]	Packets [M]	rel. [%]	Flows [M]	rel. [%]
BitTorrent	631.88	3.83	662.86	5.46	16.34	0.98
eDonkey	356.09	2.16	268.50	2.21	71.14	4.28
FastTrack	62.83	0.38	63.09	0.52	2.60	0.16
Gnutella	30.98	0.19	44.83	0.37	3.41	0.21
WWW	2’085.50	12.65	1’210.91	9.98	310.89	18.71
Total	16’483.59	100.00	12’132.56	100.00	1’661.88	100.00

Table A.1: Summary of Gathered Real World Data
Timeframe: 2003-11-21 07:00 – 2003-11-22 00:00

A.1 BitTorrent

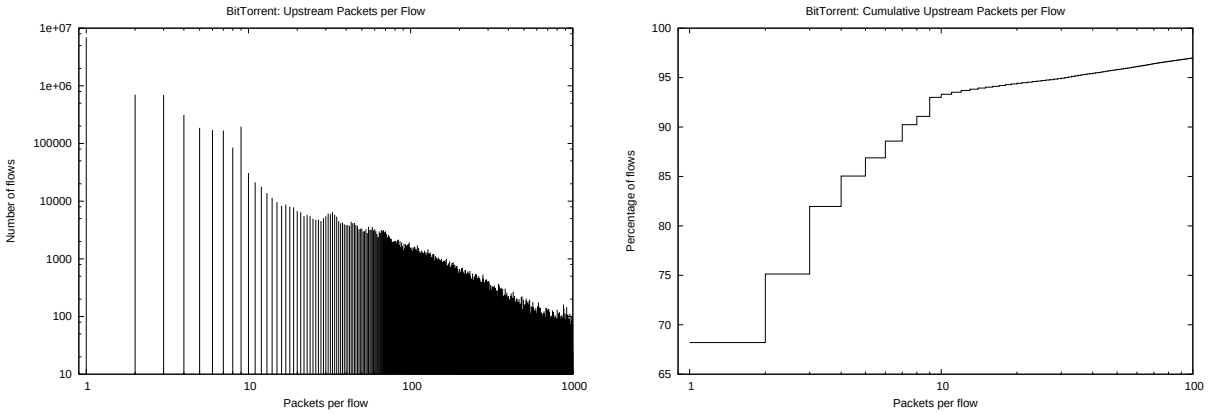
Bytes Upstream



(a) single (b) cumulative

Figure A.1: BitTorrent: Upstream Bytes per Flow

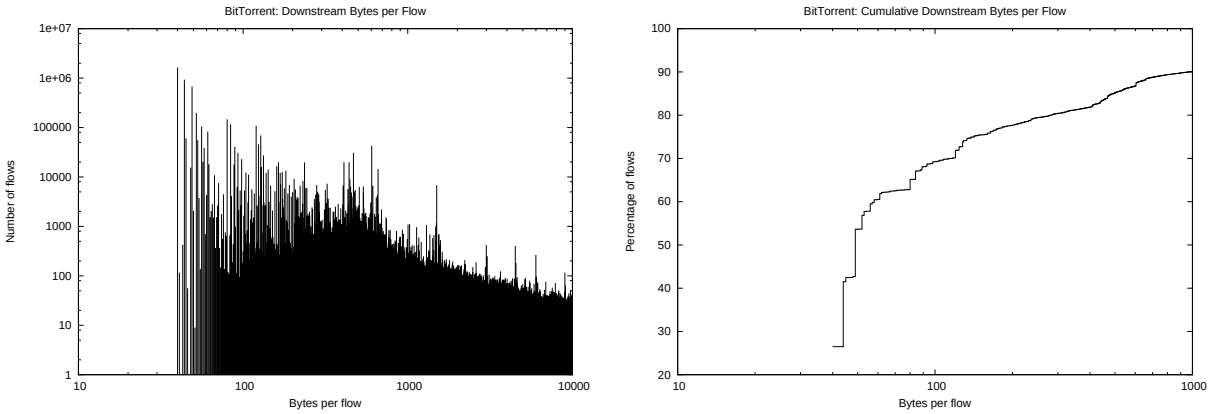
Packets Upstream



(a) single (b) cumulative

Figure A.2: BitTorrent: Upstream Packets per Flow

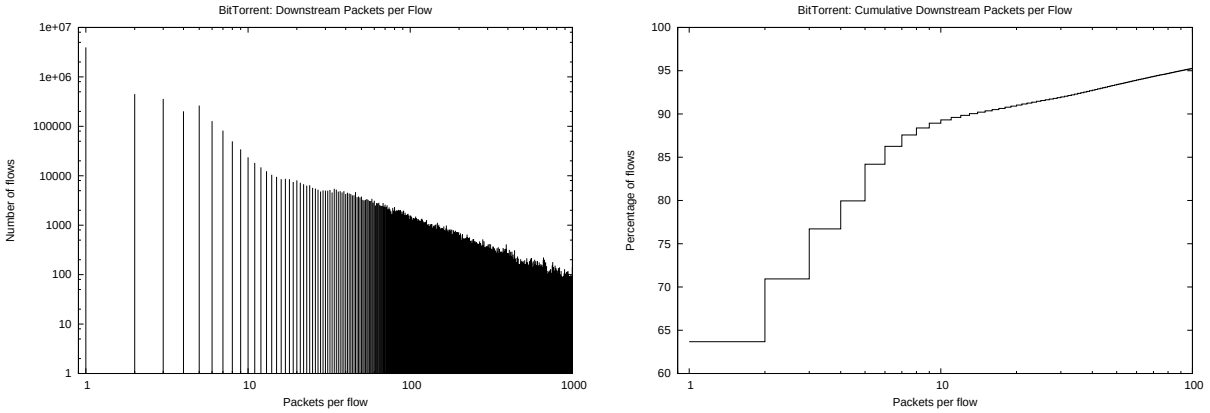
Bytes Downstream



(a) single (b) cumulative

Figure A.3: BitTorrent: Downstream Bytes per Flow

Packets Downstream

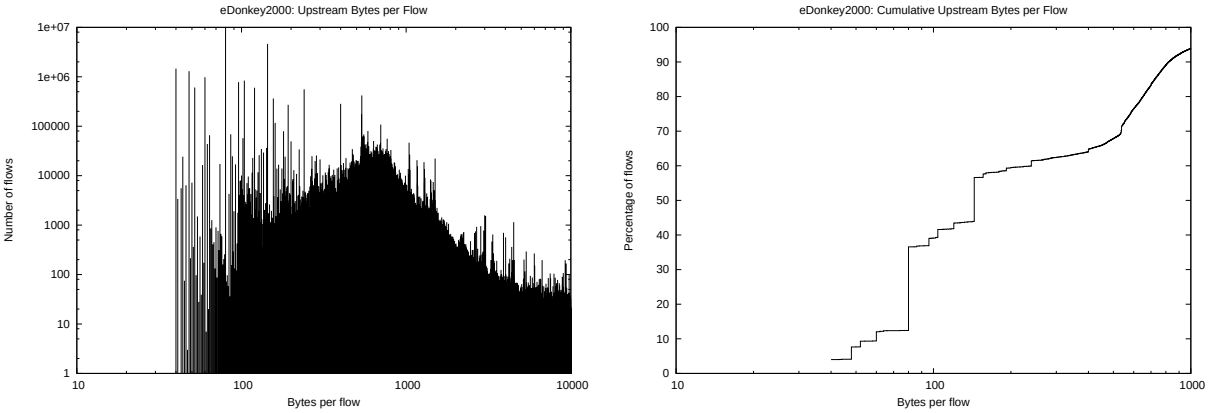


(a) single (b) cumulative

Figure A.4: BitTorrent: Downstream Packets per Flow

A.2 eDonkey2000

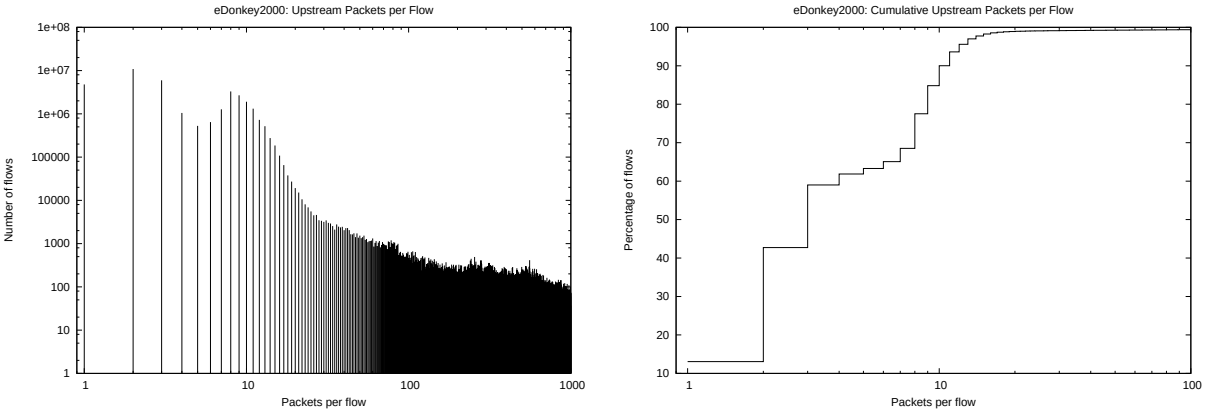
Bytes Upstream



(a) single (b) cumulative

Figure A.5: eDonkey2000: Upstream Bytes per Flow

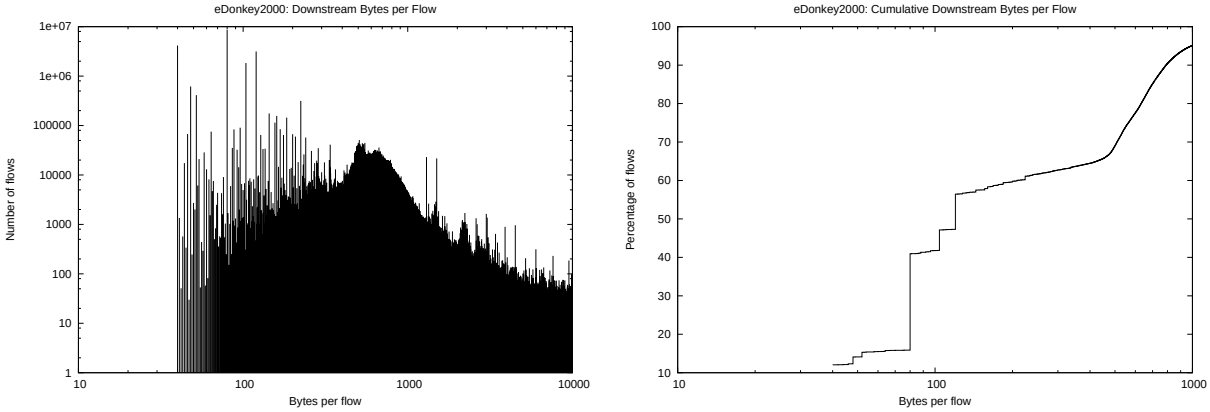
Packets Upstream



(a) single (b) cumulative

Figure A.6: eDonkey2000: Upstream Packets per Flow

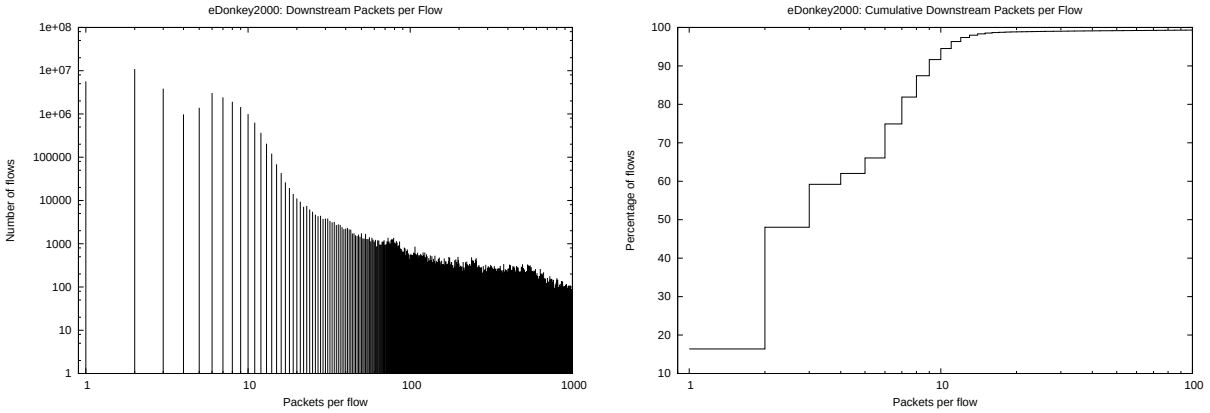
Bytes Downstream



(a) single (b) cumulative

Figure A.7: eDonkey2000: Downstream Bytes per Flow

Packets Downstream

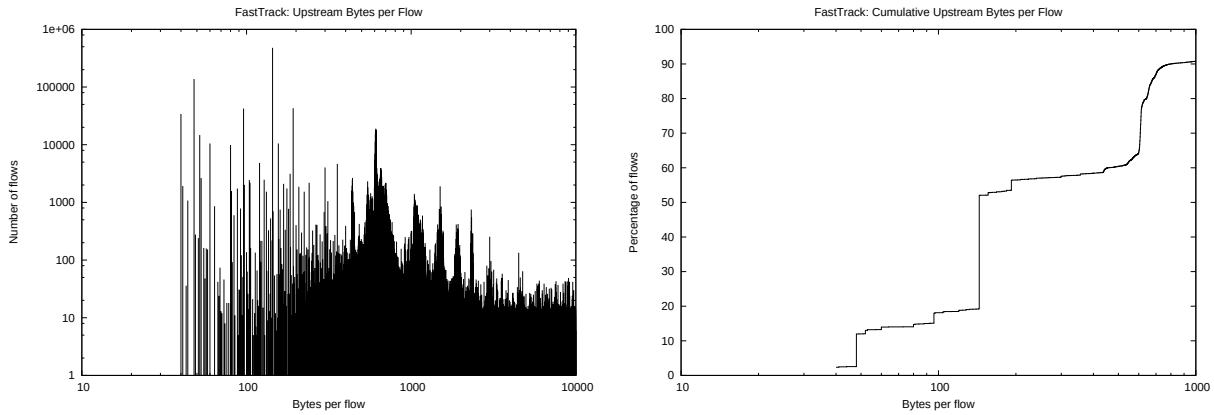


(a) single (b) cumulative

Figure A.8: eDonkey2000: Downstream Packets per Flow

A.3 FastTrack

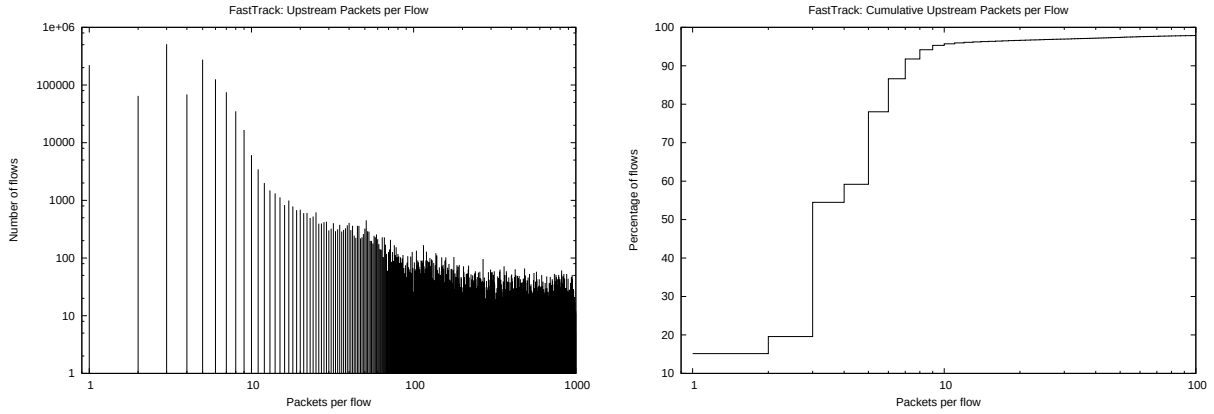
Bytes Upstream



(a) single (b) cumulative

Figure A.9: FastTrack: Upstream Bytes per Flow

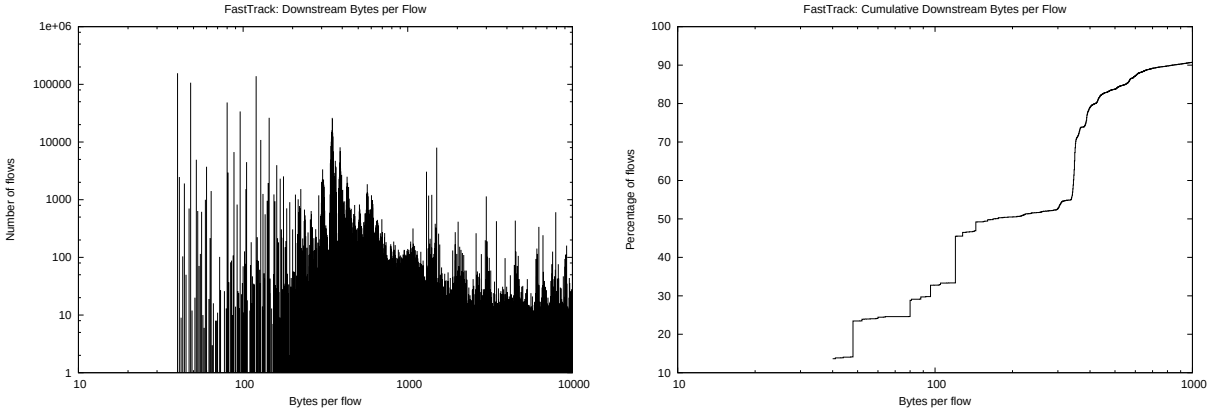
Packets Upstream



(a) single (b) cumulative

Figure A.10: FastTrack: Upstream Packets per Flow

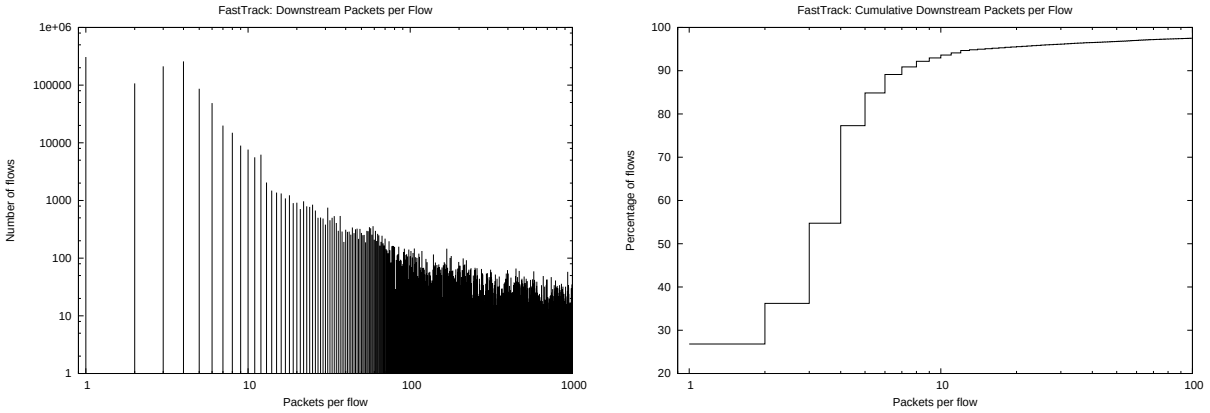
Bytes Downstream



(a) single (b) cumulative

Figure A.11: FastTrack: Downstream Bytes per Flow

Packets Downstream

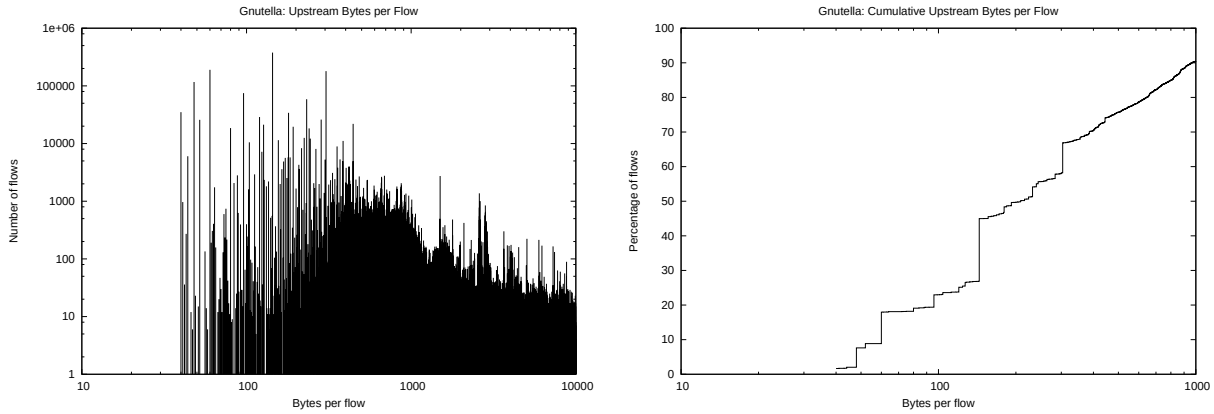


(a) single (b) cumulative

Figure A.12: FastTrack: Downstream Packets per Flow

A.4 Gnutella

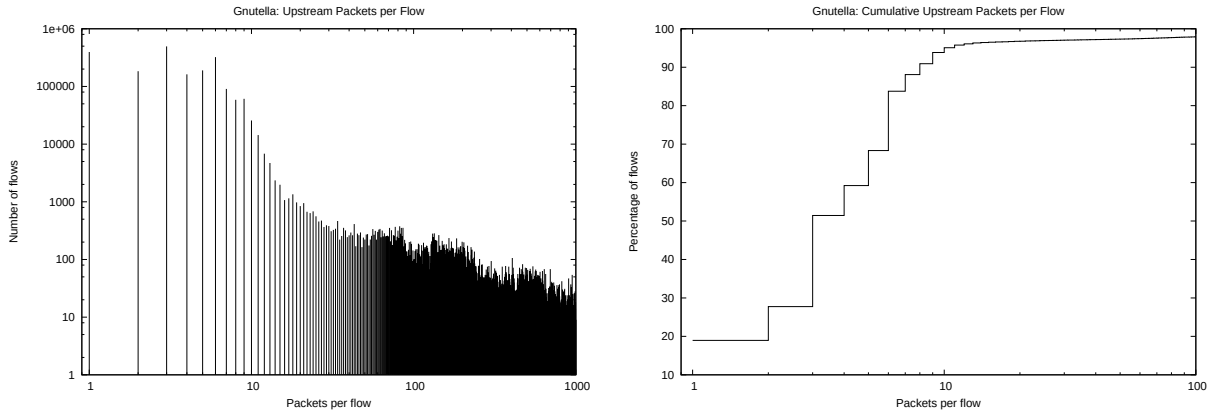
Bytes Upstream



(a) single (b) cumulative

Figure A.13: Gnutella: Upstream Bytes per Flow

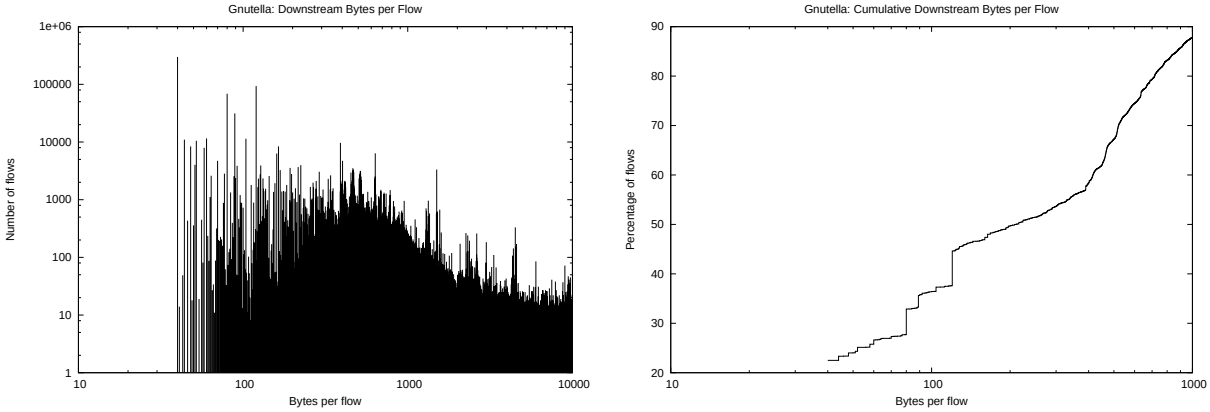
Packets Upstream



(a) single (b) cumulative

Figure A.14: Gnutella: Upstream Packets per Flow

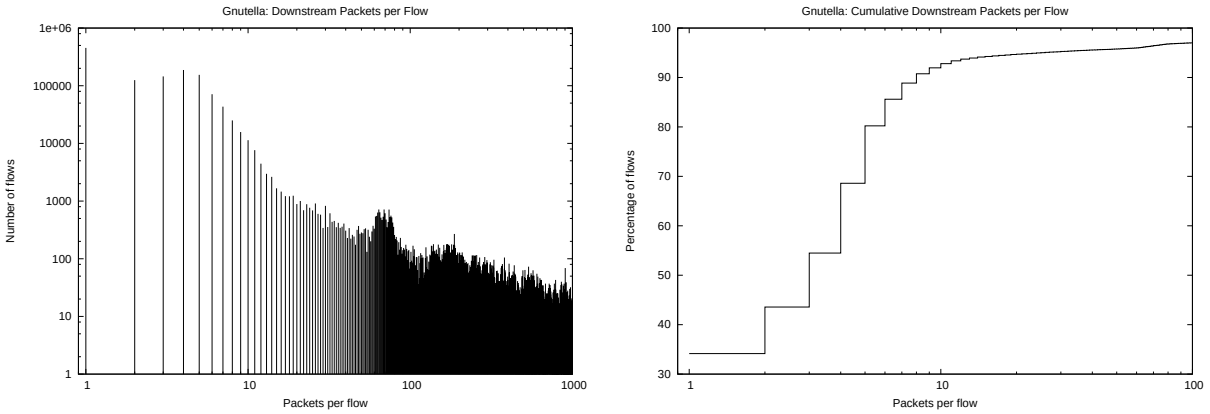
Bytes Downstream



(a) single (b) cumulative

Figure A.15: Gnutella: Downstream Bytes per Flow

Packets Downstream

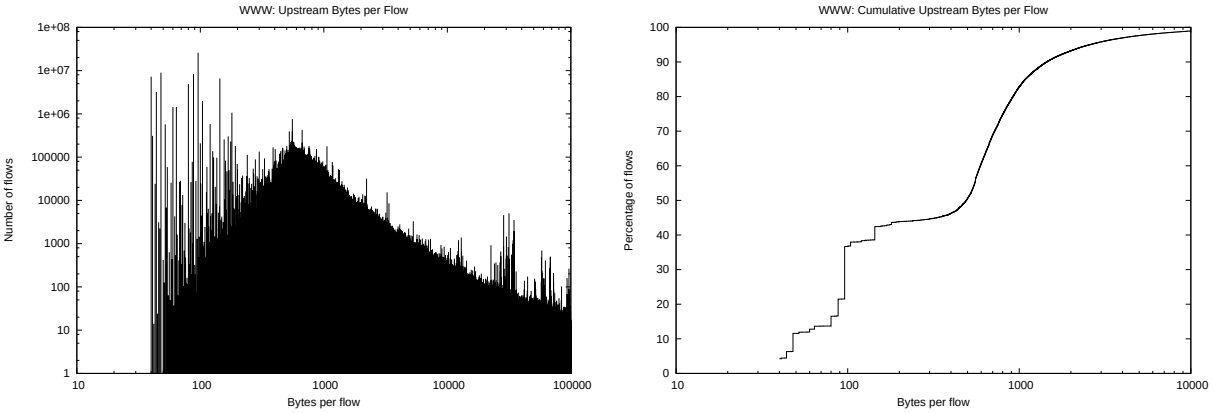


(a) single (b) cumulative

Figure A.16: Gnutella: Downstream Packets per Flow

A.5 WWW

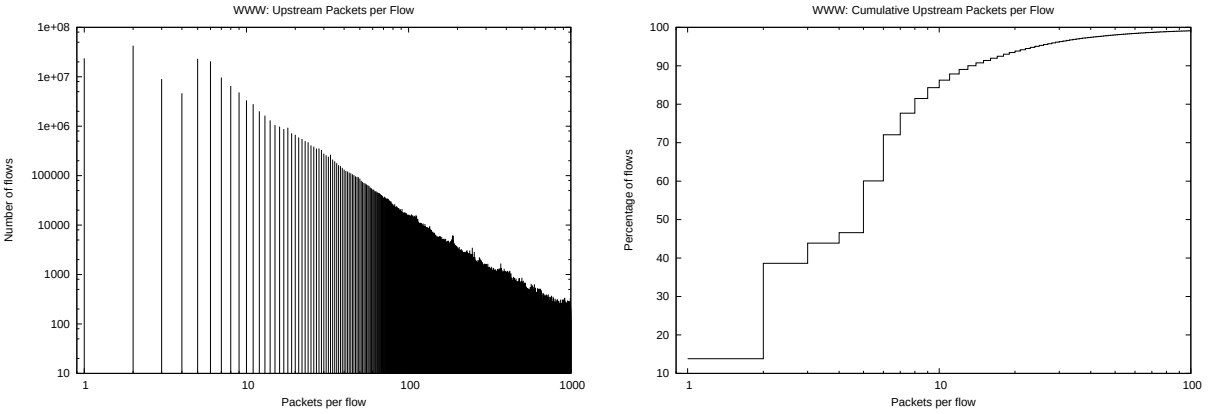
Bytes Upstream



(a) single (b) cumulative

Figure A.17: WWW: Upstream Bytes per Flow

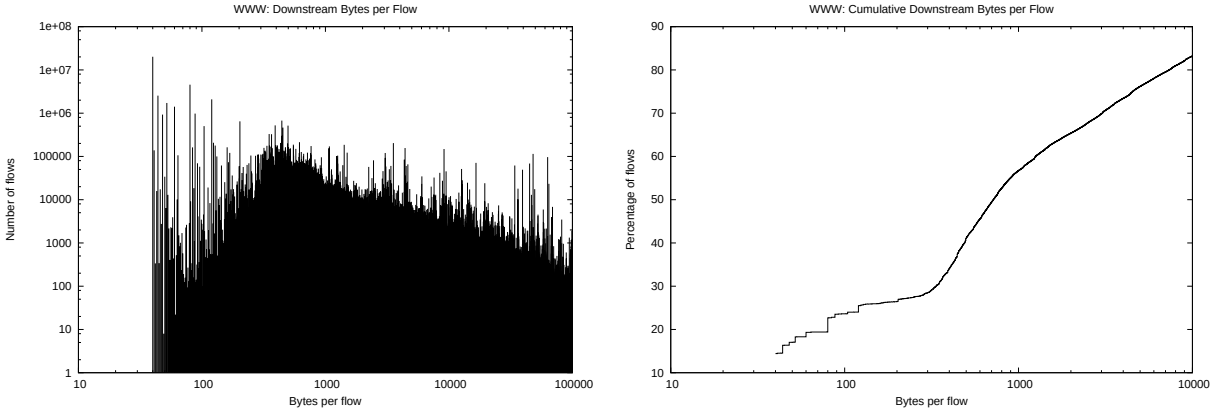
Packets Upstream



(a) single (b) cumulative

Figure A.18: WWW: Upstream Packets per Flow

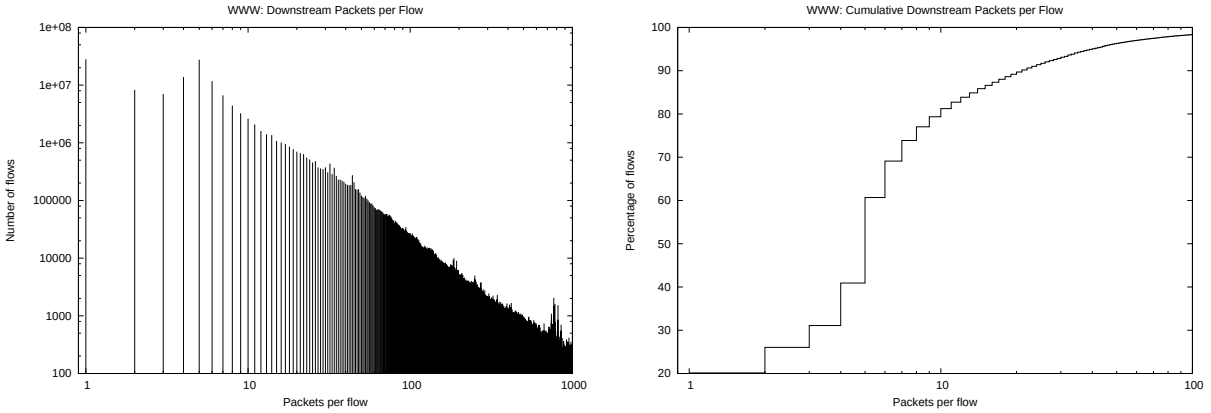
Bytes Downstream



(a) single (b) cumulative

Figure A.19: WWW: Downstream Bytes per Flow

Packets Downstream



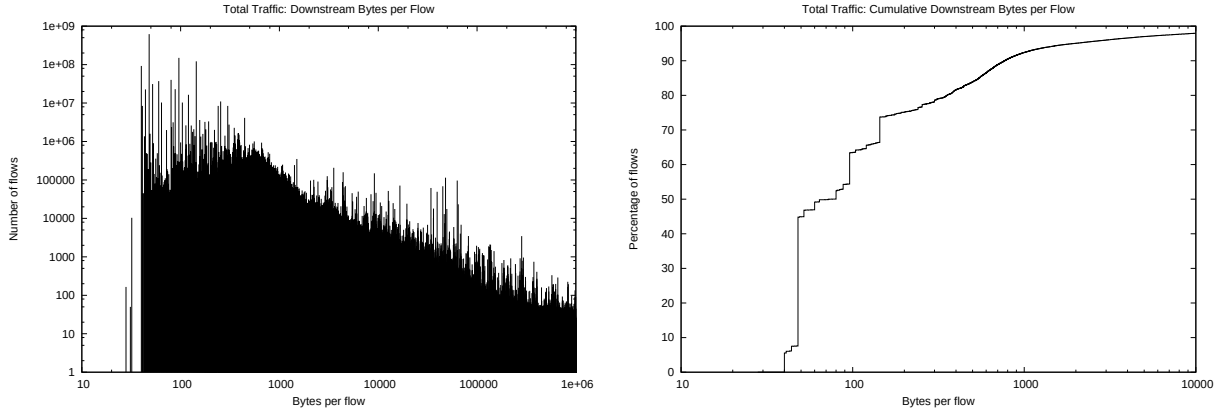
(a) single (b) cumulative

Figure A.20: WWW: Downstream Packets per Flow

A.6 Total Traffic

Note: There is no distinction between upstream and downstream in total traffic statistics.

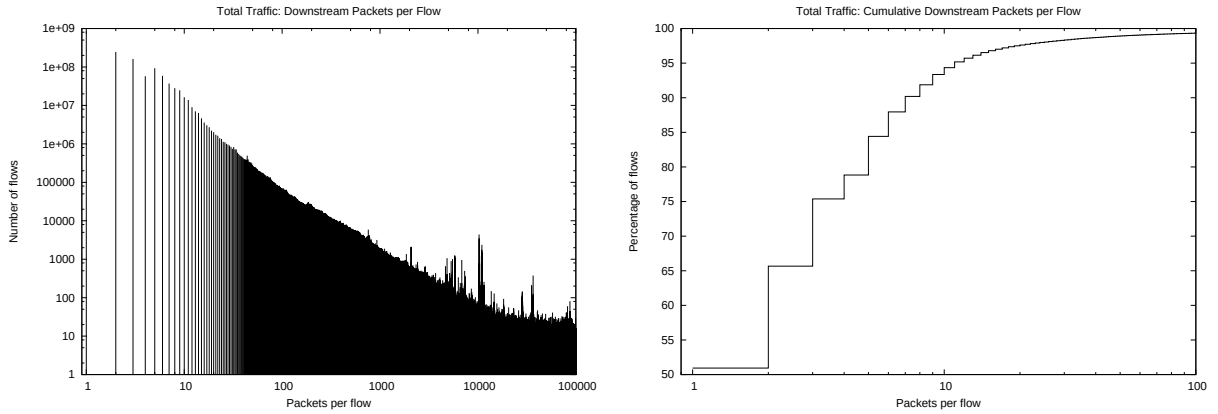
Bytes



(a) single (b) cumulative

Figure A.21: Total Traffic: Bytes per Flow

Packets



(a) single (b) cumulative

Figure A.22: Total Traffic: Packets per Flow

Appendix B

Used Software

This appendix contains a compilation of software I have been using during this thesis. I will not thank every author separately, which implies that I am grateful to all of them.

B.1 Existing Software

NetFlow Tools

These tools have been written in C by my supervisor Arno Wagner.

`netflow_head` Extracts the first x flows of a NetFlow logfile. Very useful for testing, since you do not have to let your script test run over 4G of bzip2'ed data.

`netflow_to_text` Since NetFlow data is binary, a tool is needed that converts this data to human readable form, which is exactly what this tool is doing. Data is exported in fixed width columns with two-letter postfixes like "**`TCP pr 1.2.3.4 si ...`**" for "protocol: TCP, source IP address: 1.2.3.4 . . . ". This can easily be used for **`grep`**'ing specific flows.

General Tools

An assorted compilation of common software used:

`ethereal` and `tcpdump` Network sniffers used for snooping and validating file-sharing protocols.

LaTeX Typesetting system used to create this documentation.

`gnuplot` Plotting tool used to create all plots within this documentation.

`perl` Scripting language used to do all processing of logfiles and wrapper scripts. See section B.2 on the following page for detailed information on scripts written.

`vim` Text editor that probably has one of the steepest learning curves of all editors out there. . .

`ICQ` You say there actually does exist a world out there?

B.2 Developed Software

I have written several tools to process NetFlow data and gather statistics, all written in **perl**.

make_links Reads a directory containing NetFlow logfiles and creates symbolic links with reasonable filenames (date and time).

netflow_extract Successively calls **netflow_head** and **netflow_to_text** on it's output. The human (and **perl**) readable file is gzip'ed and ready for further processing.

batch_extract Wrapper script for **netflow_extract**. Extracts different amounts of flows from many logfiles.

btports Reads a logfile and creates a statistic about the port usage of BitTorrent ports.

isSWITCH Checks whether a given IP lies within the SWITCH network.

hosts Extracts a list of hosts potentially connecting to a filesharing network by analyzing default ports.

flows Extracts all TCP flows that run between hosts both being part of a list created with **hosts** and creates a statistic about the filesharing network's bandwidth usage.

stat Runs through a logfile and creates statistics about traffic running on default ports. Used to create the data for the plots in appendix A.

plot Wrapper script for **gnuplot** that automagically creates all plots found in appendix A using data generated by **stat**.

netflow_stat The super-wrapper script for nearly all of the above scripts. Extracts many logfiles, runs both the port statistics as well as the IP set analysis on it and creates a cumulative statistic. Start this script, go home and come back in a week to see if it has already finished. The ultimate way to keep your computer busy for some time. . .

udp_flows Aggregates UDP packets in **tcpdump** files that obviously belong together. Used for the analysis of Kademlia and Overnet.

Bibliography

[Bea] BearShare. http://www.bearshare.com/.

[Ber03] Marcus Bergner. Improving performance of modern peer-to-peer services. Master's thesis, Umeå University, June 2003.

[Bita] BitComet. http://www.bitcomet.com/.

[Bitb] BitTorrent. http://www.bitconjurer.org/BitTorrent/.

[Cisa] Cisco Systems. *Cisco IOS NetFlow (PowerPoint Presentation)*. http://www.cisco.com/warp/public/732/Tech/nmp/netflow/docs/what_is_net%flow.pdf (Jan 5, 2004).

[Cisb] Cisco Systems. *Cisco NetFlow*. http://www.cisco.com/go/NetFlow/.

[Cli00] Clip2 Distributed Search Services and The Gnutella Developer Forum. *The annotated Gnutella Protocol Specification v0.4*, 2000. http://rfc-gnutella.sourceforge.net/developer/stable/ (Dec 20, 2003).

[Coh03] Bram Cohen. Incentives build robustness in BitTorrent. http://bitconjurer.org/BitTorrent/bittorrentecon.pdf (Feb 2, 2004), May 2003.

[EDi] Ecommerce dictionary. http://www.ecommerce-dictionary.com/ (Jan 3, 2004).

[eDo] eDonkey. http://www.edonkey2000.com/.

[eMu] eMule. http://www.emule.com/.

[GDN] Gnutella2 developers' network. http://www.gnutella2.com/.

[giF] giFT-FastTrack. *The FastTrack Protocol*. http://developer.berlios.de/projects/gift-fasttrack.

[Gnu] Gnucleus. http://www.gnucleus.com/.

[gnu02] Gnutella 0.6 protocol draft, June 2002. http://rfc-gnutella.sourceforge.net/developer/testing/ (Dec 20, 2003).

[Gro] Grokster. http://www.grokster.com/.

[GVi] Graphic vision glossary. http://www.graphicvis.com/Glossary/glossary.html (Jan 3, 2004).

[GWe] Gnutella web caching system (GWebCache). http://www.gnucleus.com/gwebcache/.

[HHH+02] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex queries in DHT-based peer-to-peer networks. *Lecture Notes in Computer Science*, 2429:242–??, 2002.

[iMe] iMesh. `http://www.imesh.com/`.

[KaZa] Kazaa Media Desktop. `http://www.kazaa.com/`.

[kazb] kazaa.times.lv. FastTrack history. `http://kazaa.times.lv/telo_p2p_fasttrack.htm` (Feb 2, 2004).

[Kli] Kazaa lite K++. Many websites have been forced by Sharman Networks, Ltd. to remove any links relating to Kazaa lite K++, thus there will be no URI given here. See `http://www.chillingeffects.org/dmca512/notice.cgi?NoticeID=861` for details.

[Kli03] Alexey Klimkin. Unofficial eDonkey protocol specification v0.6.1. `http://sourceforge.net/projects/pdonkey/` (Feb 6, 2004), April 2003.

[Lim] LimeWire. `http://www.limewire.com/`.

[LoC] Today in history. `http://memory.loc.gov/ammem/today/nov21.html` (Feb 7, 2004).

[Mar01] Sandra Marcus. The history of Napster, December 2001. `http://web.utk.edu/~smarcus/History.html`.

[Men] Thomas Mennecke. BitTorrent statistics. `http://www.slyck.com/news.php?story=370` (Jan 22, 2004).

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *1st International Workshop on Peer-To-Peer-Systems (IPTPS-02)*, pages 53–65, March 2002.

[Mor] Morpheus. `http://www.morpheus.com/`.

[Ove] Overnet. `http://www.overnet.com/`.

[Rit01] Jordan Ritter. Why Gnutella can't scale. No, really. `http://www.tch.org/gnutella.html`, 2001.

[Ruf] Lukas Ruf. LaTeX essentials. `http://www.topsy.net/TeX/`.

[SGD+02] Stefan Saroiu, P. Krishna Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of internet content delivery systems. In *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation (OSDI-02)*, Operating Systems Review, pages 315–328, New York, December 2002. ACM Press.

[Shaa] TheShadow's experimental BitTorrent client. `http://bt.degreez.net/`.

[Shab] Shareaza P2P. `http://www.shareaza.com/`.

[Slya] Slyck. `http://www.slyck.com/`.

[slyb] Grokster. `http://www.slyck.com/ft.php?page=5` (Feb 9, 2004).

[sup] Suprnova.org. `http://www.suprnova.org/`.

[WD03a] Arno Wagner and Thomas Dübendorfer. Observation of the sobig.f worm. `http://www.tik.ee.ethz.ch/~ddosvax/sobig/`, September 2003.

[WD03b] Arno Wagner and Thomas Dübendorfer. Observation of the w32.blaster worm. `http://www.tik.ee.ethz.ch/~ddosvax/blaster/`, August 2003.