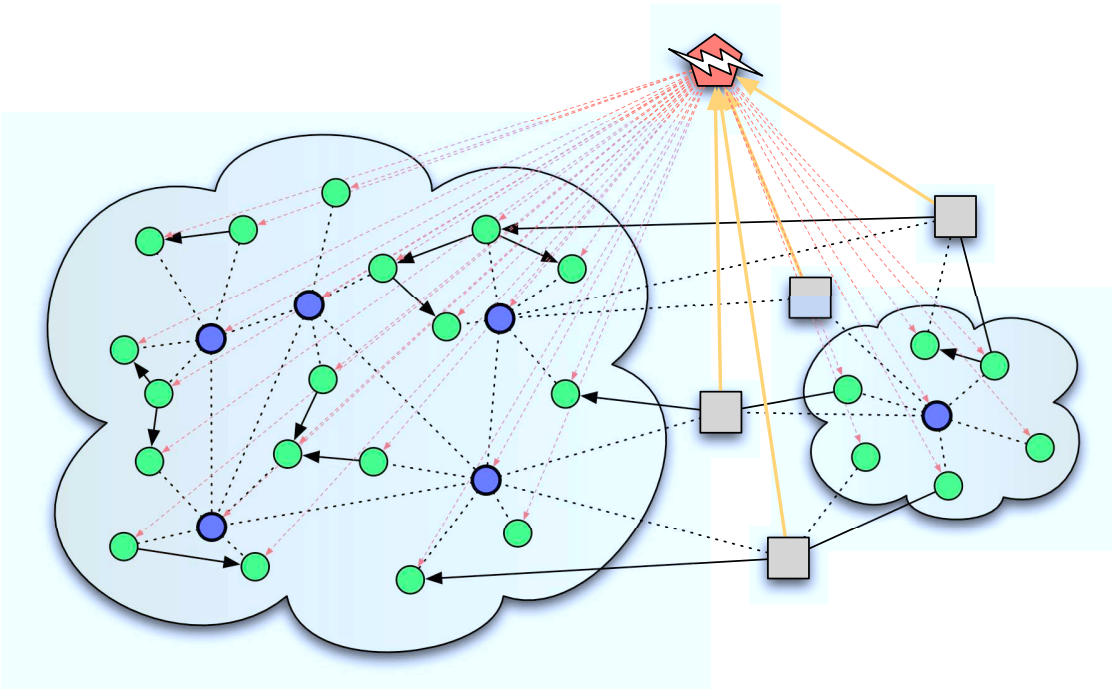
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

P2P Population Tracking and Traffic Characterization of Current P2P File-sharing Systems

Lukas Hämmerle



Master Thesis MA-2004-04
April 2004 - September 2004
Tutor: Arno Wagner
Co-tutor: Thomas Dübendorfer
Supervisor: Prof. Dr. Bernhard Plattner

Preface

While writing this thesis a lot of interesting P2P related developments took place. Some of them will change the P2P community very rapidly as it was always the case with P2P networks. It was a very exciting time and even when this thesis is finished I still will be observing the P2P (r)evolution.

At this place I also want to thank some people who supported my work and who I could count on if I needed assistance or guidance. Namely these are:

- Arno Wagner: For the good guidance and technical support
- Thomas Dübendorfer: For being my co-tutor and gnuplot guru
- Prof. Bernhard Plattner: For being my supervisor and making this thesis possible
- Dienstgruppe TIK: For providing an overall reliable computer environment that was comfortable to work with
- Stephane Racine: For introducing me to NetFlow and the cluster environment
- Philipp Jardas: For his preceding survey work about P2P file sharing systems
- Caspar Schlegel: For his support with UPFrame
- Rakesh Kumar: For providing me some of their FastTrack client port usage data
- Luca Deri: For providing us a free version of nprobe/ntop
- My fellow co-workers of G69: For the "candy-sharing" and the interesting conversations during coffee brakes

I further want to state that this thesis did not aim at finding methods to identify P2P users in order to prosecute them but in order to better analyze and observe their numbers and their usage characteristics. Since the described methods and concepts are based on flow-level data they can't be used to inspect which files a certain user shares or downloads.

Abstract

Detecting denial of service attacks and the spreading of virii and worms requires knowledge about the type of Internet traffic that passes a network. Nowadays a fairly large amount of all traffic in Internet backbones is not easily identifiable anymore since the amount of non well-known traffic has dramatically increased with the upcoming popularity of P2P file-sharing networks. For network monitoring and anomaly detection reasons it is important to know which traffic is to account for P2P networks.

This master's thesis gives an overview about problems and methods concerning identification, population tracking and traffic monitoring of P2P clients. The examined P2P networks are FastTrack, eDonkey, Overnet, Kademlia, Gnutella and BitTorrent, which represent the six most popular P2P networks worldwide. While other research on that topic mainly tried to identify P2P traffic directly, our approach first identifies P2P hosts in the medium sized back-bone SWITCH network and then uses them to track other peers. Various measurements made with a proof of concept implementation called "PeerTracker" are presented and verified. The results show that the P2P bandwidth consumption in the SWITCH network is considerable and that a substantial amount of the total P2P population can be tracked for some P2P networks.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In order to track the P2P population and to identify the P2P traffic we first need to summarize how P2P networks work and what data is used for this Task. Chapter 1 provides some basic information that is needed to understand the goals and the problems of this task.

Chapter 2 then will describe and analyze several methods for P2P identification which are used in several proof-of-concept scripts and a plug-in for UPFrame [2]. The plug-in and its usage is presented in Chapter 3. Different measurements and statistics about P2P usage in the SWITCH network can be found in Chapter 4. This chapter also deals with result verification. In Chapter 5 the results of this thesis are summarized before Chapter 6 presents some suggestions for further work.

Definitions

In this thesis terms are used as following:

host A computer that has access to the Internet with a global IP or behind a NAT box.

internal We divide all hosts worldwide in internal hosts, that are within the SWITCH network and other hosts which are extern.

P2P Technically means *peer-to-peer*. Communication relation between P2P[1] instances where all instances can act as a content requester and as a content provider. There are no dedicated fixed client or server instances. Although there are different types of P2P applications the term P2P used in this thesis generally denotes P2P file-sharing applications.

P2P protocol A specification that defines the communication between P2P instances of the same type.

P2P client Refers to a special piece of software that implements one or more P2P protocols. P2P clients are needed to participate in a P2P network.

[1] A peer-to-peer or P2P computer network is any network that does not have fixed clients and servers, but a number of peer nodes that function as both clients and servers to the other nodes on the network.

P2P host A computer connected to the Internet, running one or more P2P clients.

P2P user A human user who intends downloading and sharing of files from a P2P network by running a P2P client.

P2P network A group of hosts that communicate with each other using the same P2P protocol, equivalent to P2P system

P2P neighbor host Two P2P hosts running the same client are neighbors if they have a download or signalling connection established to each other or if they communicate over UDP.

P2P population The population of a P2P network consists of all its active peers.

P2P population tracking Population tracking is the attempt to locate as many peers as possible of a P2P network, ideally all of them.

(Ordinary) node A participant of a P2P network that has only basic functionality without any extra duties. "Servent" is a synonym often used in other publications.

Super node A client in a P2P network that performs special functions compared to ordinary nodes. Ordinary nodes must have at least one connection to a super node in order to be part of the P2P network. Super nodes (term of FastTrack protocol) are used equivalently for "ultra peers" (term of Gnutella protocol).

Overlay network Several connected hosts using the same communication protocol are forming an overlay network that uses an underlying physical network infrastructure.

FastTrack is the name of the P2P network and the protocol developed by Sharman Networks [3]. Other publications use Kazaa and FastTrack as synonyms, but actually the "Kazaa Media Desktop" (or just Kazaa) is the P2P client for the FastTrack network.

Leecher A P2P user that has configured its client to only download but hardly upload. This can be done by sharing no files or by setting a small upload transfer rate.

1.1 Overview

P2P file-sharing has become one of the most important applications in the Internet today. In 2003 estimated 81.5 Million Internet users were downloading music using a P2P file-sharing program according to [4]. This corresponds almost to 5% of the total Internet users worldwide. By the measures of bandwidth P2P file-sharing contributes a large amount of traffic that in many networks exceeds the mail/web traffic which has been dominating the Internet so far.

The following Subsections will depict the background and the goals of this thesis that mainly is about P2P but in the context of distributed denial-of-service attacks (DDoS) and Internet worm outbreaks.

1.1.1 DDoSVax Project

This thesis is part of the DDoSVax [5] which is a joint research project between ETH and SWITCH. The goal of DDoSVax is to find methods of detection and defense against distributed DDoS attacks. A DoS attack [6], is an attack on a computer system or network that causes a loss of service to users, typically the loss of network connectivity and services. Such attacks are not designed to gain access to the systems. In a distributed DoS attack several hosts are attacking a common target. DDoS attacks are committed by a person that has control over some dozens up to some ten thousands hosts. These hosts previously have been infected by a Trojan horse or an Internet worm that either installed itself automatically or with the help of its unsuspecting user that was confused by a social engineering trick which made him execute a binary containing the worm.

1.1.2 Reasons for P2P Identification

As is described in Section 1.2, one of the main goals of this thesis is to estimate the P2P traffic in the SWITCH network. To know the amount of bandwidth consumed by P2P hosts not only is important for network operators concerned about DDoS attacks as described in Subsection 1.1.3 but also for a range of other reasons which include:

- Application-specific traffic engineering
- Blocking/limiting specific traffic (e.g. P2P traffic) to reduce upstream costs
- Auxiliary mean to detect peers by actively polling them (see Subsection 4.1.2)
- Anomaly detection on P2P traffic

1.1.3 P2P Identification and DDoS

While not being obvious at first sight, there are several reasons why P2P usage is of concern for DDoS attacks.

1. In order to detect DDoS attacks and other network traffic anomalies, it is important to know which type of traffic a network sends and receives. By identifying and categorizing the traffic into different classes it is easier to monitor suspicious anomalies concerning DDoS attacks, e.g. a spreading worm.
   Usually TCP and UDP port numbers can be used for network traffic identification since most Internet applications have assigned[2] port numbers. Unfortunately the amount of unidentifiable traffic has been increasing dramatically over the past years (see Appendix E) which has it made difficult to monitor certain traffic classes. It is quite obvious that the upcoming of P2P clients born after the Napster era had a great influence on this rise and various related publications[3] show that in some Internet service provider (ISP) networks the amount of P2P traffic already in 2002 has

[2] See IANA Port Numbers http://www.iana.org/assignments/port-numbers (July 2004)
[3] See Section 4.5

climbed up to about 75% [7]. Depending on the used P2P protocol and P2P client, only a fraction of this traffic can be identified directly since quite a substantial amount of P2P users don't use their default port numbers anymore [8, 9].

2. Knowing the P2P bandwidth consumption can be helpful for DDoS detection since some recent Internet worms made use of P2P networks. Some of the worms not only did spread by email but copied themselves to the shared directories of P2P clients in order to propagate within P2P networks[4]. Using filenames of popular files they try to make P2P users download and execute them in order to infect their computer.

3. As described in [10] P2P systems provide an attractive infrastructure for DDoS attacks. They share resources and generate a large amount of traffic which could mask attack traffic without arising suspicion. A P2P worm could use security holes in P2P clients to infect them and spread among the P2P population.

4. Nowadays Internet worms usually infect hosts to make the attacker able to get control over it. To control a computer one must be able to communicate with the worm. One method to communicate with infected hosts is by using the IRC system as explained in [11]. But recent Internet Worms not only have used P2P technology for propagation, they also use it to communicate with each other in a decentral way that allows their master to command them almost anonymously[5]. For the person that controls the worm this has the advantage of a reduced risk of being caught since the decentralization of P2P systems allows to makes it hard to track down the person that inserted a command in that P2P system as described in [12]. Getting some general ideas of P2P identification can help to locate and shut down such P2P worm networks.

5. Another danger coming from P2P clients is concerning their long uptime. They usually are used to download large files that take quite a while to download. Therefore the clients run most of the time unattended by their users. Some users let their clients run for several days or even weeks as is shown in Table 4-6. If there exists a serious security flaw that could be used to infect a host running a certain P2P client, this could have serious consequences. Since many P2P clients are not open source there even exists the possibility that their creators themselves implement some hidden back doors to gain control over a host which is far more realistic than assumed[6].

1.2 Goals

Result of this thesis should be a collection of algorithms and concepts to identify and track hosts of the current most popular P2P file-sharing systems inside

[4] See article "Fizzer stealth worm spreads via KaZaA" http://www.securityfocus.com/news/4660 (July 2004)

[5] See article "The rise of P2P worms -And how to protect yourself" http://reviews-zdnet.com.com/4520-6033_16-4207594.html (September 2004)

[6] See article "The Dangers of Uncontrolled Software Use" http://www.winnetmag.com/Article/ArticleID/40477/40477.html (September 2004)

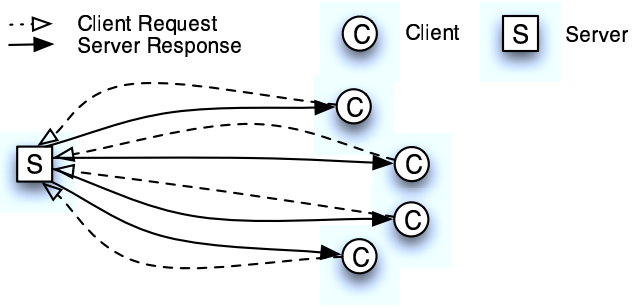


Figure 1-1: Client-Server model

and outside of the SWITCH network in order to estimate their number and bandwidth consumption. The identification of P2P hosts is done analyzing Cisco NetFlow data that was collected by the SWITCH border gateway routers.

The algorithms should be:

- Flow-based, without packet analysis
- Passive, without use of self operated clients/servers or interfering/impacting peers
- Capable of offline and online identification
- Accurate with low false positive/false negative rate
- Scalable for fast computation with data of large networks
- Robust in spite of flow/packet loss

For the full task description see Appendix A.

1.3 Internet Communication Patterns

In the beginning of the Internet there were only a few protocols that consumed almost the whole bandwidth in the backbones. Nowadays a huge number of protocols exist that use quite a substantial amount of bandwidth [13]. New applications like P2P networks or online games often use different network models as the old protocols like HTTP, FTP, SMTP or Telnet do. The following two Subsections explain the differences between them.

1.3.1 Client-Server Model

The most common type in communication between two Internet hosts is the client-server model shown in Figure 1-1. Several clients ask as content-requesters a dedicated Internet server, which acts as content provider, to send the requested data. Looking at CPU usage and bandwidth consumption this centralized model is very unbalanced since the server has the main load of work.

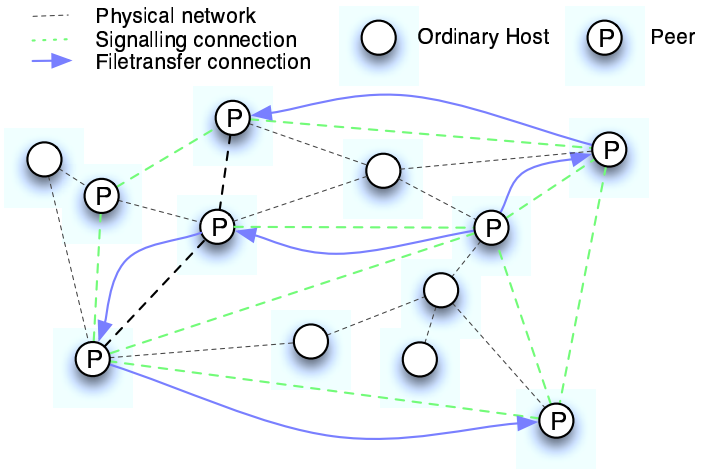


Figure 1-2: P2P model

Clients usually send only a small request without a great effort, whereas the server has to compute and then transmit the requested data. The answer moreover can be a multitude larger in size than the request.

So, depending of the number of requests, the server has to be a moderately fast computer with a fast Internet connection to respond to all requests. Since most content provider have to pay for their upstream the client-server model is rather expensive for the server owner. A more detailed overview about the client-server model can be found in [14].

1.3.2 P2P Model

While P2P applications include distributed computation like SETI [15], voice over IP like Skype [16] (which btw. is developed by some of the creators of FastTrack) and other messaging services, the most bandwidth consuming application today is file-sharing. The P2P communication model depicted in Figure 1-2 is true for general file-sharing applications but slightly differs for some P2P systems as is described in Chapter 2.

In the P2P model every participating host can act as content consumer as well as content provider. This model is more balanced out since CPU-time and memory is distributed and shared among all the P2P hosts in a network. Although there are P2P networks like FastTrack [17] or eDonkey [18] which use a two-tier system with ordinary nodes and super nodes for their network architecture, the P2P approach of most file-sharing applications is decentralized and therefore almost immune against network problems. This development is a consequence of the fall of Napster which resulted in highly decentralized clients like Gnutella which are completely independent from any central server, but have a high bandwidth overhead as shown in [19] and sometimes also a bootstrapping problem [20].

P2P traffic consists of signalling traffic and download traffic. The first is used for search queries and for general network maintenance while the latter

consists of transfer data between a content-requesting and a content-providing peer. Signalling traffic occurs either directly between peers which then form an overlay network or between ordinary node and a super node. Usually TCP or UDP is used for signalling traffic while TCP is used for download traffic.

1.4 Ways of File-sharing

While there have been numerous ways of exchanging files since the beginning of the Internet, P2P file-sharing is by far the most popular nowadays. Other ways like FTP, IRC, BBS or the (again) upcoming Newsgroups[7] are mostly used by advanced computer users that first of all know about the existence of these file-sharing methods and secondly possess the knowledge of handling the needed tools which usually are not as user friendly as todays P2P clients. It therefore can be assumed that P2P file-sharing programs are the most common way of sharing files.

1.5 P2P Networks and Clients

No doubt P2P is the killer application for ISPs. Measurements [21] at a large ISP network have shown that P2P traffic accounts for more than 75% of all traffic and that an average P2P client produces about 90 times more traffic than an average web client , exceeding WWW traffic by nearly a factor of three [13]. Therefore file-sharing is not unproblematic for ISPs which often have to pay their upstream which is massively increased by P2P traffic. Moreover they have to provide costly networks that are capable of transporting huge amounts of data. On the other side the customers are willing to pay for broadband connections since they are perfect for file-sharing. So it is no surprise that popular file-sharing websites like Suprnova.org [22] are loaded with ads from local ISPs.

1.5.1 Generations

It is a difficult task to categorize P2P systems. Although most of them have a common purpose - to share files and other resources - they use different techniques and are constantly evolving, adding new features that are characteristic for more advanced P2P generations.

1. Generation: Controlled with centralized search indexes

After its creation in May 1999 by founder Shawn Fanning, Napster became shortly very popular around the world. It didn't take long for the music industry, represented by the Recording Industry of America (RIAA), to take actions in December 1999 against a service that endangered their profitable business. After a spectacular legal battle, Napster finally was forced to go offline in July 2001. By shutting down Napster's indexing servers, the whole network can be "turned off" which made Napster very vulnerable to legal actions. After their successful legal actions against Napster the RIAA started another lawsuit against Audio

[7] See article "The Newsgroups Evolve" http://www.slyck.com/news.php?story=539 (August 2004)

Galaxy [23] that became popular after Napster's fall even though Audio Galaxy existed before Napster. While AudioGalaxy was not shut down completely, it settled out of court. They still exist, after paying the RIAA a lot of money together with the promise to block copyrighted songs though[8]. Their user base decreased very fast since then.

First generation P2P networks are using central servers for file indexes and searches. These servers are operated by the network maintainers. Therefore first generation P2P networks can be shut down completely.

2. Generation: Completely decentralized search indexes

The fact that Napster could be stopped a had a great impact on the succeeding P2P networks which became quite numerous [24]. The popular Gnutella network was designed with complete decentralization in mind. Together with other networks that have been developed since then, it can be accounted to the second generation P2P networks. The approach of total decentralization was the extreme opposite of Napster's principle and caused some scalability problems, mainly because of the search algorithms [19]. More sophisticated P2P systems like FastTrack which is actually based on Gnutella, could solve some of these problems by bringing back some central elements in form of super nodes. But in contrast to Napster's server these super nodes are not operated by the creators of the P2P networks but they are dynamically self assigned to regular clients.

Besides techniques like swarm delivery (parallelly downloading parts of the same file from multiple sources) or anti-leeching algorithms nothing revolutionary new was developed for P2P systems.

3. Generation: Encryption and anonymity

Even before the RIAA started to sue the P2P users, the FastTrack creators implemented encryption into their P2P protocol. Although the reason for this primarily was to make it hard to reverse engineer the protocol[9], it provided some protection for the users. But obviously not enough. In mid of 2003 the RIAA started a "Fear, Uncertainty, & Doubt" campaign and sued several hundred FastTrack users (see Subsection 1.5.4) which was one of the reasons why the FastTrack user population has dropped from 4.5 million to about 2.5 million users in mid of 2004. Since then the call for more anonymity and encryption became loud. But up-to-date only Freenet [12] and Earth Station 5 [25] efficiently provide some sort of anonymity. While Freenet is not aimed at file-sharing, Earth Station 5 (ES5) is an obscure[10] P2P client that provides complete anonymity together with a feature that can "penetrate" almost any firewall.

1.5.2 General P2P Usage

Worldwide there exist some dozens of P2P systems [24]. In the western world about half a dozen systems are widely used. This includes the large networks

[8] See article "R.I.P. AudioGalaxy" http://www.kuro5hin.org/story/2002/6/21/171321/675 (August 2004)

[9] SharmanNetworks, the inventor of FastTrack, earns their money by selling licenses ad ads

[10] The client has a bad reputation and its developers claim to be located in Jenin Refugee camp to escape the wrath of the entertainment industry... Moreover some very suspicious and dangerous code was found in ES5.

FastTrack, eDonkey, BitTorrent, Overnet, Kademlia, Gnutella, Warez, DirectConnect. Even among these systems there is a geographic gap in usage. While eDonkey mainly is used in Europe, DirectConnect e.g. is a P2P system with a large user base in the U.S.A. In Asia there exists also a large variety P2P clients, as Soribada [26], V-Share or Argus. These clients are localized and therefore not popular among western Internet users because of usability matters and content.

Not all P2P networks provide the same type of shared files. While Napster was used only to share MP3 files most networks nowadays can be used to download not only non-MP3 music files (like Ogg, ACC, WMA) but also a the whole range of multimedia files which includes movies (even whole DVDs), Software (mostly games), pornography and eBooks. But still there are certain P2P networks that are known or best suited to share mainly a certain type of file. Also the quality of the shared files is highly depending on the P2P network. FastTrack for example is actively polluted [27] by companies like Overpeer [28] with fake files that aim at frustrating the P2P users and make them abandon the platform.

P2P networks are used by different kind of people for different purposes. FastTrack for example is used rather by average computer users to download the latest MP3s and so is Gnutella. BitTorrent and eDonkey have their user base more in the technically more sophisticated corner that primarily downloads large files of several hundred Megabytes up to some Gigabytes which are the sizes of movies and software packages.

Concerning the activity time of P2P users there are rawly two types; one sort of users does P2P file-sharing occasionally and influenced by day-time. Results in Chapter 4 show that P2P usage in SWITCH network is highest during work time. This group of P2P users usually does file-sharing only for some hours a day, mostly to download smaller files since they don't take very long to download.

The other group of users consists of power users who let their P2P clients run for several days. They tend to connect to P2P networks where there are Linux clients available. Often these users are responsible for most of the consumed bandwidth. In [7, 21] they state that only a few of these "heavy hitters" are responsible for most of the total traffic in these networks.

1.5.3 Popularity

P2P systems become popular and disappear within years or even months. It's only about three years since the Napster saga has ended. In the meanwhile various P2P systems were more or less popular. As was also stated in Philipp Jardas work [29] it is generally difficult to come up with accurate numbers about the amount of users on a specific network, due to the decentralized architecture. In the case of BitTorrent it is not even easy to *estimate* the user population since BitTorrent peers are part of several thousand unconnected networks (see Subsection 2.1). Of the many different P2P systems the six most popular in the western world on July 2004 were BitTorrent, Gnutella, FastTrack (including iMesh), eDonkey (including Kademlia) and Overnet. But already in September 2004 a new P2P system called "Warez" was among the top 5. While writing this thesis FastTrack was overtaken as most popular P2P system in terms of users by the trio eDonkey/Overnet/Kademlia which all are strongly related to each other and sometimes are counted as one network. The beginning decrease of

users[11] of the longterm leader FastTrack has several reasons:

- Mainly FastTrack users were sued by the RIAA (see next Subsection 1.5.4). The publicity in media scares the users and makes them looking for (safer) alternatives.
- The free P2P clients for the FastTrack clients are bundled with spy- and adware. As an irony of fate the Kazaa Media Desktop [17], FastTrack's most prominent P2P client, was outrun[12]as number 1 download at download.com by Ad-Aware which is a software that removes spy- and adware. On a test machine installing the FastTrack clients Kazaa Media Desktop and iMesh resulted in 319 identified ad- and spy-ware objects (registry keys, files and processes. Checked with Ad-Aware 6). Removing these objects resulted in defunct clients.
- There exist several companies like Overpeer [28] that pollute (primarily) the FastTrack network with fake files [27] to annoy its users from downloading copyrighted material. They unofficially get paid by copyright holders.
- Other P2P clients get easier to use, are free, deliver better performance and are therefore interesting for the average computer user.

The users leaving the FastTrack community often end in the eDonkey realm, but also in the Gnutella network which is upraising again these days. But SharmanNetworks, the owner of the FastTrack protocol, is not giving up its dwindling cash-cow so easily. Nowadays they try to stop the decrease of users with localized Kazaa clients and new content rating mechanisms that replace the hardly successful existing solutions [27].

The considered P2P systems in this thesis are discussed in Section 2.1.

1.5.4 P2P Prosecution

After unsuccessful protecting media with encryption (DVD) or hardware depending copy protections (CD), the music and movie industry began to realize that they have to change their strategy. But instead of directly challenging P2P file-sharing with attractive offers like available in the iTunes Music Store [30] that could make the users legally buy their music, the music industry - or their representative association, the RIAA and MPAA [31] - first decided to choose another way. They sued the network maintainer of P2P networks. First and may be the most popular victim up to date was Napster. Due to its centralized design it was relatively easy to shut down the network. The dramatic fall of Napster led to P2P systems like the Gnutella network which is completely decentral and can hardly be shut down. Other P2P networks had learned their lesson too, so that it is very difficult to stop recent P2P networks. Therefore the RIAA had to chose another tactic when they had ruled out all the centralized P2P systems like Audio Galaxy, that was popular in the after-Napster era. Moreover in August 2004 the U.S. Ninth Circuits Court affirmed four very

[11] See Slyck.com article FastTrack Continues its Slide http://www.slyck.com/news.php?story=492 (August 2004)

[12] See Slyck.com article Kazaa Media Desktop Removed From Download.com http://www.slyck.com/news.php?story=478 (August 2004)

important points concerning P2P technology, stating that P2P software per se is not illegal [32]. In June 2003 - even before the affirmation of this ruling - the RIAA more or less successfully started an anti-P2P crusade not against the creators of the networks or the software, but they sued the P2P users directly[13]. This tactics led to a temporary decline of the P2P users but had almost no effect on the overall P2P user base except that people moved to networks which are not targeted (yet) by the RIAA. But according to ITIC [33] the P2P population in August 2004 has increased since July 2004 by 15% again in spite of new RIAA law suits against users.

When suing users directly several problems arise. The main difficulty is to prove that somebody shared copy-righted content and then to identify these person [34]. Since P2P file sharing is an Internet "killer application" that makes users willing to pay for broadband connections, the ISPs generally are not interested to give their customers name. Nevertheless the ISPs can be forced by courts to present the identities of accused users. But even if the name of the person, who pays the ISP's bill, is known this person often was not the one who actually was responsible for illegally sharing copy righted content. So often the wrong persons are sued as has been the case several times.

Other targets of RIAA and similar organizations are so called link sites which are used by many people to find and rate certain downloads in P2P networks. The principle of link sites works as follows: Files in a P2P system like eDonkey can be found in the network using a hash value that uniquely identifies a certain file. Publishing such links on a website has the advantage that false ("fakes") or corrupt files can be marked as such. The users basically give them a bad rating to warn other users. In spring 2004 the most prominent eDonkey/Overnet link site called "ShareReactor" was closed by Swiss authorities after almost three years[14]. But soon afterwards new sites arose and took ShareReactor's place.

1.6 Network environment

P2P identification in this thesis is done processing flow-level data gathered from the SWITCH network which is described in the next Chapter as is the flow format.

1.6.1 Flow-level Data

A flow is kind of an accumulated unidirectional stream of captured network packets, that is summarized by a set of fields that describe the most valuable information about that stream. These information fields include source and destination IP and port, the number of packets and bytes transfered and the duration of the flow. Although a flow does not contain any payload data of the transmitted packets, it allows network operators to use NetFlow for billing, planning, monitoring as well as for research activities.

Developed by Cisco [35] for their high performance Internet router and patented in 1996 NetFlow nowadays is de facto the standard accounting tech-

[13] See article "RIAA Lawsuits Unpopular in United States" http://www.slyck.com/news.php?story=549 (August 2004)

[14] See Slyck.com article ShareReactor Down Indefinetely http://www.slyck.com/news.php?story=424 (August 2004)

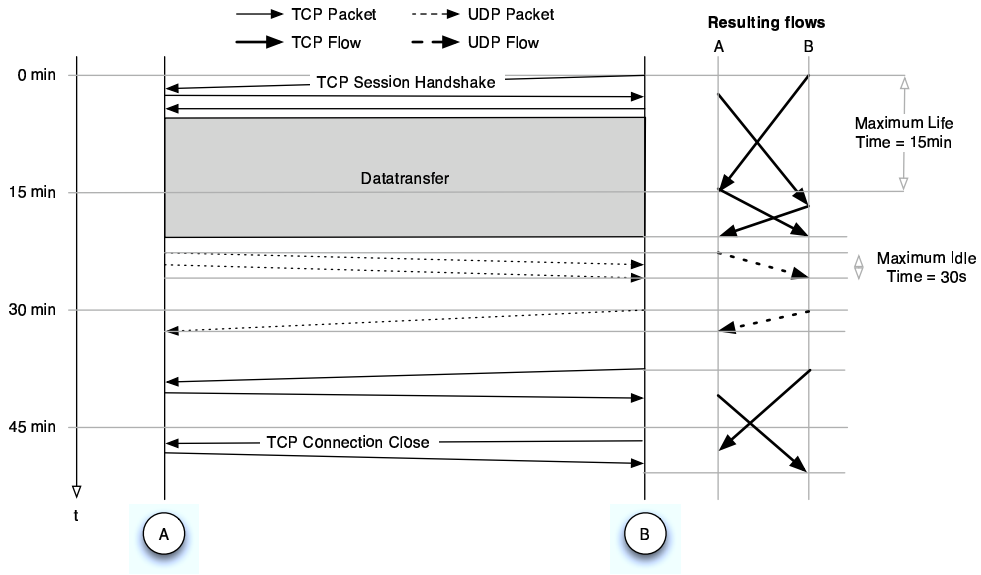


Figure 1-3: Flow Timings

nology for network service provider. Although the latest version 9 of NetFlow is on its way to become an IETF [36] standard, version 5 still is the most common used.

Since flows are unidirectional a TCP connection will result in two flows. Packets get accumulated in a flow if they have matching source/destination IPs and ports. Flows are expired from the router cache and emitted to a collector if one of the following conditions is true (also see Fig. 1-3):

- Maximum lifetime of the flow is exceeded. Default is 15 minutes.
- Maximum idle time is exceeded. During that time no other packets have been sent. Default is 30 seconds.

Since UDP is a connection-less protocol, each UDP packet in fact is independent from other packets. UDP flows therefore get emitted sooner and more often than TCP flows. Nevertheless UDP streams can get accumulated in a flow if the packet frequency is high enough.

Up to 30 flows are contained in a NetFlow packet that is sent in a UDP packet to a collector host that can store or process the NetFlow data.

1.6.2 SWITCH NetFlows

The data basis of P2P identification in this thesis is a large archive of NetFlow data that was collected from the four border gateway routers of the SWITCH network[15] depicted in Figure 1-4.

The flows are saved in NetFlow Version 5 format[16]. Emitted flows are collected by a collector host which creates two NetFlow files every hour. One

[15] See Appendix D for more information about the SWITCH network
[16] See Appendix C for more details concerning the NetFlow format

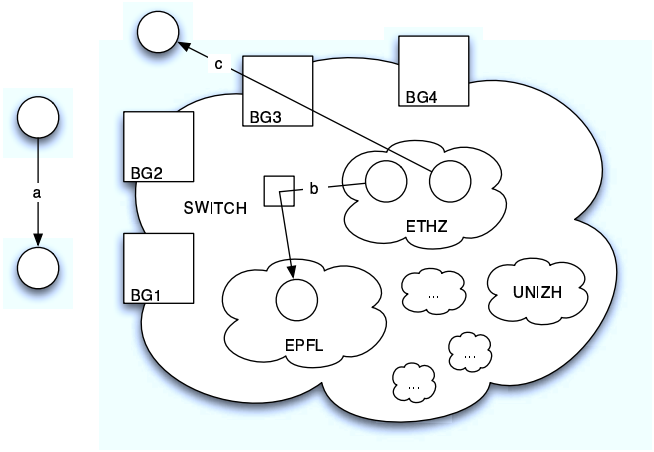


Figure 1-4: Switch network topology

containing the data of three of the four border gateway routers and the other containing only the data of the largest one which produces about as many flows as the three others together.

As shown in Figure 1-4 flows are only captured if they cross one of the border gateway routers (c). Neither packets between two internal hosts (b) nor between two external hosts (a) pass a border gateway router and thus don't result in a flow.

In a 30 day measurement an average of 45 million flows per hour resulted. The graph depicted in Figure 1-5 shows that the number of flows emitted is day time dependent. Weekends (June 27./28.) don't seem to have that a great impact on the number of flows as e.g. on the number of active web clients (see Figure 4-3). The negative peak before noon on June 22. is due to a NetFlow collector problem on that day.

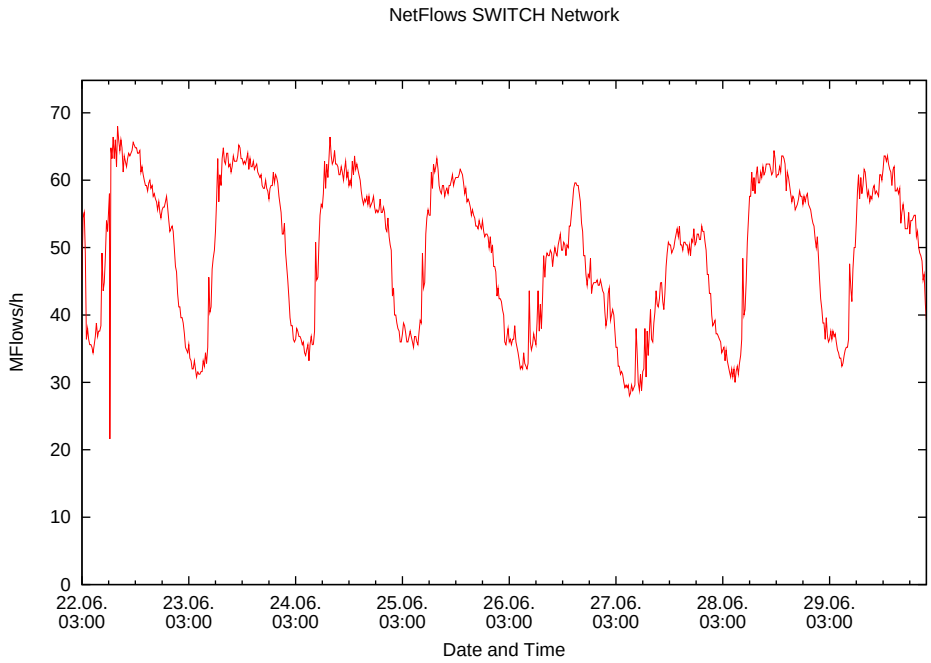


Figure 1-5: NetFlows emitted in SWITCH network

Chapter 2

P2P Host Identification

In the following chapter approaches and methods for P2P identification are presented and discussed. Directly identifying P2P traffic nowadays seems to be inaccurate as has been shown in other publications [8, 9]. Therefore we focus on P2P host identification. We try as a first step to identify P2P hosts in the SWTICH and to later track parts of the external peer population. This approach has also the advantage that traffic of P2P clients which use the same download ports (as eDonkey, Overnet and Kademlia) can be better separated.

Some of the discussed methods then are used to implement or verify an identification algorithm which is subject of Chapter 3.

Identification History

Traffic identification was rather straightforward in the beginning of the P2P age when most clients were using default ports. Newer generation P2P clients are incorporating various strategies to avoid detection (see Section 2.3.1). One of them is to choose random listening ports making it difficult to accurately identify P2P traffic and hosts with a stateless method.

While there are numerous measurements studies[1] that use packet inspection [9, 37, 13, 8] for traffic identification, recent ones have been published that use flow-level heuristics [7, 38, 39]. One of the reasons for this trend is that - depending on the amount of network traffic - it is hardly possible anymore to do packet inspection or to capture the content of Internet packets at fast rates in the GBit range [40]. Moreover flow-level identification is more robust against protocol changes of P2P clients. In contrast, flow-level identification is not always accurate enough and may get even more inaccurate in future with more sophisticated "camouflage" techniques used by newer P2P clients. Another disadvantage of flow-level identification is that hosts must be active enough to precisely identify a peer since one flow is hardly enough to state that a host is a peer as is shown in Subsection 2.3.2.

So far there don't seem to exist publications which first try to identify and classify peers using flow-based methods.

[1] In Section 4.5 some approaches of other researches are presented and summarized.

2.1 P2P Systems Considered

In this thesis we examine the following P2P systems: BitTorrent, FastTrack, Gnutella, eDonkey, Overnet and Kademlia.

According to Slyck.com [41], which is an extensive information site about P2P file-sharing, the five most used P2P systems in July 2004 were FastTrack, eDonkey, Overnet, iMesh, and Gnutella.

iMesh has been using an own protocol in its early stages, but nowadays connects to the FastTrack network. Two networks that are not displayed in their users statistics are BitTorrent and Kademlia. BitTorrent does not show up in Slyck statistics since its user base is difficult to estimate as is explained later. The Kademlia network is a creation of the developers of eMule that started as a very popular eDonkey client. Since the first quarter of 2003 Kademlia support is included optionally in eMule.

In the beginning of this thesis it originally was planed to identify only eDonkey, Overnet and FastTrack clients. The three additional networks were considered then for the following reasons:

- The Kademlia protocol is closely related to the eDonkey network and has a fast growing user base. It may get more important in near the future. Since every eMule client is part of the eDonkey network and since eMule is the only client for Kademlia, every Kademlia users is also an eDonkey user.

- BitTorrent is responsible for a huge amount of traffic and although it is hard to estimate its user base, BitTorrent traffic has increased dramatically during the past month.

- The user base of the Gnutella network is increasing again. Moreover the identification algorithms were quite promising to accurately identify Gnutella clients.

This gives us a total of six P2P networks considered. It is assumed that these networks are the most popular P2P networks in the SWITCH network.

In the following a brief description of the considered P2P systems and some of its popular clients is given. Figure 2-1 gives an overview over the relation between the networks and some of their most popular clients.

eDonkey

Is a two-tier system that uses a centralized server architecture with only about 80 worldwide available public servers which hold the file index of shared files. The dedicated server software is freely available and can be used by anyone. Each server communicates with other servers so that the overall user number is known at any time. Each eDonkey client generally has one TCP connection to a server. In newer clients they use UDP to query other servers if they are looking for files. Some clients, like the multi-hybrid client mlDonkey, opens connections to several eDonkey servers.

On August 9. 2004 the the maximum number of users connected to a server (“Razorback 2”) was 519’850 users. On an average day in July 2004 there were between 2 and 2.6 million people connected to the eDonkey network, sharing

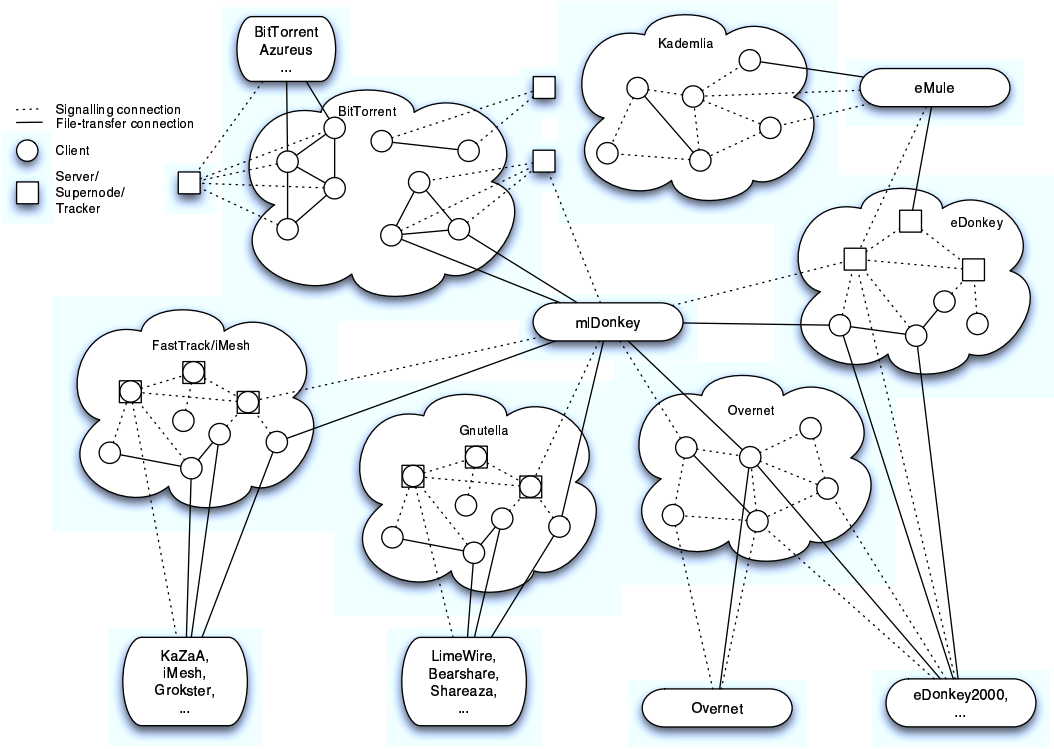


Figure 2-1: P2P network and client relations

between 190 and 330 Million files. These numbers were gathered by polling all the available eDonkey servers.

Clients for the eDonkey network include the official eDonkey2000 client by eDonkey's creator MetaMachine, the even more popular eMule and the derived open source {l,a,x}Mule clients. eMule has extended the eDonkey protocols with various features as download queuing and additional source finding mechanisms. Some of these added functions only work among eMule clients. Worth annotating is also mlDonkey [42] which is a is able to connect to various P2P networks as shown in Figure 2-1. Support for these networks is varying but especially the connection to the FastTrack network is well appreciated among the Linux users.

Own measurements made in June with mlDonkey, which can distinguish eDonkey clients, running for 4 days have shown that it had contact with eMule clients (55%), the official eDonkey client (40%) and some less popular clients (5%) during that time. Another measurement with an eDonkey server showed that 67% of all connected users run eMule.

Overnet

When the limitations (centralized and few servers) of eDonkey became obvious, *MetaMachine* designed a successor that was completely decentralized. Overnet was born. It was supposed to replace eDonkey but hasn't succeeded so far. Mainly to the great success of the eDonkey client eMule which has an easy-to use user interface that appealed to many eDonkey users. Overnet uses - like Kademlia [43]- the XOR-algorithm that distributes the file indexes to the participants of the P2P network. UDP is used for signalling traffic and it produces a lot of overhead.

At the moment there exists only the official Overnet client and the multi-hybrid client mlDonkey. It is assumed though that only very few people use the mlDonkey client since it is due to its complicated handling is used merely by rather technically sophisticated people. Moreover the official client is free and also has a GUI version that is easier to handle.

Kademlia

Establishing and populating a new P2P network is not an easy task when there exist dozens of competing P2P networks. The creators of the very popular open source eDonkey client eMule therefore in February 2004 bundled support for the new Kademlia network with their client. Kademlia is like Overnet completely decentralized and also uses the XOR-algorithm with UDP for signalling traffic, but nevertheless is incompatible to the Overnet protocol. Kademlia support is not yet enabled by default in the eMule client since it still is in a test phase. Therefore presumably only a small fraction of eMule users have turned it on. There aren't any official numbers available how many people that use the Kademlia network yet. Running an eDonkey server shows the number of eMule clients that have Kademlia enabled. According to the numbers of own measurements there are rawly 13% of all eMule users which are also part of the Kademlia network.

eMule so far is the only client connecting to the Kademlia network. Therefore every Kademlia peer is also part of the eDonkey network. Unfortunately eMule

seems to use the same port for eDonkey and Kademlia traffic why these two can't be separated.

Gnutella

One of the first completely decentralized P2P protocol, that was developed by NullSoft employees. After releasing a first implementation NullSoft's mother company (AOL) prohibited the publication of this client within hours. But it was too late. The open source community reverse-engineered and improved the protocol. The user base grew. It early was discovered that the protocol had some severe problems [19] (e.g. 1 search query could produce a total of 800 MBytes of traffic in the whole Internet). After FastTrack got popular, Gnutella's user base stagnated around 400'000 users but is now increasing again since more and more users abandon the FastTrack network.

Gnutella holds 4-10 TCP-connections to other clients. In the beginning no UDP was used which often was criticized. Nowadays UDP is also used together with a two-tier system where so called "ultra-peers" provide some extended functionality compared to an ordinary node. Ultra-peer status is self-assigned to powerful peers.

There exists a variety of commercial and open-source clients for Gnutella. The most prominent are the Java client LimeWire [44] with an approximate 50% user base, Morpheus with 20%, and BearShare and Shareaza with about 7% each. Beside them there exists about another dozen clients due to the freely available protocol specification of Gnutella.

FastTrack

Originally based on Gnutella, FastTrack with approximately 2.7 million users (September 1.) still is the most popular P2P network. But as stated more and more users abandon the FastTrack network.

FastTrack was one of the first P2P networks which used a two-tier approach. Every FastTrack client needs a TCP connection to one super node to join the network. Super nodes are dynamically self-assigned to powerful nodes. According to measurements made in [27] there were between 20'000 to 30'000 super nodes worldwide in mid of 2004. Clients seem to communicate also over UDP to query other super nodes. All FastTrack signalling traffic is encrypted. Download traffic isn't (yet).

There exist only three official sanctioned to connect to the network Kazaa, iMesh and Grokster. Since all three of them in the "free" version are packed with unwelcome third-party programs, some people have started to modify the official clients what resulted in Kazaa Lite K++ that comes without the advertising and includes some useful bonus helper programs. Since the maintainer of the FastTrack network, SharmanNetworks, earns its money by selling licenses and ads, it tried to fight against this unofficial clients using legal actions. Websites were forced to remove all links to Kazaa Lite and even Google was forced to remove search results containing Kazaa Lite[2]. Although these legal actions made it more difficult to find proper versions of Kazaa Lite and one won't find

[2]See Google pulls links to Kazaa imitator http://news.com.com/2100-1032-5070227.html (August 2004)

any version of Kazaa Lite using the FastTrack network, it still is not that hard to get it, using other P2P clients like eDonkey or Overnet.

BitTorrent

Developed by a single person (Bram Cohen) BitTorrent became more and more popular for file-sharing in 2003. Aimed at large downloads of several MBytes it uses the upload capacity of P2P users and the tit-for-tat principle to share bandwidth between users who download the same file. This approach is very successful [45] and delivers very good download rates if enough people are downloading the same file. The files downloaded are described in so called *torrent file*s.

As can bee seen in Figure 2-1 there is no single BitTorrent network, but thousands of temporary networks consisting of clients downloading the same file (e.g. on 3rd of August, over 50'000 people were downloading the newly released horror shooter game "Doom 3" from the two main torrents[3]). These split networks makes it difficult to count the total user base. In January 2004 Suprnova.org [22] which claims to host about 60% of all torrent files, reported over 1 million BitTorrent users [4] though. Traffic statistics let assume that the population since then has grown even more.

After a torrent file is downloaded from a web page or otherwise received, the BitTorrent client has to ask a *BitTorrent tracker*, to provide addresses of other peers downloading the same file. The communication between the peers is done using TCP only. The tracker is a central element in this architecture. Without it the active peers don't get new addresses from other peers what causes the network to die sooner or later. The network can also die if there are no seeders - clients that have downloaded 100% of the file - anymore. Most torrents last only for some days or weeks, but others exist for several months [46]. This is - among other reasons - because BitTorrent's philosophy that clients keep uploading even when they have finished a file. The user itself has to quit the program to stop the outgoing transfers.

Popular clients are the original BitTorrent client written with Python by Bram Cohen itself and the Java based client Azureus, which has a nice user interface and provides some extra features (like multiple simultaneous downloads, autonomous exit after a certain upload-download ratio has been reached or priority downloads of contained files) that advanced users appreciate.

There is an additional short overview about the most common P2P networks in Appendix B.

2.2 Traffic Observation Possibilities

2.2.1 Packet-level Observations

To observe P2P clients in order to reverse-engineer their protocol the following setup was used:

[3] See "Doom 3 Sales Going Mad... Downloaders March On" (September 2004) http://techbits.ca/modules.php?name=News&file=article&sid=843

[4] See Slyck.com articleBitTorrent Statistics http://www.slyck.com/news.php?story=370 (August 2004)

- Observer host: Transparent Linux router with two network interfaces. Connected to an unfiltered network.

- Client host: Debian Sarge Linux/Windows XP dual boot client host with one network interface

Goal of these observations was to analyze the common behavior of some popular P2P clients of the considered networks. This means e.g. how many UDP packets they send, how many connections usually are opened, on what port they listen and with how many hosts they stay in contact.

Network analyzers like *tcpdump* [47], *ethereal* [48] or *nprobe* [49] can be used on the observer host to capture, monitor and even replay network traffic data. A firewall like *Keerio Firewall* [50] on the client (windows) host allows to observe the current established connections and listening ports. This is also helpful to observe the transmitted traffic for each connection.

In order to run P2P clients under realistic conditions some files must be shared and it is desirable to make other clients communicate with the test client. Therefore one must initiate some file transfers either by doing search queries and downloading files or by making other P2P users download shared files. Legally sharing popular files, that are frequently downloaded, is not as easy as it sounds. Considering that there are mainly are three types of shared files (namely music, video and software) and that the most popular files usually are current chart hits, legally shared MP3s don't seem to attract many people as was observed. One way to make other P2P users download from the shared files is to trick them choosing file names of popular objects and thus "polluting" the P2P network. A slightly less misleading but quite effective way of sharing highly demanded files in a legal way is to share official movie trailers of currently playing movies or to share rogue clients[5] like Kazaa Lite or iMesh Lite although these programs obviously can't be shared on the FastTrack network, or at least not using their real names and file hashes.

2.2.2 Flow-level Observations

Packet-level analysis is technically very difficult for large backbone networks since traffic volume just is too big. Therefore the traffic can only be analyzed if the data is in some way compressed or abstracted. In the SWITCH network NetFlow data is collected and used for various purposes. One way to analyze the flows is to use UPFrame.

NetFlow Data

DDoSVax project has access to the NetFlow data of the four border gateway router of the SWITCH network. There is a longterm archive where complete, unsampled NetFlow data is saved since March 2003. A 600GB short term archive that holds data for the past four weeks is used to analyze current network events like worm or DDoS attacks.

[5] See the paragraph about FastTrack in the previous subsection

UPFrame

UPFrame [2] was developed for the DDoSVax project [5] to have the possibility to do online measurements of NetFlow data. Providing a flexible plug-in framework its users are able to write their own applications that can make use of several functions available in UPFrame. Among these features are:

- Buffers to receive incoming UDP packets at fast rates
- Smoothing functions to prevent data bursts
- Feeds to independent plug-ins that process the data.
- Automatic error recovery in case of partial framework crashes
- Released under the GNU public license

Although UPFrame was primarily written to process NetFlow UDP packets it is of a more general nature which means that it also can handle general UDP packets.

Processing

The DDoSVax project team has built a Linux cluster - called *Scylla [51]* - consisting of 22 compute nodes and multiple file servers with raid support that hold several TBytes of data. The data archives are mounted on all cluster nodes. Using OpenMosix [52] the nodes can migrate running processes to other machines which is useful for parallel processing. UPFrame and its helper programs can be run on several cluster nodes which allows to process independent NetFlow in parallel.

2.3 Host Identification Methods

As explained in previous sections we first identify hosts which run P2P clients. Then we use them to track other peers and to identify their traffic. P2P host identification methods can be classified into the following categories:

Polling Methods

If parts of a P2P protocols are known which allow to communicate directly with active peers, a polling method would contact a potential P2P client and send it any valid or invalid message (e.g. a *ping* or *hello* message). Analyzing the host's response or the absence of it, a peer can be identified with high probability. It is obvious that this approach only works online and requires at least an IP and a port number to contact a host. Depending on the P2P protocol a successfully identified peer can be used to get addresses of other active peers that can be polled. Another very serious disadvantage is that quite a lot of peers are behind firewalls, NAT boxes or proxies. Therefore they normally can't be contacted directly what makes polling techniques inaccurate.

In Subsection 4.1 a polling method is used to find a lower bound for the results of the implementation.

Packet Inspection Methods

Capturing and analyzing all the packets or partial packets which pass an Internet router is an accurate method to identify traffic as has been demonstrated in other publications. P2P protocols that use encryption will make it more difficult to use that technique in the future though.

The analysis of the packet payload can either be done online or offline. Real-time full packet processing unfortunately seems not feasible for larger networks with Gigabit transfer rates. Storing captured data of several days to disk for offline inspection is usually not feasible for moderately sized networks since the data amount often is very huge if the data is not filtered in some way. And even the capturing/storing process itself is non-trivial for high bandwidth connections as is shown in [40].

Flow Inspection Methods

A lot of large Internet routers generate flow-level data, mostly for accounting purposes but also for traffic identification. This compacted traffic logs include various information entries about connections that can be used for P2P identification. Flow-level heuristics usually don't need any detailed knowledge about the P2P protocol except for the default port numbers. More important is the architecture of the P2P network and how active its participating peers are.

In this thesis the focus mainly lies on flow identification methods. There are some subgroups among flow-based techniques:

Port-based

Storing port information about hosts and then analyzing them can be very efficient for P2P identification as will be shown. Although the amount of peers that use non-default ports is increasing, enough peers seem to use the default ports to allow an accurate identification. Moreover it has been observed that in some P2P networks like FastTrack most connections are located in a relatively small port range. This fact also can be used to identify peers and traffic.

IP based tracking

P2P traffic could be very accurately identified if all active peers of a network are known. This would be the case if the whole population could be tracked. As is shown later, tracking a substantial fraction of the population is possible for some networks.

Some P2P networks rely on super nodes which usually are used by ordinary nodes for search queries. Depending on the network there are from about 80 (eDonkey, see [53]) up to 30'000 (FastTrack) of them. Assuming that each ordinary client must have at least one connection to a super node, it would be relatively easy to determine P2P clients in an observed network if all super nodes were known. Basically it just has to be checked if a host connects to one of these super nodes. Super node tracking is possible for eDonkey, Limewire and even FastTrack. But especially tracking FastTrack super nodes is very difficult and requires a large effort as is shown in [27].

Traffic pattern analysis

As was described in Section 1.3 the P2P model works in a different way than the traditional client-server model. This also implies that their traffic patterns fundamentally differ in terms of connection count, flow lengths and upload-download ratio. A very general approach could concentrate on these characteristics to identify peers. It is assumed that this method will gain importance since more and more clients try to hide themselves using random ports.

Another classification can be made between methods that identify the kind of P2P traffic

Download Traffic

Most current P2P systems use unencrypted TCP HTTP for their downloads. This makes it rather easy for packet inspection methods to identify the traffic. Therefore these methods used in measurement studies usually only consider and account the download traffic.

Signalling Traffic

To communicate with other peers a custom and proprietary protocol is used by most P2P systems. Newer generation of P2P protocols use UDP for that purpose. Although download traffic in terms of size is far larger than signalling traffic, it should not be neglected since it can produce several MBytes traffic per hour without any downloads. Overall signalling traffic plays an important role in P2P identification since it is mostly independent from file transfers.

Depending on the P2P system, signalling traffic can be used to track almost the whole P2P population of a network. Signalling communication usually use the same listening port number as is used for downloads. Exceptions are eDonkey, Overnet and Kademlia which all use the same TCP download port numbers but different ones for signalling traffic.

2.3.1 Default Ports

Since we are using NetFlow data, this thesis focuses on flow-bases methods that mainly use port numbers for identification. Therefore in the following subsection an overview about port numbers is given.

The two most used transport protocols in the Internet are TCP and UDP. Both of them use port numbers to allow routing the packets to the process they are belonging to. These port numbers are assigned by IANA [54] and are divided into three ranges as shown in Table 2-1. Ports in the dynamic range are used mostly by TCP for outgoing connections.

While TCP is a connection-oriented and reliable protocol, UDP packets are non-connection oriented and not guaranteed to reach their target. Internet programs, communicating over UDP, normally use one single port for sending and receiving UDP packets as is shown in Figure 2-2, whereas TCP usually uses several ports for outgoing connections. When tracking instances of an overlay network one can use graph traversal to identify parts of the overlay network if the instances are somehow connected over a common property. In the case of

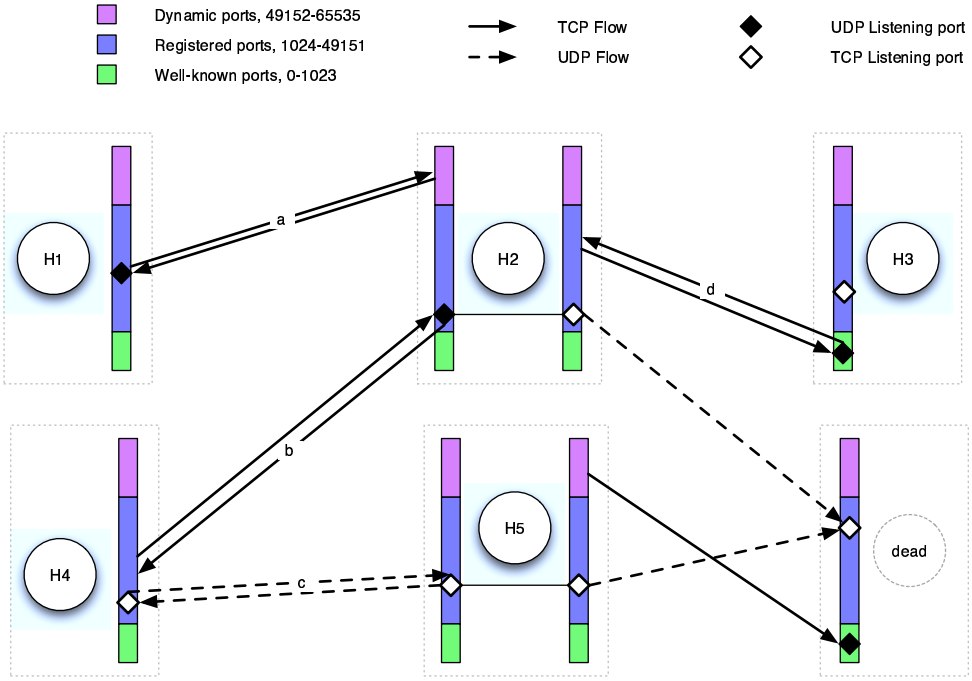


Figure 2-2: TCP and UDP connections in P2P networks

Range	Description
0-1023	Well-known Ports: e.g. WWW (80), SMTP (25), SSH (22), ...
1024-49150	Registered Ports: e.g. FastTrack (1214), Gnutella (6346), ...
49151-65535	Dynamic and/or Private Ports

Table 2-1: Official port ranges

P2P population tracking this common property are the port numbers. Looking at 2-2 the following statements could be made:

- Binding the same port for incoming and outgoing UDP packets allows to track other hosts by applying graph traversal. Looking at the UDP flows shown in Figure 2-2 it is very probable that host *H4* is part of the same overlay network as host *H2*, even if they have never contacted themselves directly with UDP packets.

- The TCP flows in general do not allow to determine if e.g. host *H1* is in the same overlay network as *H3*, even if they used the same TCP listening port. Their listening and outgoing ports are not "tied together" as it is the case for UDP. But if for example *H1, H2* and *H3* had TCP connections to each other and if they all would use the same listening port, it would be very probable that they are in the same overlay network.

The terms "remote" and "local" ports are sometimes used in the following. Since flows are unidirectional, a local port is not the same as source port and it is neither equivalent with the listening port. It depicts the port used in a flow on the side of an observed host. The remote port depicts the port used on the remote host.

Until August 2002 most P2P clients used their default ports[6]. In February 2003 about 38% of all download sessions did not use the Kazaa standard port according to measurements made for [55]. Since then more and more P2P clients have been using random ports for different reasons:

- Users change their P2P clients port number manually to evade restriction or limitation of P2P traffic by their ISPs. Commercial ISPs with a lot of private home users nowadays hardly can afford to block or limit P2P traffic since their customers would not tolerate it. Therefore the usage of default ports among home users is in general higher than that of otherwise connected P2P users. In contrast to that, universities and companies often limit or even block P2P traffic. E.g. ETH has set a P2P traffic limit of 10MBit/s which is shared by all users in the ETH network.

- Certain clients like Overnet or recent Kazaa clients automatically choose a random port when they start up the first time. They do this to evade firewalls or other limitations set up by the user itself or the network provider the host is connected to. Some even use the WWW port 80 if they can't communicate with other peers using other ports. As has been observed this is the case for less than 1% of all peers.

- Users behind a NAT box have to share one global IP for several users, but a certain port on the NAT box can be bound or forwarded only to one users. Therefore other users have to choose a different ports.

Firewalls, proxys and Network Address Translation (NAT) are in general big problems in P2P networks since they usually don't allow a P2P client behind such a device to be contacted directly. Their common principle is in general to allow all outgoing connections but deny every incoming connections unless

[6] See [8] and Appendix E

Range	TCP [hosts]	UDP [hosts]
1024-1999	2737	220
1024-2999	4734	313
1024-3999	8177	539
1024-4999	10442	642
1024-5999	12278	707
1024-6999	12519	780
1024-7999	12563	1122

Table 2-2: Hosts with suspicious flows

explicitly configured otherwise. Estimations for LimeWire state that about 60% of all its users are behind NAT boxes[7] because most home users have broadband modems with integrated firewalls/NAT boxes. Nevertheless this can be worked around either by the user itself or by the P2P protocol.

- Concerning firewalls and NAT devices: The P2P user can configure its NAT device to forward certain connections to hosts in the local net. Of course this works only if the user controls the NAT device. Firewalls can be configured to allow incoming connections on certain ports. Most computer users are not sophisticated enough to adapt their configurations though. So ideally the P2P protocol should provide ways to get around this problem.

- P2P protocols could use reverse calls where a requesting client A contacts the client it wants to download from B by a third host C, which is not protected by a firewall or a NAT device. Host C must already have a connection to B which then can be used to pass the request.

The dynamic port range usually is supposed to be used for outgoing connections. But as has been observed with various P2P clients less than half of them uses the dynamic range for outgoing connections. Clients like Kazaa start around TCP port 1050 and then count up for each new outgoing connection. Therefore it is possible to inspect a specific port range for P2P connections. A measurement made on May 27 2004 at 18:00 hour with NetFlow data shows in Table 2-2 that only a few thousand hosts out of 162'029 active SWITCH hosts at that time had connections with source and destination ports in such a range. The result is even more surprising for UDP hosts. It is assumed that a substantial amount of them are P2P clients.

The following subsection describes and analyzes some generic approaches for P2P identification. Then these methods are discussed for each of the considered P2P networks.

2.3.2 Generic Approaches

In the following some flow-based techniques to identify P2P hosts are described and discussed.

[7] See article "Gnutella on the Rise" http://www.slyck.com/news.php?story=530 (August 2004)

Protocol identification

The general approach to identify P2P clients of a specific network would be to analyze the protocol and then find methods that can be used to track down the peers. Unfortunately only a few P2P protocols (including e.g. the Gnutella protocol) are available for the public, most are proprietary and therefore matter of numerous re-engineering attempts like the pDonkey project [56]. The advantage of protocol identification is that protocol can't be changed as fast as the client implementation since the protocol must remain backward compatible.

Client identification

P2P clients often vary in their implementation which affects their performance (e.g. choice of super node [20]) and makes them distinguishable. One approach to identify peers could be to use these very specific properties of a P2P client or a part of the bundled software. These properties can include different traffic patterns in setup or hand shaking or the use of a client specific incoming or outgoing ports as is the case with eMule.

"Free" clients often only will work if ads get displayed. The ads come from a limited but changing number of ad servers whose IPs could be used to identify peers. Unfortunately the ad servers are also used by ads on WWW sites. Moreover there exist unofficial rogue clients like Kazaa Lite or iMesh Lite that are free from adware. Some P2P clients come with helper application like "PeerEnabler" [57] comes with Kazaa. PeerEnabler itself is kind of a P2P system which is some kind of content distribution system that can be used by other companies (for money) to push software to their customers. The fact that PeerEnabler is bundled with Kazaa and uses a default port can be exploited.

The advantage of a client identification lies in these very specific properties that can improve the results. But since clients change their behavior with every update, client identifications in general should be used only as support for protocol identification. Otherwise the identification algorithm must be updated frequently.

Traffic patterns

A P2P host, when transferring files, has lot of outgoing and incoming connections to different hosts on various ports. This in contrast to a web client that on one hand uses port 80 as remote port and has only outgoing TCP connections. So the number of contacted or requesting hosts could be used for identification.

Another property that separates P2P clients from common web clients is the traffic ratio which is more balanced for p2p clients. Web clients hardly upload as much data as they download, but P2P clients upload about the same amount of data as they receive.

To use packet count and packet sizes of setup or hand-shake connections was shown in [29] to be not very successful for identification, since hand-shake is too variable and too dependent on the implementation.

P2P protocols that use file swarming have block- and chunk-sizes for the transmitted files (e.g. 9.28 MB with sub chunks for eDonkey). Flows with corresponding payloads could be searched. Unfortunately only a few flows exactly match these defined block-sizes as our observations have shown, confirming results of [29].

Super node tracking

P2P networks like eDonkey, Gnutella and FastTrack rely on the existence of few powerful super nodes. Depending on the network there exist from 80 up to 30'000 of them. In the case of eDonkey they are static and in fact are not ordinary peers but dedicated servers. Gnutella and FastTrack self assign super node status. These super nodes are part of a two-tier system where the ordinary clients must have one or more connections to one of these nodes. Typically the servers have to maintain the connection with ping/pong messages that are sent periodically when there are no search queries (e.g. every 412 seconds for eDonkey or every 112 seconds for FastTrack). If it is possible to track all the super nodes worldwide, P2P clients for the networks with super nodes could be identified with a straightforward method (see Subsection 4.1).

Tracking all eDonkey servers is feasible with a managable effort because of the following reasons:

- The eDonkey servers are dedicated and static, so they don't change very often
- There exist worldwide only about 80 active and populated servers which makes it easy to track them
- They communicate with each other to exchange server lists
- The eDonkey protocol was subject to various reverse-engineering attempts [56, 58]

These conditions allow to write a crawler program that queries every server for its server list. That way new servers could be found and queried too for their servers list. There exist a few sites like [53] which exactly do that.

For FastTrack it is rather hard but not impossible to track all super nodes for the following reasons:

- The protocol is not well reverse-engineered since it uses encryption. The open source client mlDonkey, that is also able to join the FastTrack network, does only partially support the protocol.
- There are between 20'000 and 30'000 super nodes worldwide [59, 27] which is a large number to track within short time.
- Super node status is self-assigned and may change with time. In [59] they state an average life-time of a super node of about 2.5 hours. Therefore the tracking should be done in less than this time.

Nevertheless it seems to be a possible to track all super-nodes worldwide within an hour using a very sophisticated crawler program and about a dozen of Linux computers as it is described in [27].

According to [60] for Gnutella there existed about 3684 ultra-peers at August 12. 2004 at 0.16 pm. This seems to be a realistic number of hosts to crawl and since the Gnutella protocol is open source this should not be as hard as in the case of FastTrack

Beside its accuracy another good thing about super node tracking is that peers can be identified relatively early with a high probability because of their

Country level domain	Occurrence	Country level domain	Occurrence
ch	156 (27.61%)	be	24 (4.24%)
net	117 (20.7%)	es	22 (3.89%)
fr	51 (9.02%)	de	20 (3.53%)
com	41 (7.25%)	il	17 (3%)
pl	30 (5.3%)	uk	15 (2.65%)

Table 2-3: Top level domain names of 565 Kazaa peers (TCP)

bootstrapping which is a very important process of P2P protocols. Clients usually will try to connect to several known super nodes. If the contacted super nodes were tracked and therefore known, these connections confirm that a host is a peer.

Reference of locality

A P2P identification method which uses flow-based methods can be improved for some P2P networks running own clients. P2P protocols like FastTrack use locality explicitly when a client chooses a super node. Our measurements shown in Table 2-3 with Kazaa running in super node status show that the test client clearly was preferred by "local" clients. The results confirm the observations made in [59] which assumed that FastTrack takes locality into account when an ordinary client chooses super nodes.

In the case of eDonkey this reference of locality could also be observed in [61] but may be more content dependent in the sense that P2P users tend to search "localized" content. So it can be assumed that most P2P networks use locality either by the protocol or at least by the content of the shared files.

As Table 2-4 shows not only the choice of super nodes shows some signs of locality but also the origin of search queries. In comparison to the user base of the corresponding countries, there are quite a lot of European country level domain names among the requesting peers. Surprisingly most of the UDP packets came from hosts with the nl country-level domain. One explanation could be that the P2P population in the Netherlands is higher than in other European countries. There are some reasons why the P2P user base in the Netherlands may be large in comparison to other European countries. The FastTrack protocol originally was developed in the Netherlands where also other P2P projects like the Gnutella group are located. Moreover the Dutch supreme court was the first high court when it in December 2003 affirmed the legality[8] of P2P file-sharing technologies like Kazaa. The Netherlands are rather P2P friendly and it is possible that Dutch people use P2P software more often than other European users. But probably there other unknown reasons for this result, since the number of hosts in the nl-domain is just too high. Overall it is assumed that the locality of hosts contacting the test super nodes by UDP presumably is content dependent.

In general, running one or more super nodes in a larger network probably can be used as a kind of *honey pot* for local P2P clients since they seem to prefer

[8] See Slyck.com article RIAA and MPAA Ask, Why Me? http://www.slyck.com/news.php?story=353 (August 2004)

Country level domain	Occurrence	Country level domain	Occurrence
nl	1026 (25.84%)	be	115 (2.89%)
com	866 (21.81%)	ch	107 (2.69%)
net	527 (13.27%)	at	106 (2.67%)
se	281 (7.07%)	fi	89 (2.24%)
fr	148 (3.72%)	de	82 (2.06%)

Table 2-4: Top level domain names of 3'975 Kazaa peers (UDP)

	k=1	k=5	k=10	k=30	k=50	k=100
BitTorrent	1149	520	373	135	105	89
FastTrack	2586	702	175	99	82	64
Gnutella	738	113	78	46	35	21
eDonkey/Overnet/eMule	4720	547	390	292	267	241

Table 2-5: Hosts which had at least k flows with P2P default ports

them either due to the protocol or due to the content provided.

Stateful port number method

A decrease in default port usage of P2P clients has been observed since 2002. While this decrease makes it harder to directly identify P2P clients, the default port numbers still are widespread enough to use them for identification as is demonstrated later. Since it is far from accurate to declare a host a peer when it uses a single P2P default port the approach must be stateful and identification can't be done instantly but after a potential peer has been observed some time. A one day measurement which counted all SWITCH hosts from three of the four border gateway routers in periods of 15 minutes that had at least k flows with P2P default ports is shown in Table 2-5. Since eDonkey, Overnet and eMule use the same download port, they are accumulated. For hosts which used default ports from several P2P networks the most dominating was chosen.

This result shows that it is non trivial to set a reasonable threshold to determine a peer just by its flow default port usage. Obviously there occur a lot of non-P2P connections that use P2P default ports. An explanation for this could be that, as it is the case for many P2P clients, many Internet applications don't use the dynamic port range for the local port of outgoing connections. So they may start with port 1024 and then count up for every new connection made which leads sooner or later to a P2P default port. Of course NAT boxes and proxy servers also contribute to this effect since they normally use the whole non-well-known port range for outgoing connections.

False positives will be high for this approach if k is too low and false negatives would be rather frequent if the threshold is set too high. Moreover this method is not accurate since it does not account that flows - especially UDP flows - to the same host can be quite frequent and therefore have a greater impact. A better identification method would check the relations of a potential peer to other hosts and count flows between the same hosts only once.

A promising approach of a stateful identification method is to check a peer's neighborhood. If a P2P client or user changes its listening port to a random port and not all of its neighbors are doing the same - which is rather unlikely for most P2P networks except FastTrack - the hiding peer has to communicate over the default ports anyway to contact its neighbor peers. The *"good"* neighbor peers, which use default port numbers, therefore *"betray"* the *"bad"* peers that try to hide themselves. The most important approach of the P2P identification used in this thesis uses this fact. Therefore it is explained in greater detail in the following:

Most Used Remote Port Approach (MURP)

As was explained above, just counting the flows that have default port numbers is not accurate enough since it could all be flows from and to one single neighbor host, which would falsify the results. One has to differentiate between the neighbor peers. This is done saving the last N contacted hosts and their used remote port numbers if they contacted the host under observation in a certain port range, in the following called "suspicious port range". When there were enough IP/port pairs collected, a chart with the most used port numbers of contacted neighbor peers is generated. Depending on the most used remote ports it can be determined to which P2P network this host is belonging to if at all. Since it is rather unlikely that a non-P2P application has contacted several other hosts on the same port which also is used as a P2P default port, this approach is supposed to be quite accurate concerning false positives. Using the same method for local ports, it is also possible to determine the TCP/UDP listening ports of a peer, which then also can be used to improve identification or to track other peers.

This approach, in the following called MURP ("most used remote port"), has been found quite effective and proved well for all P2P networks considered in this thesis.

In the following Subsections the identification algorithm used in the implementation (see Chapter 3) for each of the considered P2P networks is discussed. In most cases it basically is the stateful port method described in this Subsection, sometimes extended by auxiliary methods.

2.3.3 Host Pool

For the following identification methods to work some information about potential peers have to be gathered first. Therefore a pool of many internal (and later also extern) peers which are under observation is used. There are several ways a host could get inserted in that pool

1. P2P default ports used in a flow:
   Since the default-port usage in general still is high enough this method works quite well. Only for FastTrack with a decreasing number of peers which use port 1214 this method may not be work well enough.

2. Suspicious port range:
   Some P2P clients use a small band or ports that they use for incoming and outgoing connections. For FastTrack e.g. it was observed that 86% of all

connections had source and destination port in the range of 1024-4000. As is shown Table 2-2 there are relatively few active hosts which have flows in such a small port range and it is assumed that most of them are P2P hosts. Hosts which have flows in such a "suspicious range" could be added as well to the host pool.

3. External tracked peers contact unknown internal host:
   An internal host that is contacted by a known external peer probably is a peer too.

Evaluating showed that method 1 is the most efficient. The host pool increased by a factor of 5 when method 2 was applied but had a marginal effect on the number of identified peers (less than 1%). This was also the case for method 3 which only marginally influenced identification but dramatically increased the host pool. Since a large host pool needs a lot of RAM it is preferable to keep it small.

2.3.4 Non-peer Identification

Before distinguishing the peers it is important to define some requirements that they must fulfill so that it may be considered as peer. After the analysis of numerous peers, including the test client, some reasonable rules are of the type:

- A peer must have at least *MinTCPConn* TCP connections to other hosts within a time *ProbationPeriod*
- There must have been at least *MinTCPBig* TCP-big flows to other hosts within *ProbationPeriod*
- More than *MinSuspConn* connections in the suspicious range must have been made within *ProbationPeriod*

See Section G for reasonable default values.

2.4 Identification for Concrete P2P Systems

2.4.1 eDonkey Identification

Strongly related to each other, eDonkey, Overnet and Kademlia clients all use the same TCP download port 4662. This makes it hard to distinguish them unless the signalling traffic is part of the identification too. Most measurements in other publications in fact don't separate them. While eDonkey clients only use few UDP packets to query other eDonkey servers than the one they are connected to by TCP, Overnet and Kademlia use quite a lot of UDP signalling traffic what can be used to distinguish them from eDonkey.

eDonkey clients mainly use TCP connections with the default port 4662 as is shown in Table 2-6. This measurement was done running an eDonkey server for 24 hours and analyzing the successfully logged in clients. It seems as if all ports except the default port 4662 were set by the users. Surprising is only the third most used port 6346 which is the Gnutella default port. One reasonable explanation for this would be that a hybrid client like mlDonkey which also

TCP port	Occurrence (22'093 peers)	Percentage
4662	16'497	74.67&
4661	606	2.74%
6346	224	1.01%
80	222	1%
5662	213	0.96%

Table 2-6: eDonkey and Kademlia top 5 TCP listening ports

connects to the Gnutella network, uses the same TCP port for different P2P protocols. Observing the official eDonkey client on the test host showed that about 68% of all TCP source ports connecting to destination port 4662 are in range 1024-6000.

The approach that showed best results is the stateful most used remote port method (MURP).

A host H is identified as an eDonkey peer if the following requirements are met:

- The most used local or remote TCP port of H is 4662 and this port was used by at least *MinPeerPortThreshold* unique hosts
- The most used local or remote UDP port of H is 4665 and this port was used by at least *MinPeerPortThreshold* unique hosts or H is neither a Kademlia peer nor an Overnet peer.

2.4.2 Kademlia Identification

eMule started as an eDonkey client and became very popular even exceeding the official eDonkey clients in terms of users. Since April 2004 the eMule client optionally connects also to a new Kademlia network (or just *Kad*). This makes eMule a hybrid client that has one foot in the eDonkey and one in the Kademlia P2P network. Kademlia uses the XOR routing [43] approach that can be used in completely decentral networks. On a test client after a 24 hour run eMule had contact with more than 180 unique hosts per Minute (see Table 2-13) which makes it relatively easy to identify with the standard approach because Kademlia uses the default port 4672 for UDP signalling traffic. Unfortunately this port is used for both, the eDonkey and the Kademlia messages, why the number of identified Kademlia hosts in fact is the number of eMule clients and not the number of hosts actually in the Kademlia network.

A host H is identified as a Kademlia peer if the following requirements are met:

- The most used local or remote TCP port of H is 4662 and this port was used by at least *MinPeerPortThreshold* unique hosts
- The most used local or remote UDP port of H is 4672 and this port was used by at least *MinPeerPortThreshold* unique hosts.

UDP Port	Occurrence (92'933 peers)	Percentage
4665	491	0.5%
4672	473	0.5%
15848	283	0.3%
4662	204	0.2%
1025	200	0.2%

Table 2-7: Overnet top 5 UDP listening ports

2.4.3 Overnet Identification

Overnet has no default UDP listening port. The first time the client is started it chooses a random port number above the well known range. A 24h run of the test client showed that 55% of them are in the range 1024-10'000 and only 0.56% of the used ports were in the well-known range, most of them on port 80 (UDP WWW) and 53 (DNS). In Table 2-7 the most used ports are shown. It is also interesting that among the most frequently used ports there are a few default UDP listening ports of eMule (4672) and eDonkey (4665) which are not part of the Overnet network. There is no peak since no default port exists for Overnet. This makes it makes it a bit harder to identify Overnet peers. But since eDonkey and eMule both have default ports, an identification approach can check for a random port distribution to determine an Overnet peer.

Interesting to mention is also that the test client was contacted by 30 different hosts in the 30th hour after it was shutdown. Within the first 24 hours after it was stopped 5'445 different hosts sent 12'380 UDP packets to the used listening port. This effect will be used for tracking external peers as is described later.

A host H is identified as an Overnet peer if the following requirements are met:

- The most used local or remote TCP port of H is 4662 and this port was used by at least *MinPeerPortThreshold* unique hosts
- At least *MinPeerPortThreshold* unique hosts have contacted H and the occurrence of the most used UDP remote port is less than *MaxPeerPortThreshold*.

2.4.4 Gnutella Identification

Gnutella as one of the older P2P clients that was born directly after the Napster shut down, has since then been constantly evolving. While it was completely decentral in the beginning it today supports a two-tier architecture with so called *ultra peers*, which are powerful peers that self-assign this status to themselves to cache file index data for ordinary peers. Every ordinary peer has - depending on its network connection speed - several (typically 4-10) connections to ultra peers. These connections are rather dynamically and often change. Gnutella uses the default port 6346 and 6348 and in the beginning used TCP connections only. Nowadays UDP is used too by newer clients that also support Gnutella2, a disputatious but extended protocol introduced by the P2P client Shareaza.

According to the Gnutella network crawl statistics of [60], Limewire is the most used Gnutella client which is used by about half of all Gnutella users. In a

TCP port	Occurrence (1'724 peers)	Percentage
6346	1027	59.6%
6348	482	28%
6349	90	5.2%
6350	50	2.9%
6351	15	0.9%
UDP port	Occurrence (3'764 peers)	Percentage
6346	2440	64.8%
6348	1018	27.0%
6349	190	5.0%
6350	59	1.6%
6351	19	0.5%

Table 2-8: LimeWire top 5 listening ports

UDP port	Occurrence (63'221 hosts)	Percentage
1214	2620	4.14%
32656	789	1.24%
1530	36	0.05%
3636	36	0.05%
1193	35	0.05%

Table 2-9: Kazaa top 5 listening ports

12h run the TCP port usage of Table 2-8 resulted. As can be seen most Gnutella peers still use their default ports or ports near to them. Therefore chances are good, that the identification is successful using the stateful port method.

A host H is identified as an Gnutella peer if the following requirements are met:

- The most used local or remote TCP or UDP port of H is in range of 6346-6349 and this port was used by at least *MinPeerPortThreshold* unique hosts.

2.4.5 FastTrack Identification

In Table 2-9 the UDP port usage after a 16 hour run of Kazaa as super node is shown. Since Kazaa uses the same port number for UDP and TCP listening ports it may be assumed that the port distribution for TCP is equal to that of UDP. An ordinary client receives far less UDP packets and maintains basically one connection to a super node plus connections to other peers that download or upload. There are at minimum 8 packets sent to the current super node within 15 minutes.

A super node in contrast has about 35 outgoing TCP connections which remain constant after some time and a large number of incoming TCP connections from other super nodes and ordinary clients. The number of established incoming connections was at 205.

As can be seen in Table 2-9 the default port usage is surprisingly low. The reason for this is that since version 2 of the Kazaa client, it incorporates some stealth techniques to make it harder to block or limit its download traffic. Choosing a random port number for the listening ports is one of these measures. This makes it rather hard to identify FastTrack peers. But fortunately the situation is not that bad yet. First of all iMesh, which is the second most popular official client, still seems to use the default port mostly and since - according to Slyck.com - there still are about 800'000 iMesh users in the FastTrack network which is about 25% of all users. These 4.14% default port usage observed in Table 2-9 could be explained with the fact that the test client was run as a super node which communicated quite frequently with other super nodes. Running iMesh as super node seems was never achieved. Explanation for this is that iMesh uses an older FastTrack library than Kazaa and so may not be able to run in super node mode. Therefore the Kazaa test super node in fact may have primarily communicated with other Kazaa super nodes and some ordinary clients with few iMesh or older Kazaa clients among them, which were using default ports. Measurements made for [59] in January 2004 show in Table 3-4 that the default port usage was approximately 13% at that time. Today it seems to be lower but still sufficient to identify FastTrack peers.

Second, what sets Kazaa apart from e.g. Overnet which uses a random UDP port too is the fact that Kazaa uses the same port number for TCP and UDP which is not very common for other applications. The third fact that helps to identify FastTrack clients is the application *PeerEnabler* [57] which is installed together with Kazaa. This program is meant to "distribute files efficiently and securely over the Internet", using a similar principle like BitTorrent. Since PeerEnabler uses 3531 as default port and since it is bundled with Kazaa this can be used as supporting identification method.

A host H is identified as a FastTrack peer if *one of the following requirements* are met:

- The most used local or remote TCP port of H is 1214 and this port was used by at least *MinPeerPortThreshold* unique hosts.
- The most used local TCP port is the same as the most used local UDP port and both were used by at least *MinPeerPortThreshold* unique hosts. Additionally port 1214 was used by at least one neighbor peer.
- The TCP and UDP remote port distributions are random with peaks smaller than *MaxPeerPortThreshold* and at least two of the four TCP/UDP ports 1214/3531 were used in a flow.

2.4.6 BitTorrent Identification

BitTorrent doesn't consist of one large network like the other discussed P2P networks but there are many thousand small networks, where all participating peers are downloading the same file. Download of the file is done using the *tit-for-tat* principle, the more a peer uploads data to other peers, the more he receives from them. This is a proven counter measure against leechers that are a problem in other networks. BitTorrent uses a total of 10 default TCP ports. Port 6969 is the reserved default ports for trackers, although measurements

TCP port	Occurrence (2559 peers)	
6881	1285	50.2%
6882	325	12.7%
6883	208	8.1%
6884	127	5%
6885	81	3.2%

Table 2-10: BitTorrent top 5 listening ports

done in April 2004 with 65 different trackers have shown that about 50% of all trackers use this default port.

If a BitTorrent user wants to download a file he first has to find either a torrent file or a link to a Tracker that serves the torrent file. So there is no search function included in BitTorrent. Therefore various torrent archives have been set up where users can publish torrent links. The largest of them is Suprnova[22] which claims to have about 60% of all trackers worldwide in their database. A list of maximum 40 peers is sent to new clients which then contact the other peers themselves. This means that a new BitTorrent client that starts downloading a file will open 40 connections to other peers. Most of the successful connections then are held open since the neighbor peer may have chunks that were not downloaded yet. Connections are bidirectional and are used for signalling and download traffic at the same time. BitTorrent peers don't contact as many neighbor peers as e.g. Overnet.

On the client side the range 6881-6889 is mostly used. Analyzing the peer lists of 96 different torrents the port usage of Table 2-10 resulted. In total 86% of all peers used the default ports. The fact that there is not only one default port but ten makes identification with to most used remote port approach a bit slower since for that method it was assumed that most of the peers use only one default port. So the threshold that is used to determine if a host is a peer may not reached that fast. Therefore identification is extended by a probability value that increases with the count of the used default ports. This improves identification time.

A host H is identified as a BitTorrent peer if one of the following requirements are met:

- The most used local or remote TCP port of H is in range 6881-6889 and this port was used by at least *MinPeerPortThreshold* unique hosts.
- The probability value p of H is at least 0.5. p i*s* calculated as follows*:*
  - Add 0.4 to p if more than 6 default ports were used by neighbor peers
  - Add 0.3 to p if more than 4 default ports were used by neighbor peers
  - Add 0.2 to p if more than 2 default ports were used by neighbor peers
  - Add 0.4 to p if at least 3 default ports were used including 6969
  - Add 0.05 up to 0.25 for each single default port used depending on their usage (see Table 2-10)

2.4.7 Undetectable and Safe P2P Usage

After having examined several methods of P2P identification which were mostly successfully we also have an understanding how to design a P2P client that makes its detection rather hard. Such a client would:

- Use a proprietary protocol (security by obscurity) that is hard to reverse engineer and difficult to actively poll.
- Encrypt signalling and download traffic. Downloads could be encrypted with an asymmetric key pair using key exchange algorithms. This would prevent packet inspection.
- Use a two-tier architecture where the super nodes are dynamically assigned and change frequently.
- Choose random port numbers in the whole port range and may be even change the port numbers from time to time (*port hopping*).
- Use random ports for outgoing TCP connections and random UDP listening ports for signalling traffic. That way the host can't be scanned reliably to determine if the client is running.

To hide a P2P client is one thing but today the threat of being sued is more important. Even with today's P2P clients there are ways to evade being sued by the RIAA or similar organizations in other countries. Means to do so include:

- Don't use FastTrack or the current most popular network since they were and will be the RIAA's main target. In August 2004 they started to sue eDonkey and Gnutella users too[9]
- Use an anonymizing trusted proxy, e.g. JAP[10] when running file-sharing programs.
- Don't share any or only legal files. But under no circumstances share music files which are in the top 40 or currently playing movies. Sharing no files at all is not appreciated in the P2P community and may result in slow download rates
- Use IP blockers like Protowall [62] or Peerguardian [63] with up-to-date block lists that give some security by stopping peers that are know to be operated by the RIAA or other P2P unfriendly organization.

2.5 Population Tracking

One goal of this thesis is trying to track all peers worldwide. As will be shown, this seems to be very difficult in general. Nevertheless the implementation tracks a substantial amount of external peers for all considered P2P networks in order to monitor their overall activity which is expressed in the amount of tracked external peers.

[9] See Slyck.com article RIAA Sues 744 Individuals More http://www.slyck.com/news.php?story=552 (September 2004)

[10] See university of Dresden's free service JAP - Anonymity and Privacy http://anon.inf.tu-dresden.de/ (September 2004)

After 24 hours run	Received	Sent	Total
Unique hosts packets	92'933	92'497	94'046
UDP packets	444'896	493'558	938'454
UDP KBytes	13'636	37'705	51'341

Table 2-11: Overnet activity of client on test host

2.5.1 Conditions

To track all peers of a P2P network worldwide some conditions must be met:

- All peers of the same type must be connected to the same network, i.e. no network fragmentation
- A substantial number of peers must be observable, they serve as base population
- Contacted peers must somehow be identifiable as such, e.g. through port numbers
- Communication activity of these peers must be very high, e.g. they must frequently contact other peers, even without file transfers

Of the six considered P2P networks only two fulfill all of these requirements. BitTorrent consists of numerous independent and unconnected networks. Gnutella, FastTrack and eDonkey use a two-tier architecture where super nodes masquerade the activity of ordinary peers what reduces their activity. Gnutella at its early ages was a one-tier architecture too but even then a host mainly had contact with a few neighbor peers.

Concerning Overnet and Kademlia; since they are completely decentral P2P networks with a one-tier architecture that makes heavy use of UDP messages - due to the XOR search algorithm - the possibility to track all peers worldwide was expected to be rather high. The official clients, run on the test host, showed that the UDP usage is massive for both of them. Tables 2-11, 2-12 and 2-13 give an idea about the activity of the Overnet and eMule test client. It's noteworthy that these numbers may be higher than for an average Overnet/eMule peers since the rather powerful test host is located in an unfiltered net that has direct 100MBit access to the Internet. Looking at the total transfered bytes it gets obvious that both protocols have quite a large overhead of about 2-4 MBytes per hour which is needed for network maintenance and search queries. Interesting - concerning the tracking - is the number of 94'046 unique hosts contacted by the Overnet client. At that time the overall Overnet population was about 920'000 users. So during these 24 hours the test client has had contact with about 10% of all hosts worldwide. Of course most of the contacted hosts probably did not remain online during that time but nevertheless quite a substantial number of the total users were contacted.

In June the Overnet population was about 800'000 users daily according to Slyck.com [41] and the client which can report user numbers. At the same time the eDonkey network counted about 2.4 million users. Assuming that a bit more than 50% of the eDonkey users run eMule (estimations go up to 85%) and

per minute	Received	Sent	Total
Unique hosts packets	63	63	64
UDP packets	304	338	642
UDP Bytes	9'564	26'445	36'010
Average UDP Bytes/Packet	31	78	56

Table 2-12: Overnet activity of client on test host

per minute	Received	Sent	Total
Unique hosts packets	178	185	186
UDP packets	533	542	1075
UDP Bytes	35'331	38'516	74'660
Average UDP Bytes/Packet	66	71	69

Table 2-13: eMule activity of client on test host

that about 13% of the eMule users have Kademlia support enabled (according to statistics of a self run eDonkey server) this would give a population size of about 156'000, which probably could be tracked. As is shown in Section 4.4 far more hosts than this were tracked for the Kademlia network since unfortunately eMule uses the same port to contact other peers in the Kademlia and the eDonkey network. Therefore they can't be distinguished clearly. This was the reason it was decided to analyze Overnet peers in more detail.

2.6 Example: Overnet Tracking

Tracking was done as follows:

1. Specify a number of safe "seed" Overnet peers (10 internal peers were used) and their listening ports as starting population

2. Declare any host an Overnet peer that contacts or is contacted by one of the already known peers on its known listening port

Since Overnet chooses a random UDP listening port as does the only non-official Overnet client mlDonkey, there should be no peaks in the port usage[11] of the tracked hosts. But in the first try a lot of hosts with port 53 (DNS), 123 (NTP) and 137 (netbios) showed up. Overnet clients sometimes seem to use their listening port number for DNS queries. Therefore hosts contacted using these ports were filtered in a second try.

There still were substantial peaks (up to 33%) at 4672/4673 (eMule), 6346 (Gnutella) that led to the conclusion that some Overnet peers occasionally also contact eDonkey, Kademlia and Gnutella peers, possibly because of hybrid clients like mlDonkey which connect to several P2P networks. As soon as a peer from another network is tracked as an Overnet peer this other network will be tracked too without further measures. After ignoring the ports 137, 135, 123,

[11] See Table 2-7

UDP Port	Occurrence (154'093 peers)	Percentage
4665	1023	0.66%
4672	819	0.53%
4662	417	0.27%
5468	340	0.22%
15848	306	0.19%

Table 2-14: Overnet top 5 UDP listening ports of tracked hosts

53 and not tracking hosts that were contacted using the local UDP ports 4672, 4673, 6346, 6348 and 5672 for outgoing packets, the port distribution shown in Table 2-14 was almost the same as in Table 2-7. Analyzing only one hour of NetFlow data (6 pm of a Tuesday during the semester) resulted in 154'093 tracked Overnet peers, which corresponded almost to 20% of the total Overnet population. This first result was promising but as is shown in Section 4.4 tracking of the whole Overnet population seems to be very hard even if more internal Overnet peers are used to track external peers.

2.7 Traffic Identification

Another goal of this thesis is to identify P2P traffic of peers within the SWITCH network. It was assumed that a substantial amount of P2P traffic is transmitted on non-default ports and therefore can't be accounted using straightforward traffic identification methods. Identifying P2P hosts is an intermediate step to improve the accuracy of P2P traffic estimation. Knowing that one or more P2P clients run on a specific host is one thing, but then accounting the traffic on non-default ports of this host to the correct P2P client is a non-trivial task that gets even more difficult if there are multiple clients running on that host.

Most P2P programs use several listening ports. Examining P2P clients on the test host showed that e.g. the Kazaa client was listening on three UDP ports and on two TCP ports. Hybrid clients can even be accepting connections on more ports. Detecting all of them would need a disproportional effort.

Moreover outgoing TCP connections are set up independently from the listening port by most applications. Usually an unused port above the well-known range is chosen for the local port and the listening port of the remote peer to set up a connections[12]. So outgoing TCP connections, which are primarily used for downloads and therefore important to identify, are not easy to distinguish and account either unless they use default ports. Of course other applications running on the same host normally use default-port numbers and could be ignored using this information. But this would require a complete and up-to-date data base with all known port numbers that would have to be updated frequently since more and more Internet applications are developed. This may be inaccurate if the P2P clients chooses a port randomly.

The used estimation approach is based on the assumption that hosts running P2P clients do not run other applications with listening ports in the non-well-known port range. Such applications typically include video/audio streaming or

[12]See Section 2.3.1 for more information about ports

online games. Both of them them are sensitive to network latency and get the full attention of the user while running. Running a P2P client at the same time will noticeably decrease their performance which makes this situation rather unlikely to happen. Since only a very small fraction[13] of P2P users manually sets the listening port of their client to a port number in the well-known range and since most users seem to read Emails and surf the web while file-sharing, all flows where either source or destination port is in the well-known port range are not considered for the P2P traffic estimation.

The bandwidth estimation works as follows:

1. All flows where either source or destination port is in the well-known range are ignored for the bandwidth estimation
2. Flows with P2P default ports are accounted to the corresponding network. All other flows count as unidentified.
3. All unidentified traffic for observed hosts is summed up and if the internal host it is an active peer directly accounted to the P2P network this host is belonging to according to identification
4. If a candidate peer gets identified as an active peer, its so far generated unidentified traffic is accounted to the corresponding P2P network. This way the identification latency has less impact on the bandwidth estimation.

2.8 Limitations

As described in Section 2.7 the traffic estimation is based on the assumption that P2P hosts don't run other applications in the non-well-known port range. This seems to be true for most hosts but definitely not for all hosts. Therefore the estimated P2P traffic trends to be higher than the actual P2P traffic of identified peers. Moreover the accounted traffic per P2P net may not be accurate since only one of possibly several P2P clients per host is identified and therefore all unidentified traffic from that host is accounted to one network unless default ports are used.

The longer the identification takes for a host the less accurate will the traffic measurement be since the transmitted data of an identified peer is accounted for the current time step although that traffic may have been from previous periods. But since the identification time for most hosts is about the time of one evaluation period[14] this is not of great importance.

Peers that have only few connections need more time to be identified since the proposed algorithm benefits from large activity. Fortunately most P2P clients contact many other hosts during the bootstrapping process, so that for most hosts this is no problem. It is though a problem for clients that don't have much signalling traffic using default ports and that don't upload or download anything. But since we primarily are interested in the P2P bandwidth consumption and since these peers don't cause much traffic - otherwise they would be more active - they are not of great concern.

[13] Our measurements have shown that for all of the considered P2P networks less than 1% of the peers use listening ports in the well-known port range

[14] See Table 4-7, "Identification Time"

Chapter 3

Implementation

The proposed identification methods of Chapter 2 were used for several implementations that are subject of this and the following Chapter.

3.1 Offline Scripts

To evaluate some identification methods several Perl scripts were written. They are well for experimenting but not fast enough to analyze several days of NetFlow data within reasonable time. Some of the more useful scripts are the *SafePeer* scripts, which can be used to rapidly collect some P2P hosts that have a very high probability to actually be peers. These scripts won't find all the peers though, since the requirements to be considered a safe peer are quite high. Most of them use the same approach as the plug-in described in the following section. The *verifyPeer* script will try to poll active peers in order to verify them. It can be used only if the port number of a peer is known or if it uses the default port. Peers that are behind firewalls or NAT devices can't be polled.

3.2 Online Plug-in

As a proof of concept and to be able to actually measure the P2P traffic in the SWITCH network an UPFrame plug-in called *PeerTracker* was developed under Debian Linux using C.

One of the disadvantages of online processing with UPFrame is that the sending rate can't be dynamically adapted. This either causes packet loss or takes more time as needed to process the data. Therefore the plug-in may also be used in a stand-alone mode (without UPFrame) where the NetFlow data is read from a named pipe which is filled by *netflow_sreplay* on the same local machine. This allows flows analysis to be as fast as possible without packet loss. See Section 3.4 for an overview about how fast PeerTracker can process data and how much resources are needed.

3.2.1 Identification Algorithm Overview

As was described in Section 2.3 the identification algorithm makes heavy use of the MURP approach that uses the neighborhood relations to determine a peer.

That approach assumes that internal peers can be reliably detected when the relations to their external neighbors are examined. As is done in most of the offline scripts too, PeerTracker reads NetFlow data, decomposes and analyzes it as is shown in the pseudo code Algorithm 1.

```
For all flows f do {
 if (EvaluationInterval.exceeded()){
  evaluateHosts(HostPool);
  writeStatistics();
  resetStatistics();
  }

 updateGeneralTrafficStatistics(f);

 if(f.srcport > 1023 or f.dstport > 1023)
  continue;

 if (h.internal and !HostPool.exist(h) and f.usesP2PDefaultports())
  HostPool.add(h)

 if (h.intern and HostPool.exists(h))
  updatePortStatistics(h);

 if ( isSendingHost(h)
    and HostPool.exist(h)
    and
    (f.srcport == h.detectedListeningPort ()
     or withinSuspiciousRange(f))
  )
  updateTimeStatistics(h);
}
```

Algorithm 1: PeerTracker identification algorithm

If source or destination port of a flow use a P2P default port, the corresponding internal host first is added as *candidate peer* to a host pool. Beside the candidate peer status the known hosts in the host pool can have one of the following states. The state may change every time a host is evaluated:

Candidate peer Internal host that used a P2P default port and is going to be evaluated earliest after the *ProbationPeriod*. Candidate peers are either identified as peers or as non-peers. They are only identified if there is enough information available to safely determine its state, otherwise it remains candidate peer at most for the time *MaxCandidateLife.*

Non-peer Internal hosts that were evaluated and found to be not peers. Non-peers are immediately deleted after evaluation. Active peers can't change their status to non-peers immediately.

Active peer Internal or external hosts that ere evaluated and found to be a

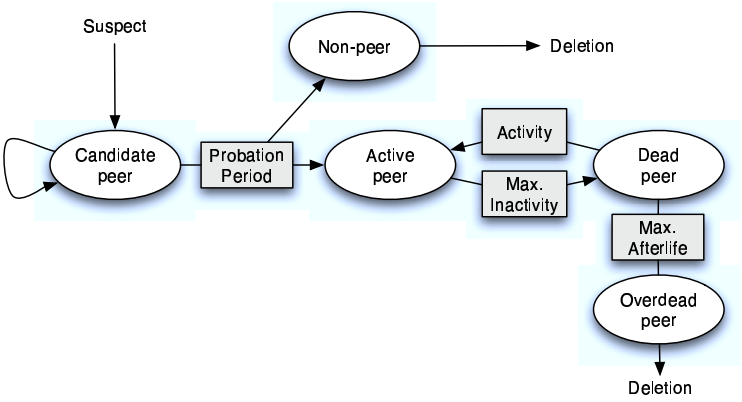


Table 3-1: States for internal hosts

peer. Hosts remain active as long as they send data on their detected UDP listening port or in the suspicious port range.

Dead peer Formerly active peers whose timeouts *MaxPeerInactivity* have exceeded. Although they seem to be dead, other peers will try to contact them. This fact is used to track external peers. If a host is active again during his after life it gets assigned active peer status again when it is the next time evaluated.

Overdead peer Dead peers whose timeout *MaxAfterLife* is over are immediately deleted.

As can be seen in the state diagram of Figure 3-1 a candidate peer remains in the pool for a certain time where it is observed but not deleted. After this *ProbationPeriod* it can get deleted or assigned another status. The status of a host is changed after they were evaluated which is done periodically after *EvaluationInterval* of processed flows. *EvaluationInterval* was set to 15 minutes for all measurements in this thesis. Figure 3-1 shows the amount of candidate peers after evaluation. There were more candidate peers before all the hosts were evaluated. Most of them (about 84%) then get deleted since they are identified as non-peer hosts. The amount of these hosts can be seen in Figure 3-2. The source of the peaks in these figures could not be determined but may come from network scans. The negative peak on June 22. was caused by a temporary processing delay in the NetFlow data collector.

The evaluation function examines a host's gathered data[1] and determines which - if any - is the most likely P2P network this host is belonging to. The actual code that handles identification of peers can be found in Appendix F. Due to the evaluation interval the peers get identified with some latency, which is needed to gather enough information about them. Every hosts in the pool is observed further even if it was identified. So the network an identified peer belongs to, may not remain static. This can have the following reasons:

[1] See Section 3.3.1

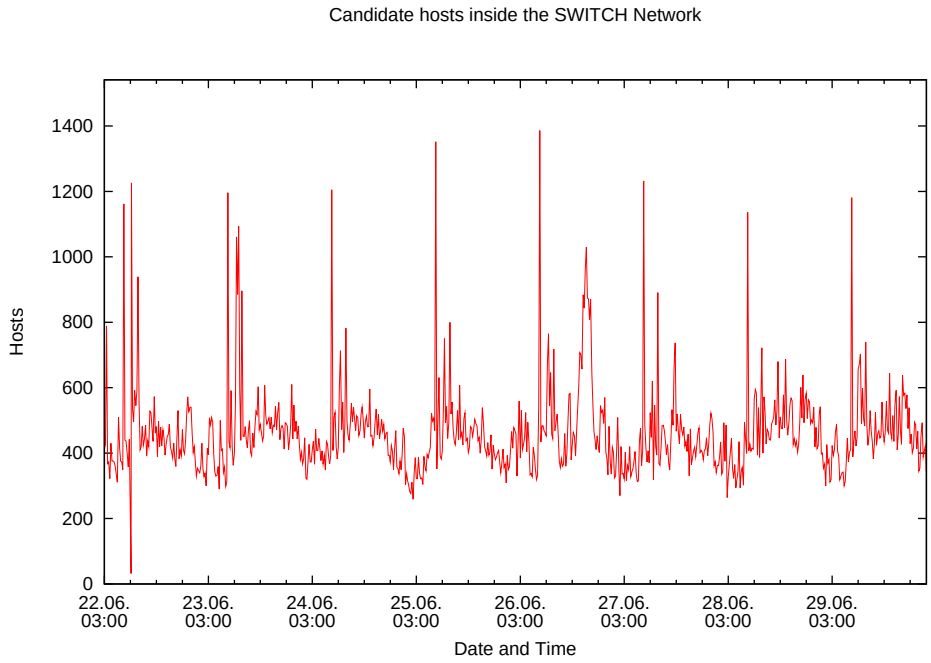


Figure 3-1: Internal SWITCH candidate peers

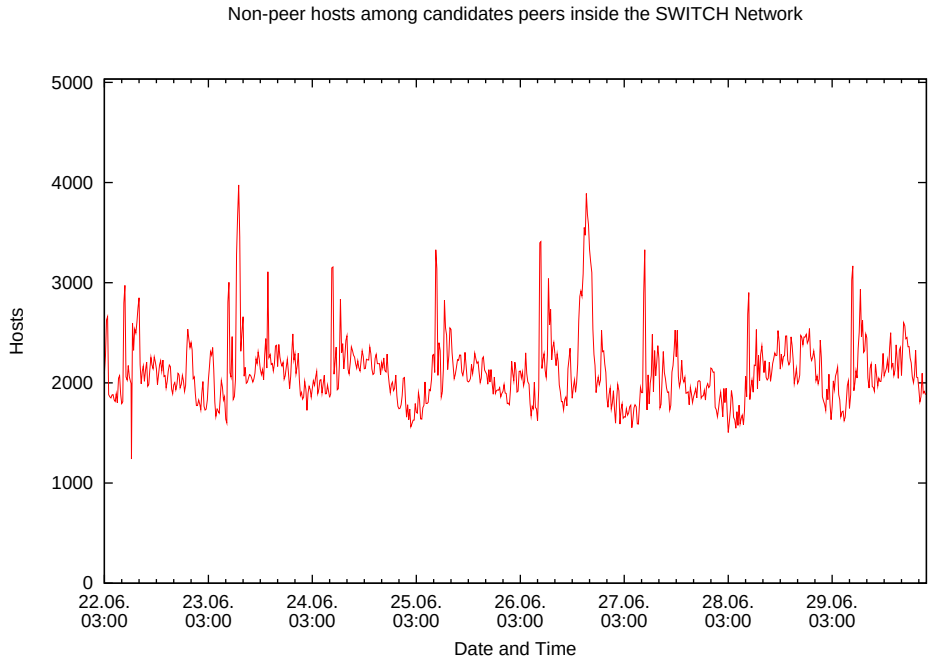


Figure 3-2: Internal SWITCH non-peers

Switches	BitTorrent	FastTrack	Gnutella	eDonkey	Overnet	Kademlia
BitTorrent	**98.42%**	0.3%	0.38%	0.25%	0.27%	0.38%
FastTrack	0.44%	**98.25%**	0.23%	0.29%	0.32%	0.47%
Gnutella	0.5%	0.22%	**97.61%**	1.08%	0.35%	0.24%
eDonkey	0.23%	0.17%	0.69%	**92.97%**	**3.73%**	**2.21%**
Overnet	0.44%	0.45%	0.33%	**6.8%**	**91.73%**	0.25%
Kademlia	0.41%	0.33%	0.18%	**1.93%**	0.15%	**97.0%**

Table 3-2: P2P network switches

- There has not been enough data gathered about a host to safely determine the corresponding P2P network. This is especially the case if an eDonkey/Overnet/Kademlia host is evaluated the first time. Afterwards the amount of gathered data stabilizes the identification result.
- More than one P2P client is running on a host. The activity of the clients varies which leads to different port distributions that influence the identification.
- The user may start an additional P2P client which is better identifiable.

Analyzing all identified peers of an eight day trace, P2P network switches shown in Table 3-2 have been observed. The table shows the network switches of consecutive evaluation intervals. If a host first is identified as eDonkey peer then after the next evaluation as Overnet peer and finally again as eDonkey peer, this counts as one switching host with two network switches. Of 3'047 identified peers, 1'244 at least once changed the P2P network according to the identification algorithm. As assumed the separation of eDonkey and the other two related P2P networks Overnet and Kademlia seems to be difficult for the algorithm. Therefore relatively much network switches occur from and to the eDonkey network.

3.2.2 Suspicious Port Range

As was described earlier, P2P clients mostly use port ranges above the well-known ports. Therefore the identification algorithm uses a suspicious ports range to track external peers, to update activity timeouts and for some statistics. If source and destination port of a flow are within that the suspicious port range the corresponding flow is flagged as suspicious.

The identification of internal clients is influenced by the suspicious port range in the way that only suspicious flows are considered as potential P2P flows. This means that only ports of such flows are stored and analyzed for MURP. In order to analyze the influence of the suspicious port range on the identification of internal clients, measurements with various ranges were made. The NetFlow data from one typical weekday was used. The suspicious port range starts at port 1024 and ends at a specific port. As can be seen in Figure 3-3 the optimal range - in terms of maximum identified hosts - is 1024-49150 what corresponds to the registered port range. Overall the difference is small for ranges above 7000 and below the dynamic port range (e.g. 1348 vs 1357 for ranges 1024-10000 vs 1024-30000).

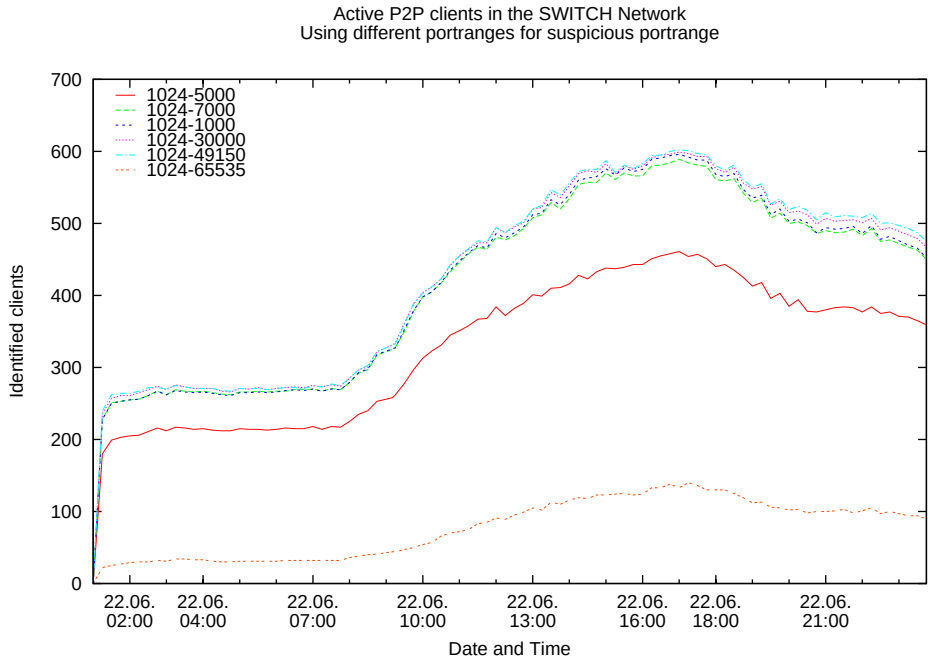


Figure 3-3: Influence of suspicious port range on identification

Range 1024-5000 is not large enough for some clients like BitTorrent which has its default ports at 6881-6889. If the range in contrast is too large (1024-65535) the identification fails because too many non-relevant ports (local ports of outgoing connections) are stored too, so that either the thresholds must be reduced or more ports most be saved. Both alternatives have disadvantages related to accuracy and memory usage.

3.2.3 Tracking of External P2P Hosts

As explained we only identify internal peers that are then used as a start population to track external peers. External hosts are considered as active P2P peers if

- they contact an internal active or dead P2P host on its TCP or UDP listening port if it was identified
- they contact an internal active or dead P2P host using source and destination port in suspicious range
- they are contacted by an internal active P2P host on its TCP or UDP listening port if it was identified
- they are contacted by an internal active or dead P2P host using source and destination port in suspicious range

External peers inherit the P2P type - the P2P network they belong to - from the internal hosts that they contact unless a P2P default port is used. For multi-client hosts, external peers are attributed the network of the best identifiable P2P network which normally is also the most active network.

3.2.4 Client Shutdown Detection

To detect if a P2P client still is running, we use the suspicious port range and the default ports together with a timeout *MaxPeerInactivity* that is refreshed under certain conditions. After the timeout is exceeded an active internal peer is declared *dead*. It then stays dead for another time *MaxAfterLife* until it changes its state to *overdead* and is deleted. The afterlife of peers can be used to track external clients that still contact shut down peers on their listening port, even hours after they stopped running.

Since external peers can't be observed that well as internal peers, there is also a single timeout for them without an afterlife. But even if they are contacted again shortly after they got deleted, they immediately get active status again.

In general the timeout is refreshed for sending hosts only if

- they use a default P2P port
- they send UDP packets on their identified listening port
- they send packets in the suspicious port range
- an external active peer contacts an internal active peer

The average inactivity time is less than 2 Minutes as is shown in Table 4-7, which basically means that the timeout is reset quite often. Measurements have shown that the effect of a timeout *MaxPeerInactivity* beyond 600s are negligible.

External peers have a different timeout value and no afterlife since they are deleted immediately when they are considered dead. See Section 4.4 for the timeout value of external peers.

3.2.5 Listening Port Detection

Beside the actual peer identification, for some peers the listening ports can be detected if a peer is active enough. The detection works hand in hand with the peer identification. Under the assumption that a peer is contacted by many neighbor peers on one local listening port, the most used local TCP and UDP ports are considered as listening ports if they exceed the threshold *MinIncomingPortThreshold* which - after some tweaking measurements - was set to 60, which means that at least 60 out of *NeighbourHosts* (100 was used for all measurements) must have used the same port to contact a host. The higher the percentage, the less identified listening ports but the more accurate are the results. Table 3-3 shows for which P2P networks the UDP and listening ports were identified, according to an eight day measurement.

Protocols that heavily rely on UDP (Overnet and Kademlia) have a better chance that their UDP listening port gets identified. The result of BitTorrent is surprising at first since it does not use UDP at all. After inspecting their listening ports (see Table 3-5) it seemed that these hosts were running not only

Protocol	Hosts	Identified UDP ports	Identified TCP ports
BitTorrent	6451	5.1%	16.4%
FastTrack	3772	36%	14.1%
Gnutella	4724	40.7%	24.9%
eDonkey	3141	40%	18.6%
Overnet	2034	85.6%	41.7%
Kademlia	2592	80.8%	67.1%

Table 3-3: Detected listening ports

1214 (13.1%)	2568 (4.6%)	32656 (0.97%)	2872 (0.11%)	1969 (0.04%)

Table 3-4: FastTrack file transfers port number

BitTorrent but also another P2P software. The ports were distributed quite random with 4672 (eMule) as most used UDP port. About a third of the identified ports were located in the 4000-5000 port range what is the preferably used range of eDonkey, Overnet and eMule. Obviously the identification of BitTorrent was easier than any of the other P2P clients running on that host.

The three most used ports of all identified hosts are depicted in Table 3-5. It is not surprising that the default ports for each P2P network are dominant. Interesting but expected though is the relatively low number of FastTrack default ports that is with 8.3% far lower than for the other peers.

In measurements made in January 2003 for [59] the five most used ports of 961'461 FastTrack file transfers are shown in Table 3-4. The default port usage has further decreased since then (13.1% vs 8.3%) although a direct comparisons between host port numbers (1 peer = 1 port) and port numbers used in file transfers (port of one peer may be in statistics several times) should be compared with some caution.

The cause of this low default port usage is the fact that recent Kazaa clients choose random port numbers for communication with other peers.

Remarkable are the non-P2P default ports that are used quite frequently as most used ports. Most of them are not registered and not even known in port databases[2]. Possible explanations for these high numbers are:

- Users which changed their IP (e.g. per DHCP) during the eight days of this measurements were counted several times and therefore have a large impact on the results.
- Some numbers like 14662, 2662, 1223 or 3333 are either quite similar to the default port numbers of the corresponding P2P protocol or they are just "beautiful" numbers, set by the users. Several users who manually changed their listening port numbers have taken the same choice.

[2]All non-P2P default ports in the Table are unregistered/unknown except, 3333 (DEC notes), 8118 (privoxy HTTP proxy), 2579 (mpfoncl), 8080 (HTTP-proxy # common HTTP proxy/second web server port), 4660 (maclmgr, OpenMosix migrates local processes), 17300 (kuang2 back door), 1584 (tn-tl-fd2), 1223 (tgp), 1469 (aal-lm, active analysis limited license manager), 2672 (nhserver), 4200 (vrml multi user system), 4240 (vrml multi user systems), 2662 (bintec-capi)

Protocol	Hosts	1. UDP	2. UDP	3. UDP
BitTorrent	6451	4672 (12.5%)	**3333 (12%)**	4307 (3.3%)
FastTrack	3772	1214 (6.6%)	8118 (3.45%)	2579 (2.86%)
Gnutella	4724	6346 (65.6%)	6348 (5.1%)	25295 (1.5%)
eDonkey	3141	6257 (8.7%)	**1584 (7.4%)**	6346 (4.1%)
Overnet	2034	12059 (7.6%)	10490 (3%)	7035 (2.5%)
Kademlia	2592	4672 (58.4%)	**16001 (8.8%)**	2672 (4.6%)
Protocol	Hosts	1. TCP	2. TCP	3. TCP
BitTorrent	6451	6881 (69.7%)	6991 (6.6%)	6901 (5.37%)
FastTrack	3772	1214 (8.3%)	8080 (5.64%)	4660 (4.14%)
Gnutella	4724	6346 (58.6%)	**17300 (10.8%)**	6348 (2.9%)
eDonkey	3141	4662 (55.56%)	17300 (7%)	14662 (6.84%)
Overnet	2034	4662 (83.9%)	1223 (3.4%)	1469 (2.7%)
Kademlia	3496	4662 (66.6)	4200 (2.63%)	2662 (1.5)

Table 3-5: Most used ports of identified listening ports

- Network scans (e.g. for the kuang back door at port 17300) which aim at finding hosts with a kuang2 back door.

3.3 Containers

Collecting information about millions of hosts needs some consideration about which and how data should be stored. Especially when searching a specific information in large memory areas of several hundred MBytes, it is important to have data structures that are optimized for space and speed.

3.3.1 Hashed Table

The implementations discussed in Chapter 3 makes heavy use of IP lookups. When storing some million IPs and their corresponding data, an efficient hash table can speed up searches enormously. Therefore the data structure *hashed_table* by Arno Wagner was used, which was released under the GPL. It is dynamically resizable and can easily be adapted to use different hashing algorithms. Since PeerTracker uses IPs as key, no hash algorithm is used but just the IPs.

To avoid frequent rehasing the table is initialized with a size of 5 million entries. Allocated memory in Linux is only "physically" used (page wise) when actually needed.

The implementation differentiates between internal host and external hosts. For every host in the host pool basically a data type of the format in Table 3-6 is used.

An extension data type shown in Table 3-7 is used for observed (intern) hosts.

Field	Description
ip	IP of the host
udpport	UDP port if available, 0 otherwise
home	1 if host is in home net, 0 otherwise
status	One of the following states: Candidate, Active, Dead, Non-peer, Overdead
type	To which P2P network this host belongs. Host can be only in one P2P network.
st	Start time, UNIX times tamp of when this host was inserted in host pool
en	End time, last time this host sent an identified P2P flow
data	For internal host; pointer to additional data container. *NULL* for external hosts

Table 3-6: Standard host container *Host*

Field	Description
tcpport	Hosts TCP port if available, 0 otherwise.
udplocal	Pointers to hashed queue data type, see Subsection 3.3.2
udpremote	“
tcplocal	“
tcpremote	“
identtime	Time needed to identify this host, highly depending on *EvaluationInterval* (see Appendix G)
totalsentbytes	Total traffic that were sent to hosts in home net
P2Pexternsentbytes	P2P traffic that were sent extern hosts
totalreceivedbytes	Total traffic received by hosts in home net
P2Preceivedbytes	P2P traffic received by extern hosts
susp	Number of suspicious flows
tcpbig	Number of TCP-big flows
[t|u][1214|3531]	Number of tcpTCP/UDP flows with hits on FastTrack/PeerEnabler default ports
t[6881-6889|6969]	Number of flows on BitTorrent default ports

Table 3-7: Host container extension *HostData*

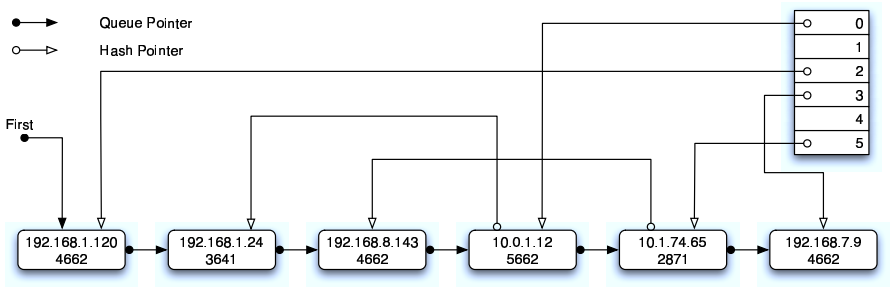


Figure 3-4: Hashed queue

3.3.2 Hashed Queue

In order to observe the port distribution of observed hosts, a special data structure that stores pairs of IP and port as elements is needed. It shall store the last N ports the host has contacted while ensuring that flows to the same host do exist only once in the queue. The following requirements for the data structure must be met:

- Dynamically resizable queue
- Searches using the IP as key of $\mathcal{O}(1)$
- Low memory usage

To fulfill these requirements an ordinary list can be used, extended by a hashing table that points on elements in the list, as it can be seen in Figure 3-4. In the implemented *ip_port_hashed_queue* each element stores the pair of IP and port, together with two pointers. One of the pointers links the next element in the queue, the other one is used if there occurs a collision in the hash table. The hash table itself is per default of the same size as the queue length, but can be adapted at run time. The hash code uses the IP as a positive integer and returns the remainder of the division with the size of the hash table. This number then is equivalent to the position in the hash table. Tests with some thousand IPs of existing hosts have shown, that the collision rate is around 45% if the hash table is of the same size as the queue. Increasing the size of the hash table while keeping the queue length constant of course reduces collisions but consumes more memory, which seems to be the more limiting factor for the plug-in.

The data structure has an overhead of 64 Bytes with additional 14 Bytes per actually **used** element and another 4 Bytes times the hash table size what gives a total of 2064 Bytes for a hashed queue with 100 elements as it is used per default in PeerTracker.

From the various functions that operate on *ip_port_hashed_queue* the most important are a *searchHost* function which tests if an IP already exists in the queue, an *appendHost* function which inserts a new element at the end of the queue (pops the first element if needed) and a *getChart* function which returns the n most used ports in the queue, using binary sort algorithms ($\mathcal{O}(n*log(n))$).

Moreover there are functions to resize the queue/hash table, to look for a certain port ($\mathcal{O}(n)$, n=100, used once for every host every time it is evaluated) and to output the content of the queue.

3.4 Resource usage

3.4.1 Space Complexity

PeerTracker was designed with fast processing speed and low memory usage in mind. The amount of consumed memory is linearly depending on the number of hosts stored since memory usage for every host is constant. An external host consumes 20 Bytes of data and another 20 Bytes overhead. An internal host has an overhead of 5'860 Bytes to store 2518 Bytes of data at most. This large overhead is needed for the hash table management. internal hosts have a total of four hashed queues attached. The number of internal and external hosts is influenced by the networks which are observed and by the timeouts set to detect dead peers. See Appendix G to discover how to configure these parameters.

If the command line option *'-d'* for detailed traffic statistics is enabled, memory usage is increased further depending on the number of active (sending) internal and external hosts observed. Storing 16 million hosts with maximal 2500 internal hosts among them uses about 1.2 GB in RAM with all of PeerTracker's traffic statistics turned on. In the SWITCH network, PeerTracker consumes about 330MB if there are 1.5 million hosts in the pool. These numbers are typical for SWITCH network if all P2P networks are tracked with detailed statistics and a timeout for external peers of 6 hours.

To limit PeerTracker's CPU and memory usage the Linux bash function *ulimit* [3]can be used. E.g. *'ulimit -v 5000000'* in PeerTracker's startup script would limit the virtual memory size that it can use to 5'000'000 KB. PeerTracker needs at minimum 310MB of virtual memory to start. This is because the host pool hash table is initialized with 5 million entries to prevent costly rehashing operations.

If the *'-m'* switch is enabled and less than 5MB of memory can be allocated, PeerTracker stops allocating more memory. This means that the statistical data gets unreliable until enough memory is available again which normally is the case after the next time the hosts are evaluated. Without this memory check PeerTracker would exit automatically but could be restarted by the watch dog script if one is available. Of course this would mean that every known host is lost and population tracking has to start anew.

PeerTracker will output for every statistics file if the data it contains is valid or not, depending on packet loss and memory availability.

3.4.2 Time Complexity

Processing of NetFlow data is independent of the number of hosts in the host pool since efficient hash tables are used. Although table collisions increase with the number of hosts contained in the hash table lookup, insertion and deletion of hosts are of $\mathcal{O}(1)$. As was described in Subsection 3.3.2, every time the hosts are evaluated a chart with the most used ports has to be calculated which has

[3]See *"ulimit man page" (September 2004)* http://www.ss64.com/bash/ulimit.html

complexity $\mathcal{O}(h * n * log(n))$ where h is the number of internal hosts and n the amount hosts that are stored for MURP. n is a constant set to 100 in the default configuration[4].

On an AMD Athlon XP 2800+ with a 2GHz CPU clock it takes 13.5 minutes to process 100 MFlows, 10 minutes on an Athlon 2.1GHz dual CPU (detailed traffic statistics and memory check turned off). In the SWITCH network about 45MFlows are emitted per hour on average, which makes PeerTracker in this case about 9 to 14 times faster than real-time. Computing 8 days of NetFlow data took about 14 hours on the dual CPU Athlon. This is more than fast enough to compute NetFlow data in real-time in the case of the SWITCH network. It is to annotate that *netflow_sreplay* consumes about 50% more CPU time than PeerTracker if it has to uncompress the NetFlow data.

In plug-in mode PeerTracker's processing speed is dependent on UPFrame's packet receiving rate. Processing speed should be the same as in stand-alone mode. If UPFrame received packets faster than PeerTracker or UPFrame can process them, this results in packet loss. PeerTracker will note packet loss in the log file as well as in the P2P statistics files. Packet loss can only occur in the plug-in mode since processing speed in stand-alone mode is automatically adapted by the named pipe and in general as fast as possible.

[4]See PeerTracker configuration file in Appendix G

Chapter 4

Findings

This Chapter first will show that the verification of identified peers is rather difficult but that at the same time the measurement results gathered with PeerTracker are reasonable for most P2P networks. Afterwards additional statistics and numbers are given, concerning the P2P usage in the SWITCH network and the population tracking. Finally, approaches and discoveries made in other research papers are presented in the section on related work.

4.1 Peer Verification Approaches

An accurate way to verify the identification results concerning false negatives and false positives would be packet inspection of all packets passing the border gateway routers of a network. Unfortunately this is hardly possible for high bandwidth links with almost GBit transmission rates since processing the packets or even only parts of them is too time consuming. An alternative would be to capture and store the data for an offline analysis. But as is shown in [40], capturing the packets for offline analysis is a difficult task too. So this hasn't been an option either.

Without packet inspection one could use polling techniques. In that case a detailed understanding of the P2P protocol is required but sometimes is hard to achieve since except Gnutella and BitTorrent, all protocols are proprietary and only reverse-engineered publications are available to the public. The success of polling is very protocol dependent. Furthermore polling can only detect false positives. Finding false negatives is very hard since this would require to poll all hosts of a network on every port for every P2P protocol. Having at least a large pool of suspicious hosts and used ports helps decreasing the number of hosts to poll. In Subsection 4.1.2 a polling technique is used to find a lower bound of the identification degree for Gnutella, FastTrack, eDonkey, Overnet and Kademlia.

A third way of verifying peers is to run test clients that are known to be active at a certain time and then determine wether the tracking algorithm identifies them. This approach is used for BitTorrent in Subsection 4.1.3.

Verification is done for SWITCH internal peers only since they are of larger importance for the P2P traffic identification than the tracked external peers.

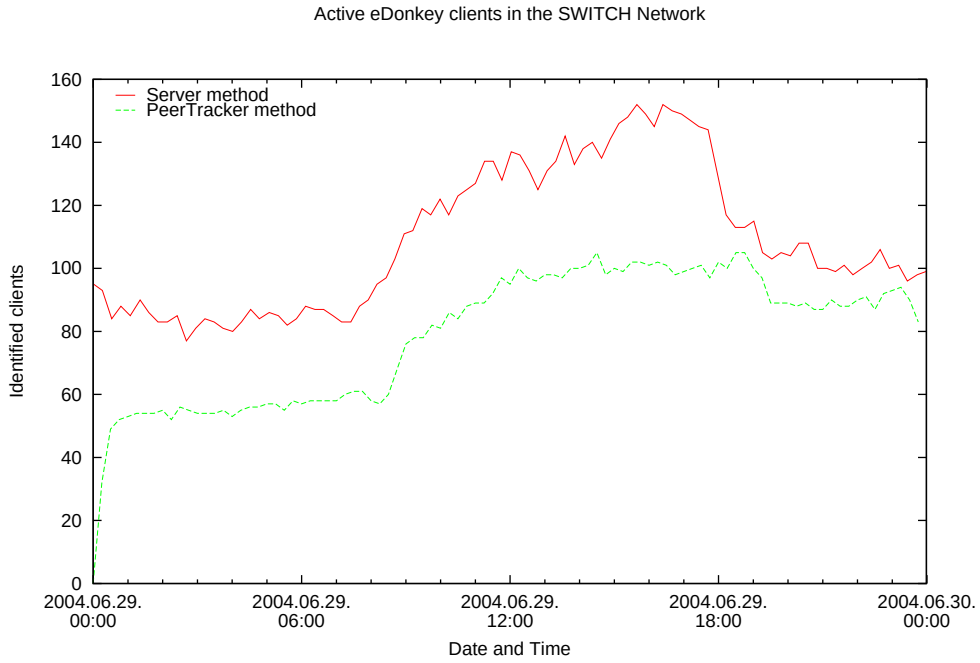


Figure 4-1: One day eDonkey comparison server method vs port method

Server tracking method	eDonkey only	+Overn. + Kad.	all peers
481 hosts (reference value)	378 hosts	831 hosts	1357
Unique hosts	571 hosts	652 hosts	1405
Matching hosts	288 hosts (60%)	421 (88%)	433 (90%)

Table 4-1: Identified eDonkey clients for one day

4.1.1 eDonkey Server Tracking

Since there exist only about 80 static eDonkey servers worldwide and since every eDonkey client needs a connection to one of them in order to join the eDonkey network, it is possible to look for hosts connecting to these servers. The chances are high that hosts which connect with the same IP and port to an eDonkey server are running a corresponding client. The results of an ordinary weekday were examined. Using the eDonkey server lists from [53] which contains all active and populated servers worldwide (updated every six minutes) all hosts of the SWITCH network connecting to one of these servers were searched. At least one flow was necessary to be considered as eDonkey peer at first. Comparing the number of identified eDonkey clients between the server tracking method and the peers identified by PeerTracker gives Figure 4-1, which shows a substantial difference in identified hosts. PeerTracker doesn't seem to find all the eDonkey hosts.

If furthermore the IPs of identified peers are compared between the two

methods, PeerTracker seems to find only 50% of the eDonkey hosts. If also the IPs of identified Kademlia and Overnet clients were used for comparison about 72% of the hosts matched between the two methods. Adding also the identified FastTrack, Gnutella and BitTorrent IPs for the comparison finally gave a match of 75%. So PeerTracker has found 75% of the peers which were found with the server tracking method. The reason why the number of matched IPs increases when the IPs of the other P2P networks are considered too, is that eDonkey clients use the same download port as the Kademlia (eMule) clients and the Overnet clients. Therefore the classification between these three clients is not always accurate[1]. Moreover the official eDonkey client is a hybrid client that also joins the Overnet network. Since PeerTracker only identifies the most obvious P2P client for a host, some hosts running eDonkey and another clients are not counted as eDonkey hosts.

Analyzing the missing 25% hosts showed that most of them were not identified by PeerTracker because they were running only a short time. Therefore the requirements to be considered as eDonkey peer were increased for the server tracking method. Since eDonkey is contacted by its server, which it doesn't change after start up, at least every 412 seconds and since the average uptime of peers is far more than half an hour the minimum time a peer must have been connected to a server was set to 1'000 seconds. That way peers which unsuccessfully connected to a server were not counted any more. Table 4-1 shows the results of this measurement. PeerTracker found 90% of the eDonkey peers after the requirements were increased. The missing 10% can be a result of peers that have no uploads or downloads in progress. Therefore they mainly communicate with their eDonkey server and have hardly contact with other peers. PeerTracker in such cases has difficulties to detect peers since it benefits from active peers. Concerning the bandwidth estimation those 10% are not of concern since they don't contribute much to the P2P traffic without file transfers.

4.1.2 Polling Peers

To further determine the identification accuracy of the considered P2P networks, a Perl script was written whose goal is to poll and verify identified peers. Since polling is an active process that needs the identified peers still to be online when it is contacted, an almost real-time identification is needed. Unfortunately DDoSVax had - at the time this thesis was written - no practicable possibility to process the NetFlow data instantly after it was emitted by the border gateway routers. Instead the NetFlow data is collected and then after some hours transfered to the DDoSVax NetFlow archive. A latency of several hours would severely impact the polling results. Therefore a script was written that can be used to download the latest NetFlow data from the SWITCH collector host. To keep latency short, only data of the first 20 minutes after a full hour was downloaded. In total we have a latency of about 20 minutes between the first flow of the downloaded data and the time the identification has finished. Polling 350 hosts using 20 parallel threads takes then another 30 minutes (due to the number of polling attempts and a generous timeout). It is probable that some identified peers shut down during that time.

The polling itself is partially based on the signatures used in [9] to identify

[1] See also Table 3-2

peers using packet inspection techniques. Every identified active peer is polled first on its three most used local TCP ports and then on all P2P default ports of the considered networks. A timeout of 7 seconds was set after which unsuccessful connections were cancelled. Four measurements were done on week days (2./3./7./14. Sept.) at 4 or 5 pm when the number of P2P clients normally is high.

In the following the different queries and signatures used to poll the potential peers are described.

Gnutella

Unlike most P2P protocols, the Gnutella protocol [64] is freely available to the public. A TCP connection to a potential Gnutella host is set up and the standard query string *'GNUTELLA CONNECT/0.6'* then is sent. It should be answered by a real Gnutella host with *'GNUTELLA'* as the first word. Beside the three most used local ports the ports 6346, 6348 and 6349 were polled.

FastTrack

The FastTrack protocol is encrypted and not publicly available. Some reverse engineering attempts [65] were done, e.g. for the open source mlDonkey of giFT clients. But the published information is far from complete. Nevertheless the FastTrack downloads are not encrypted and a download request for FastTrack peers starts with the HTTP command *'GET /.files HTTP/1.1'* that is sent using TCP. A FastTrack client then will reply with a pseudo HTTP response of the form *'HTTP/1.0 403 Forbidden'* followed by two decimal numbers that stand for the encryption type and a challenge number. Some Kazaa clients also answer with *'HTTP.1.0 404 Not Found\nX-Kazaa-Username'*. Some Kazaa clients listen and answer on several ports including the ports 1214 and port 80 even if their mainly used listening port was chosen random. Ports 1214 and port 80 are polled in addition to the three most used local ports.

BitTorrent

The BitTorrent protocol [66], though openly available, includes no possibility to directly identify a BitTorrent peer. The reason for this is that a contacting peer first has to start a hand-shake which includes the hash value of the torrent file. Since we don't know this value the polled peer won't answer. But one thing noted is that BitTorrent peers immediately close the connection if the torrent hash value differs from their own. If the hand shake is not complete yet or not valid the connections is held open until a time out value is exceeded. Another property of BitTorrent peers that can be used for polling is that they - depending on the client and the number of current downloads - normally have several (usually 2) listening ports that will accept connections.

Therefore the a technique that could be used to poll BitTorrent clients, could send it a random hash value on all BitTorrent default ports. If the connection is accepted and closed immediately after sending the query this would be a good sign but is no proof since it is a recommended action for protocols using TCP to close a connection upon an invalid message. Therefore peers were not checked for BitTorrent activity.

Network	Gnut.	FastT.	eDon.	Overnet	Kad.
Identified hosts	98	83	186	140	256
Responded to ICMP pings	37	29	30	43	100
Responded to TCP pings	35	21	55	48	173
Verified	24	16	42	25	166
Verification ratio	65%	55%	76%	52%	96%
Additional clients verified	2 eDon.	0	3 Gnut.	0	3 Gnut.

Table 4-2: Polling verification results

eDonkey/Overnet/Kademlia

Although there is no official version of the eDonkey protocol available there exist some usable reverse engineered protocol specifications[56, 58]. Overnet and Kademlia (since included in the eDonkey client eMule) use many of the original eDonkey protocol messages. One of them is the *eDonkey Hello* message that is used by peers to announce their presence before a file transfer is started. The eDonkey protocol is binary and therefore a bit more complicated to understand. The polling algorithm basically sends a potential peer a hello message which then is supposed to be answered by a real peer. If the first byte of the answer is the eDonkey protocol marker hex value *'e3'* or *'c5'* (eMule extension) the host is verified as an eDonkey, Overnet or Kademlia peer according to the assumption that was made when this peer was identified.

Ports 4662, 4672 and 5662 are polled.

Verification results

One of the main difficulty with polling techniques is that hosts behind firewalls, NAT devices or proxies can't be contacted normally. As was described in Section 2.3.1 this is also a problem that P2P protocols have to deal with. Firewalls and intrusion detection systems or P2P client internal security measures can make polling even more difficult. After some unsuccessful connection attempts a contacting host is seen as attacker and put on an *ignore* list. All further connections from that host are then blocked. About 60% of all Limewire users are believed to be behind firewalls or NAT devices[2] . So the results shown in Table 4-2 are not that surprising.

More than half of the hosts didn't even respond to ICMP pings or a TCP ping which was done on more than 12 different ports. Since - for latency reasons - only about 15 minutes of data are evaluated, the listening port may not be identified for most hosts so that polling does not work unless the host listens on one of the polled default ports.

The verification ratio is calculated as the number of verified peers divided by the maximum number of hosts which could be pinged (ICMP or TCP). As can be seen the results are quite varying and not very promising. Especially FastTrack clients are not only harder to identify but also more difficult to verify than peers of other networks. A reason why the number of verified hosts is rather small for

[2]See article "Gnutella on the Rise" http://www.slyck.com/news.php?story=530 (August 2004)

some networks is that for polling the listening port of a peer has to be known. Unfortunately the listening port is not detected[3] for all the peers and it usually takes several hours or a lot of file transfers for a detection. Therefore, if a peer does not use the default listening port - as is the case for most FastTrack clients - and if it can't be detected otherwise, it is almost impossible to poll this host without polling all of its ports. It was tried to poll a host also on its three most used local ports though, but local ports are not equivalent to listening ports in general since local ports could be the ports used for outgoing connections.

For a small fraction of peers there were additional clients validated, mostly Gnutella and eDonkey. Clients like mlDonkey include support for both of these networks. It is believed that the amount of hosts which are part of several P2P networks actually is higher than these numbers make believe. E.g. the polling method used can't separate eDonkey, Overnet and Kademlia peers from each other.

One has to keep in mind that the results of Table 4-2 should be considered as a lower bound for the identification accuracy when using polling as verification method.

4.1.3 BitTorrent probe clients

Since the polling method couldn't be used for BitTorrent and since downloading with BitTorrent can be done easily with one single command, 18 test clients were set up to run BitTorrent. Two clients were connected over a 1MBit cable connection in the ETH-Cablecom IP range the others were directly connected to the Internet with a 100MBit connection. To make things a bit harder all clients were configured to use a non-default listening port in the range of 3254 and 3289. 8 pairs of clients were downloading the same files. The downloaded files were at least 200MB in size and at least 10 people were downloading or uploading them. The clients were shut down after one hour.

All of the test clients were detected by PeerTracker and became candidate peers. 16 of them were recognized as BitTorrent clients immediately after less than 15 minutes, 2 clients downloading the same file needed half an hour to be recognized and a single client stayed candidate the whole time. The client which was not identified had contact with only 12 other peers which was too less for the identification algorithm to work.

4.2 Interpretation of Verification Results

As the previous Sections has demonstrated peer verification is a difficult task without the availability of packet inspection. The method used in Subsection 4.1.1 promises high accuracy and can be done passively in an offline mode. It was used for the eDonkey network since the few eDonkey server are dedicated servers which don't change frequently. For other P2P systems like Gnutella or FastTrack which have a two-tier architecture too, the same principle could be used. But since their super nodes change more frequently and are far more numerous, the effort to track them all is rather high as has been demonstrated for FastTrack in [27]. Developing a crawler program for Gnutella seems more feasible though as is demonstrated on [60]. The numbers for eDonkey look not

[3] Also see Table 3-3

that bad with 90% verified peers which is higher than the result achieved with the polling method.

In general polling doesn't seem to be an option to verify the complete identification results. Since only a small amount of peers at all accept incoming connections due to firewalls and NAT devices, it is only a fraction of peers that can be validated. But the ability to connect to a host is needed to poll them. Furthermore the listening port of a potential peer must be known. For most protocols the default port can be used in most cases. Polling is an active an intruding technique which may be interpreted as an attack by some hosts which causes them to ignore the polling hosts.

The number of verified peers with the polling method are low for some of the considered protocols. They may serve as a lower bound but it is believed that the amount of correctly identified peers actually is higher which also is confirmed in the case of eDonkey. Surprising are the differences between eDonkey, Overnet and Kademlia. They all were polled using the same method which consisted of an eDonkey hello message. But obviously eMule peers (Kademlia) first of all could be contacted more often than the two others. Additionally the verification ratio of Kademlia was found to be substantially higher than for the other two related networks.

Running test clients is a safe way to detect false positives and to test the identification algorithms. One though has to consider that the test environment can have a great impact on the results. Overall the results for BitTorrent look promising with 94% correctly identified true positives and only 6% false negative clients.

4.3 P2P Usage in SWITCH Network

In this section some facts and statistics about P2P usage in the SWITCH network are presented. Keep in mind that the SWITCH network is not comparable to an ISP network which provides its services mainly to home users whose Internet usage is different as in an educational network.

4.3.1 Measurement Setup

All measurements published in this thesis were done for the eight days from 22nd to 29th of June 2004 or a 30 day period from 7th of August to 6th of September. The week in June is one of the last weeks of summer term[4] for all large Swiss universities (ETH, EPFL, Universities of Basel, Zurich, Lucerne, St. Gallen and others) that produce most Internet traffic in the SWITCH network. If only one day of analyzed data was used it always was a Tuesday. Processing with PeerTracker was done in the stand alone mode since this mode allows to analyze the NetFlow data as fast as possible and moreover does not produce any flow loss as may be the case in online mode. For all measurements the NetFlow data of all four border gateway routers was processed. All measurement graphics were created with a resolution of 15 minutes. All date and times correspond to local time in Switzerland (CEST, UTC + 2 hours). If not otherwise mentioned the PeerTracker configuration shown in Appendix G was used.

[4] See academic calendar of all Swiss universities Academic Calendar http://www.crus.ch/mehrspr/iud/semester.html (August 2004)

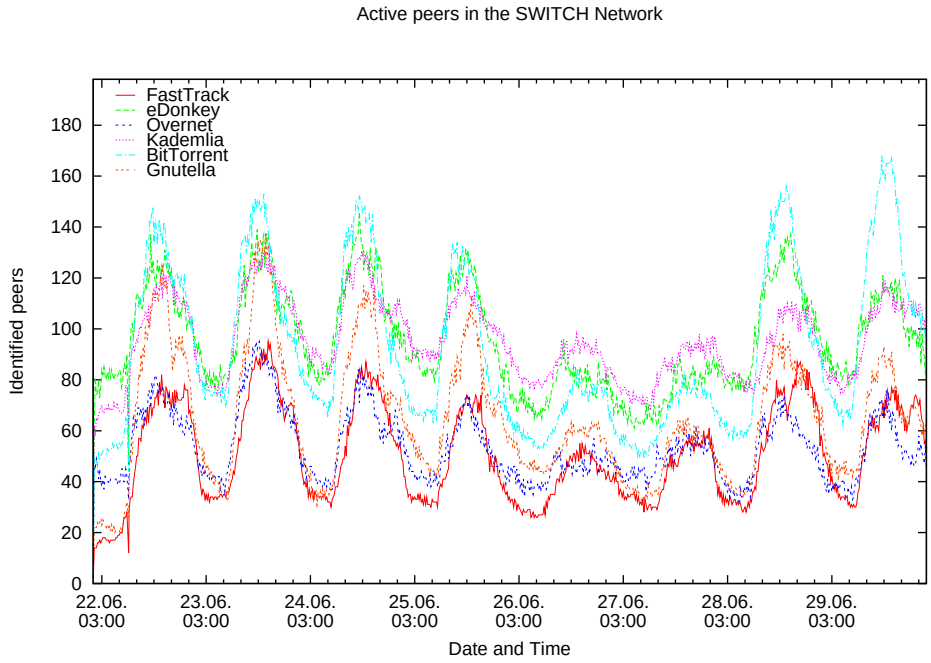


Figure 4-2: Active peers within SWITCH network

4.3.2 Peers in SWITCH network

Figure 4-2 shows the identified peers during an 8 day run at the end of summer term in Switzerland. It can be clearly seen that the weekend (June 27./28.) has a great influence on the P2P usage as it has on web usage.

Comparing the P2P usage with web surfing usage as shown in Figure 4-3 shows that there are on average 7.5 times more people surfing the web than there are P2P users. P2P usage seems less day time dependent. Every night there seem to be at least 300 people who let their P2P clients run while there are at least 620 hosts that surf the web.

That "negative" peak in WWW usage that is characteristic for every weekday is probably due to launch since it exactly is in the time range of 11.30 am and 1 pm. People leave their desks at 11.30 am to take lunch. After 1 pm, WWW usage starts to increase again.

P2P usage seems to be highest around 5 pm during weekdays and about an hour sooner and more constant at weekends. In general the different networks seem to have different users as was described in Subsection 1.5.3.

Overall it is surprising that so few FastTrack clients were found although it is supposed to be the largest network worldwide in terms of active users. This can have several reasons:

- Identification for FastTrack peers is more difficult since Kazaa, the most used client, in recent versions chooses random port numbers. Manual examination of the non-peers has shown though that there is no substantial amount of FastTrack suspicious hosts.

	Average hosts	Standard deviation
BitTorrent	92.3	**31.61**
FastTrack	50.8	18.7
Gnutella	64.5	**24.8**
eDonkey	95.6	19.9
Overnet	53.4	**13.4**
Kademlia	95.3	15.6

Table 4-3: P2P average usage per network in June

- Kazaa - in the free version - contains a lot of ad-ware and spy-ware. Most users in the SWITCH network are in general technically sophisticated and may have heard from that. Therefore they may avoid Kazaa, especially when network administrators explicitly dissuade from using it.
- Most of the sued P2P users in the USA and in Germany were FastTrack users. This had a great influence on the usage in companies and universities where the system administrators explicitly prohibited the usage of FastTrack clients.

Noteworthy are also the numbers of Table 4-3 which show a significant variation in P2P usage. According to the standard deviation shown there, some networks have a more constant user base in the SWITCH network than others. Reasons for this are probably on one hand the kind of content served by the networks, the technical knowledge that is needed to operate the clients and the download speed. BitTorrent seems to be quite popular, probably because of its ease of use and its fast download rates. Overnet in contrast is not very popular but seems to have the most constant user base. Limewire is good for downloading music files which usually are small in size. Therefore they don't take very long to download so that the client faster can be shut down by its user.

When the measurements are compared to a 30 day measurement of August the number of P2P users seems to be lowered by 14% while the web surfing people are 15% less. The P2P usage of both measurements shows the same daily and weekly characteristics as in Figure 4-2.

4.3.3 Peer Characteristics

Analyzing the 8 day measurement of June resulted in a total of 2'982 unique peers within the Switch network. Interested in the organizational location of the peers, a reverse domain name lookup was done for each of them. For 566 peers no domain name was found. But examining the domain names of the remaining 2'416 peers showed that more than half of them seem to be in a dynamic IP range as can be seen in Table 4-5. But these numbers should be taken with caution because of the incomplete DNS lookup and since the same peer may get counted several times if its IP changes, e.g. after the user turned off its notebook while connected to a DHCP network.

Since most of the regular university and research institute employees have workstations with static IPs this huge amount of dynamic IPs may come from students or private users who are either temporarily connected to the university

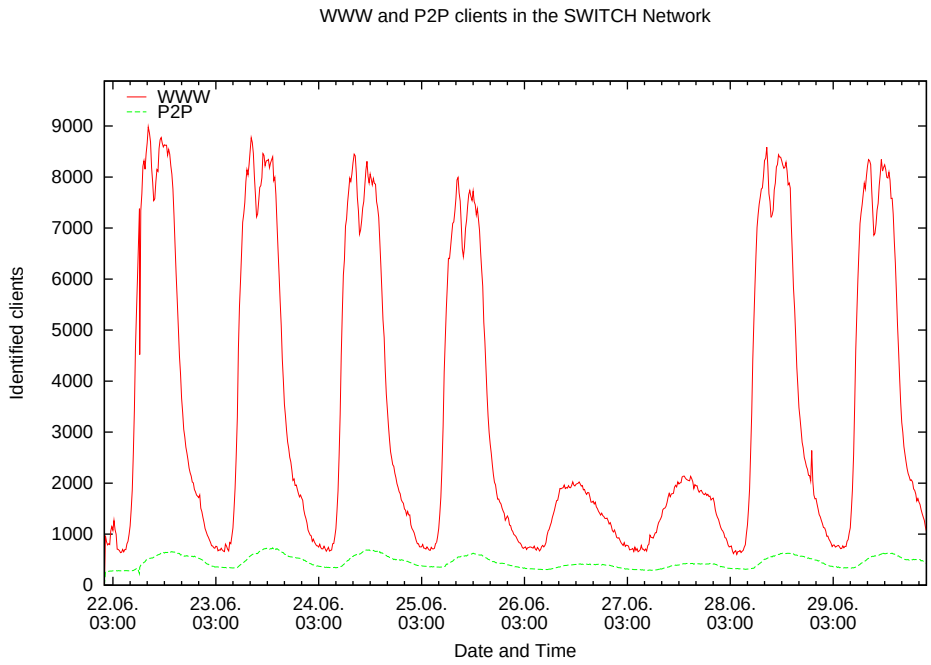


Figure 4-3: P2P clients vs web clients comparison

Description	Occurrence (2'416 hosts)	Percentage
In VPN IP range	97	4%
In DHCP IP range	710	29.4%
In DHCP-VPN range	702	29.1%
No number in DNS name (static IPs)	249	10.3%
Proxy in DNS name	9	0.3%
Remaining (static IPS, lab machines)	649	26.9%

Table 4-4: Peer domain name analysis

Description	Occurrence (2'416 hosts)	Percentage
ethz.ch	1029	42.6%
via-eth.ch	622	25.73%
epfl.ch	251	10.4%
unil.ch	99	4.1%
zhwin.ch	98	4.1%

Table 4-5: Top 5 second level domain names

network using their laptop or a dial-up connection. This assumption is confirmed when a chart with the most used second level domains is made. The corresponding numbers are shown in Table 4-5. According to this result more than 25% of all peers in the SWITCH network are subscribing the ETH-Cablecom service which provides broadband access for ETH students and employees. The ETH network seems to be home for almost 70% of SWITCH's P2P users. Among them a quite substantial amount of users which are connected over the Swiss ISP "Cablecom" but located in the ETH IP range ("via-eth.ch"). Looking at the other second level domains it can be said in general that the more technically related an organization within SWITCH is and the more students it has, the higher is the P2P usage.

A SWITCH peer statistic can be seen in Table 4-7. As was described in Subsection 4.3.2 the number of peers is dependent on the day time and the holiday seasons, so is the amount of transmitted data.

Since the *EvaluationPeriod* was set to 900 seconds for all measurements the identification time is closely related to this value. But it can bee seen that FastTrack clients in general need more more time to be identified.

Concerning the uptime it seems that eDonkey, Overnet and Kademlia peers are running longer than the peers of the three other P2P systems. This is confirmed by the numbers shown in Table 4-3 which show that eDonkey et al. have a more constant user base with fewer daily variations. Table 4-6 shows some absolute numbers about the uptime statistics. A few P2P users let their client run for several days or even weeks. Therefore these uptime and other values have to be taken with caution since they may be different in measurements that cover more than 30 days.

The average inactivity time observed is less than two minutes which would allow to further reduce parameter *MaxPeerInactivity* which is responsible to detect dead peers.

Although BitTorrent is responsible for most of the P2P traffic, its clients transfer less data than Overnet since there are almost twice as many BitTorrent clients online on average than Overnet clients. Interesting are the numbers for FastTrack. While all other peers on average send more data in total than they receive this is not the case for FastTrack peers. One explanation for this may be that some users stopped sharing files and just are "leeching" (only downloading) due to the legal threats FastTrack users are faced with.

Surprising is also the amount of well-known data sent. It was assumed that most users while running P2P clients produce traffic in the well-known range (namely WWW, SMTP), but that the received traffic is much greater than the sent traffic. As it seems the well known data sent is of substantial size. Checking

Uptime greater than	5 days	10 days	15 days	20 days	25 days
Peers	275	160	90	50	28

Table 4-6: Peer uptime

Avg./Std. deviation	BitTorrent	FastTrack	Gnutella
Peers analyzed	1'210	963	1224
Identification time [s]	942/261	1049/335	939/210
Uptime [h]	17.6/61	11.6/43	12/57.1
Inactive time of active peers [min]	0.9/2.9	0.95/3.45	1.4/3.6
Total data received [MB]	2'686/12'477	841/3'136	1'084/6'863
Well-known data received [MB]	239/2'091	69/738	233/1'500
P2P data received [MB]	1'444/7145	89/420	447/3'044
Unknown data received [MB]	1'003/6'393	683/2'500	514/4'021
Total data sent [MB]	4'373/24'618	776/5'260	1'718/18'861
Well known data sent [MB]	95/908	10/79	54/803
P2P data sent [MB]	2'582/15'695	113/724	644/7'737
Unknown data sent [MB]	1'696/10'299	653/5'040	1'020/13'960

Avg./Std. deviation	eDonkey	Overnet	Kademlia
Peers analyzed	954	633	801
Identification time [s]	942/253	941/227	922/222
Uptime [h]	24.7/79	24/90.1	34.4/92.2
Inactive time of active peers [min]	1.4/4	1.1/3.3	1.4/5.1
Total data received [MB]	1'192/4'518	2'205/16'640	3'004/14'847
Well-known data received [MB]	187/1'006	292/3'148	190/1'563
P2P data received [MB]	609/2'937	971/6'841	1'638/9'195
Unknown data received [MB]	396/'1821	942/8'173	1'176/8'793
Total data sent [MB]	1'691/8'329	3'753/44'378	4'785/26'154
Well known data sent [MB]	92/518	147/1'415	116/588
P2P data sent [MB]	986/5534	2'076/21'835	2'568/12'299
Unknown data sent [MB]	613/3512	1'530/22'163	2'101/17'139

Table 4-7: SWITCH Peer statistics

Traffic	June	August
Total avg. [MBit/s]	690	522
TCP	96.6%	97.5%
UDP	2.5%	1.8%
Other	0.9%	0.7%
Web	13.1%	14.3%
Mail	1.6%	1.12
P2P (TCP+UDP)	36%	28%

Table 4-8: Total traffic distribution

the IPs of the 2'982 identified hosts from that measurement showed that 497 of them had a web server running which may be the cause for this balanced well-known traffic. Probably there are also other well-known services running on these hosts so that the traffic in the well-known range makes sense.

4.3.4 Bandwidth Consumption

To get an idea of the P2P bandwidth consumption in comparison of the total traffic in Figure 4-5 the TCP incoming and outgoing traffic is shown. As can be seen there is more data sent out of the SWITCH network than comes in. Figure 4-6 shows the web, mail and P2P traffic which account for most of the TCP traffic. Interesting to note here is that the outgoing traffic is higher than the incoming traffic not only for WWW but also for P2P traffic. So the SWITCH network is - as most university networks - more content provider than content-requester[5]

In Table 4-8 a general overview about the SWITCH traffic of the two measurements can be seen. P2P in/out traffic ratio is 1:1.6 which is about the same as for WWW traffic. While web traffic often exceeds P2P traffic during work time (8 am - 5 pm), P2P traffic is more constant and not so much day time dependent what results in an overall higher bandwidth consumption than web traffic as can be seen in Figure 4-4.

Figure 4-7 shows the separated traffic of the considered P2P networks. As can be seen quite clearly, BitTorrent is responsible for a large amount of transferred data. Although the trio of eDonkey, Overnet and Kademlia has more than twice the users in the SWITCH network, they produce less traffic than BitTorrent.

Comparing the total amount of P2P traffic between the measurements of June and August showed that P2P traffic was 19% lower in August. As was explained above, in August there are holidays for most Swiss universities. Therefore less students and also less employees are using the SWITCH network.

Knowing that stateless port based traffic accounting is less accurate than a

[5]One side note concerning this: Some universities like ETH limit P2P traffic. ETH e.g. shapes P2P traffic to 5MBit/s using port numbers. This is not effective if they want to save upstream bandwidth. P2P users will change their ports to non-default ports. But this won't improve their download rates since the traffic shaping still will effect downloads from peers that use default ports. But the opposite way, extern peers downloading from ETH peers, will be unlimited since no filtered port is used.

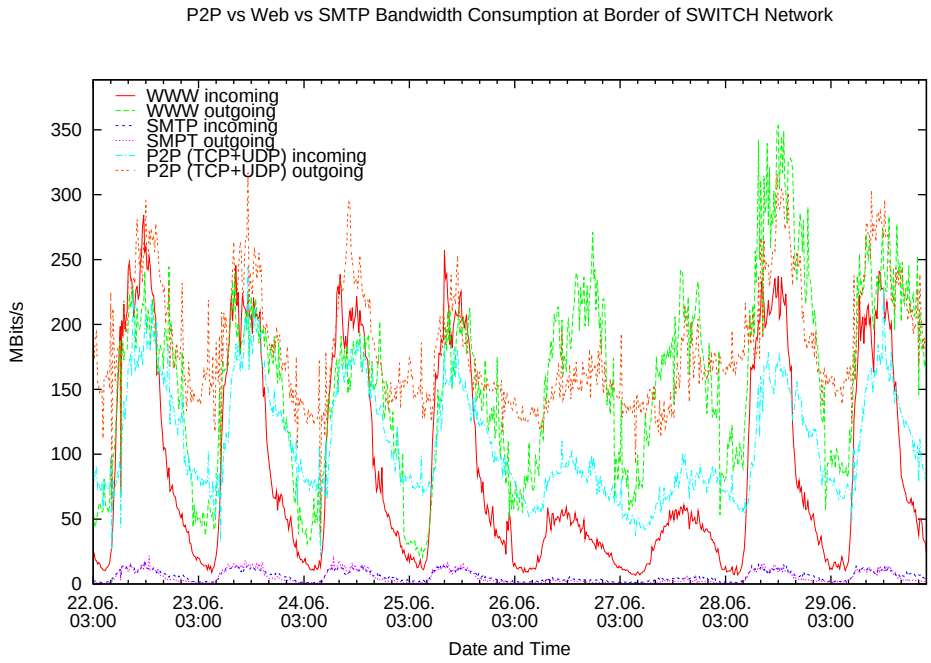


Figure 4-4: Comparison Web vs P2P vs Mail

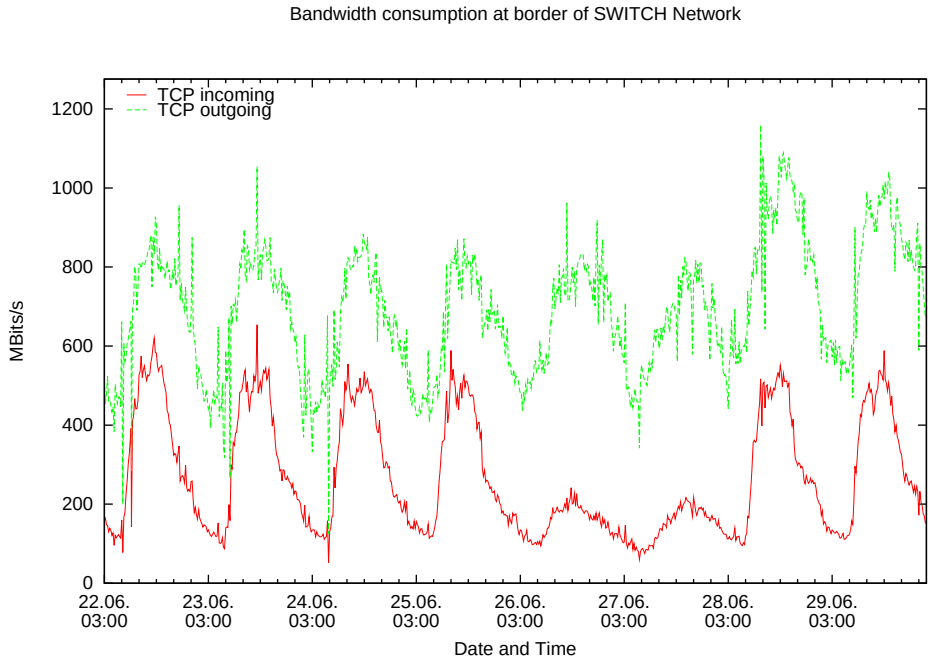


Figure 4-5: Total TCP bandwidth consumption

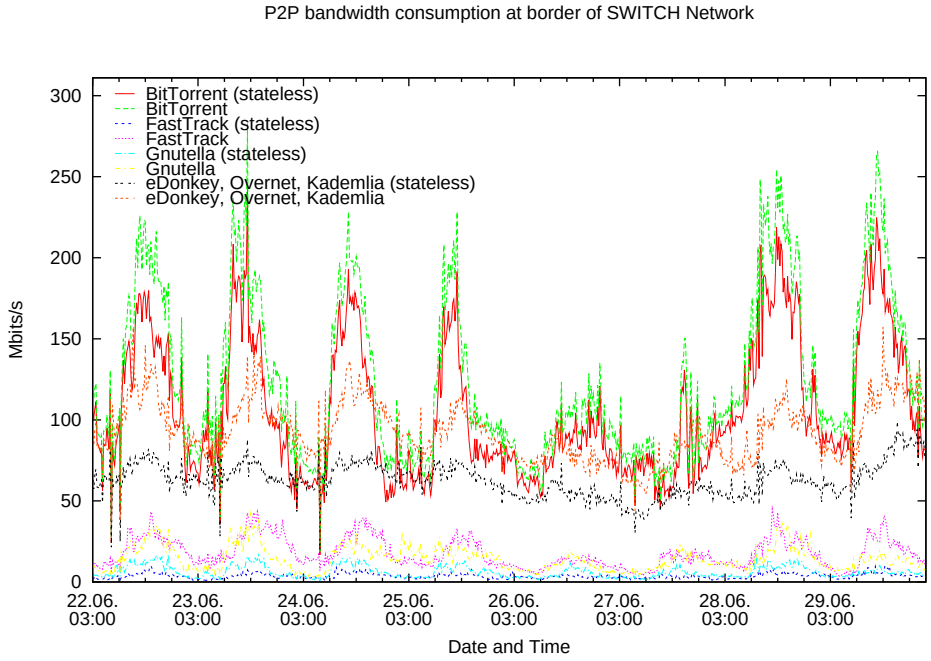


Figure 4-6: P2P traffic in comparison to other protocols

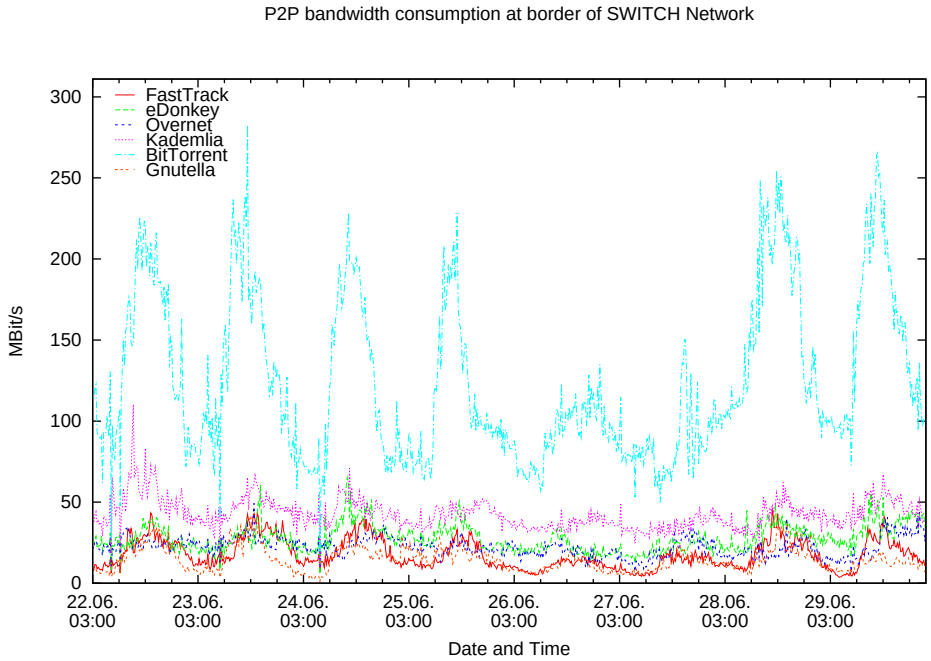


Figure 4-7: P2P traffic of considered networks

Network	Stateless	Stateful	$\frac{stateless}{stateful}$
BitTorrent	17'099 GB	27'722 GB	61.7%
FastTrack	546 GB	3'786 GB	14.4%
Gnutella	1'572 GB	3'299 GB	47.7%
eDonkey, Overnet, Kademlia	14'650 GB	25'146 GB	58.2%
Total	33'867	59'953	56.5%

Table 4-9: Comparison to port based method

stateful method, it is interesting to examine how large the difference is related to the values of our approach.

The 30 day measurement (Aug.-Sept.) was used for this comparison. TCP and UDP traffic was accounted for all considered P2P networks. Since eDonkey, Overnet and Kademlia all use the same default download port, a stateless method can't separate these three protocols from each other. Therefore they are summed up for this comparison.

As can be seen in Table 4-9 the difference is quite substantial, especially for the FastTrack network, whose clients use some more sophisticated techniques to camouflage themselves. Overall the stateful port based method used in the implementation seems to estimate 1.4 times more P2P traffic than the stateless method. These numbers exceed those of [8] where depending on the P2P network the difference was between 30% and 60% at that time.

4.4 Population Tracking Results

Population tracking is one of the main goals of this thesis. As was assumed in earlier chapters for some P2P networks it should be possible to track the whole user base using a medium backbone network as SWITCH. Unfortunately the SWITCH network seems to be not large enough and there are not enough P2P users within it so that only a part of the population could be tracked. A timeout (*MaxExternPeerInactivity*) value of 6 hours was found to be reasonable since it is below the minimum average uptime (Gnutella; 7.6 hours, see Table 4-3). Increasing the *MaxExternPeerInactivity* value further may "track" more external hosts but the results won't become more accurate since many of the tracked peers probably are not online anymore.

For Overnet a considerable amount of peers can be tracked since they contact neighbor peers frequently. Figure 4-8 shows the tracked Overnet peers of an eight day measurement where the timeout *MaxExternPeerInactivity* value was set to different values. Assuming a population of 800'000 users more than half of the users could be tracked with a timeout high enough.

Figure 4-9 shows the tracked peers of all considered networks with a timeout of 6 hours. It can be seen that P2P protocols with one-tier architectures (Kademlia and Overnet) result in more tracked peers. An interesting observation when comparing the number of active internal peers of Figure 4-2 with the number of tracked external peers is that the latter are less periodic and less regular. Especially the numbers of tracked FastTrack and Gnutella peers seem less dependent on the numbers of SWITCH peers as depicted in Figure 4-10

Network	Avg. Int. Peers	Avg. Tracked Peers	Tracking Factor
BitTorrent	89.3	158'418	1'774
FastTrack	36.6	68'041	1'859
Gnutella	45.1	68'187	1'512
eDonkey	68.1	156'066	2'291
Overnet	43.8	322'458	7'362
Kademlia	97.4	505'869	5'193

Table 4-10: Tracking factors in August

for FastTrack. A possible explanation for this is the varying number of super nodes within the SWITCH network. Ordinary FastTrack and to some degree also Gnutella nodes don't seem to communicate with many other nodes beside their current super node(s) and peers that they are transferring files to or from. But if a peer becomes super node it has numerous connections to other super nodes via TCP and UDP. Moreover many ordinary nodes connect to super nodes or occasionally query them with UDP search requests. Therefore the number of current super nodes in the SWITCH network has significant influence of the amount of tracked external peers.

Table 4-10 shows how many external peers in average can be tracked for an internal peer according to a 30 day measurement with a timeout of 6 hours. Again it is quite obvious that Overnet and Kademlia have the highest chances to track the total population, especially when keeping in mind that their population is estimated below 1 million users in contrast to most other P2P networks. But as explained before, the actual Kademlia network is probably much smaller and this high number of tracked Kademlia peers actually are tracked eMule clients which are part of the eDonkey network mostly. The reason why there couldn't be tracked an equal number of eDonkey hosts is that eMule has extended the eDonkey protocol and tends to communicate more extensive and more frequently with its own kind than with other eDonkey clients.

As Figure 4-8 demonstrates, complete population tracking using SWITCH NetFlow data seems to be difficult in general. A substantial amount of peers from networks where each peer has contact with a lot of different neighbor peers like Overnet and Kademlia can be tracked though. The SWITCH network, consisting of about 30'000 active hosts on average is very small in comparison to the approximately 800 million[6] estimated Internet users worldwide. Moreover it is no transit network and located "at the border" of the Internet. Since it is an educational network it has fewer P2P users in comparison to regular ISP networks. Overall there are too little P2P users which could be used for the tracking. To track the complete population the NetFlow data of a large back bone provider would be needed, e.g. one that possesses transatlantic fibers.

One possibility to improve tracking in the SWITCH network is to run some very powerful super nodes that are contacted by most active peers once in a while. This could be an option for eDonkey where there are only a few servers worldwide, e.g. the largest one (*"Razorback 2"*) is directly connecting to about 25% of all eDonkey users and communicates with a substantial part of the

[6] As of September 2004, see World Internet Usage Stats and Populations statistics http://www.internetworldstats.com/stats.htm.

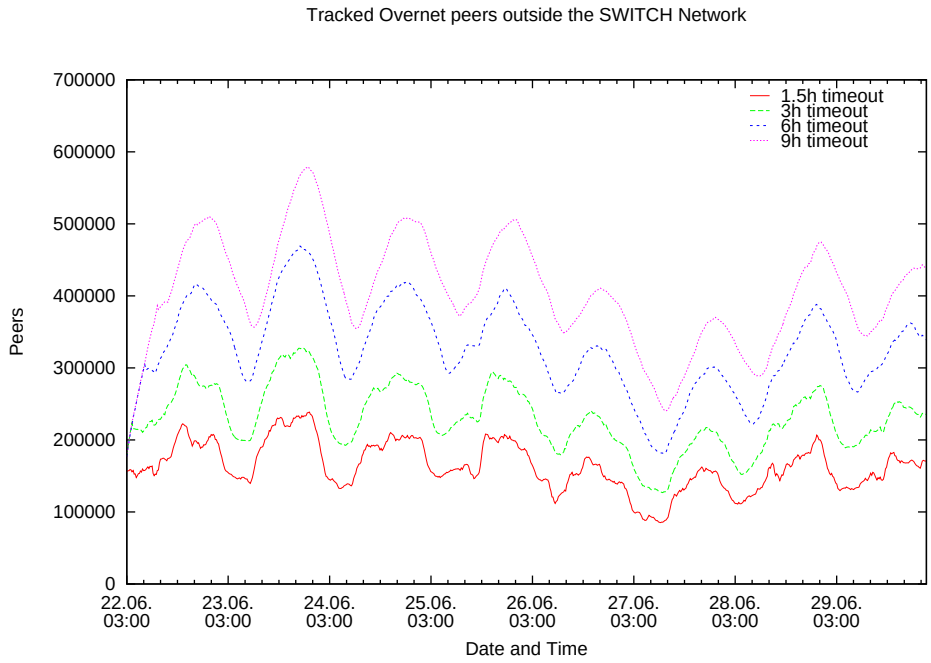


Figure 4-8: Tracked Overnet peers with different timeouts

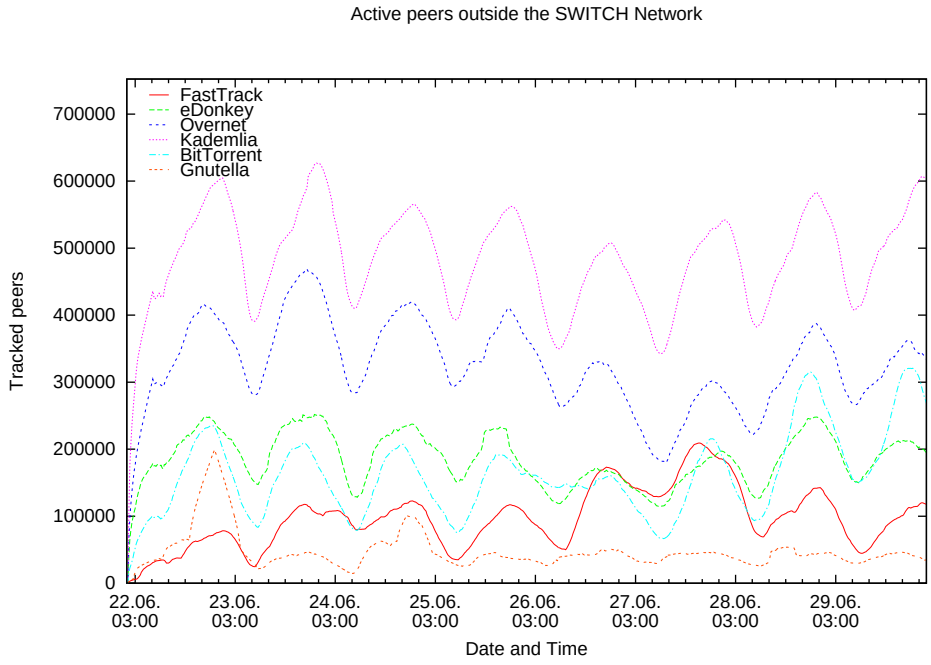


Figure 4-9: Tracked extern peers with 6 hours timeout

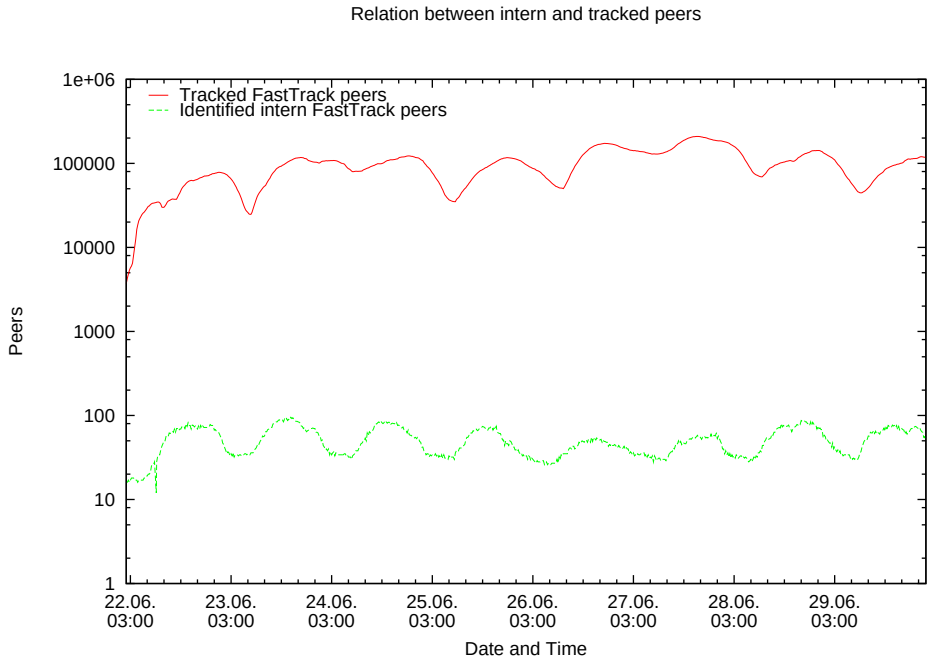


Figure 4-10: FastTrack tracked vs internal peers

remaining users. But as our tests have shown that it takes quite some time to attract some hundred users, even with a fast connection and a powerful computer. Since there are at most some hundred users connected to a FastTrack super node and since there are between 20'000 and 30'000 of them, the benefit of running FastTrack super nodes is assumed to be relatively low concerning the tracking.

4.5 Related Work

Methods in P2P User Prosecution

The main target of the RIAA legal attack was the FastTrack network since it is the leading network in terms of active users. In August of 2004 they also started to prosecute Gnutella, eDonkey and BitTorrent user. The methods used to identify users that share copy righted material are not known to the public. But one can imagine different approaches depending on the P2P network. Tracking BitTorrent users that download a copyrighted file is straightforward analyzing the IP addresses received by the tracker and then polling the peers to check how much of a file they already have downloaded. For the other two-tier based networks one method could be to operate a super node. A super node has an index of all shared files of the connected ordinary nodes. This allows to compare file names, file sizes and possibly file hashes to identify copy righted material. Moreover for Kazaa clients it was possible for older clients to get a complete list of all the shared files from a client directly. Therefore finding IPs of peers that

illegally share content is in general not that difficult, but to identify the persons that actually are responsible for sharing the content is much more difficult due to reasons explained in Subsection 1.5.4.

Other Approaches

The ways to measure P2P traffic have been examined by various other researchers. So far no study has actually tried to identify P2P clients to improve bandwidth estimations or to track other peers. No publication did separated eDonkey from Overnet or Kademlia traffic.

- In [9] a signature based traffic analysis for TCP packets was used. Only the first few packets in a beginning download is inspected with the goal for scalability in mind. Considered protocols were Gnutella, FastTrack, DirectConnect, BitTorrent and eDonkey. The chosen approach worked well for high-speed links too but it is believed that application layer signature methods will become inaccurate with the incorporation of sophisticated camouflage or encryption techniques of modern P2P clients.

- A flow-level based approach was used in [21]. Signalling and download traffic was measured in a large ISP network using state-less default port numbers. Considered P2P networks were FastTrack, Gnutella and Direct Connect.

- At the University of Washington a 200-day trace of FastTrack TCP upstream data was analyzed in [37]. The external download requests were used identified and stored in a 1TB archive using packet inspection methods.

- Another packet level inspection measurement was done in [13] where incoming and outgoing HTTP traffic of FastTrack and Gnutella was captured and analyzed.

- So was in [8] where packet traces of max. 48 hours were analyzed. Only the first 44 bytes (IP, TCP/UDP headers + 4 Bytes payload) of incoming and outgoing, FastTrack, DirectConnect, Gnutella, Napster, BitTorrent, eDonkey and five less popular P2P protocols were captured. One of their statements is that the P2P traffic increased by 30%-60% when using packet inspection in comparison to just using stateless default port methods.

- An interesting approach is presented in [39]. The idea is to relate flows to each other according to source and destination port numbers using a flow relation map heuristic with priorities and SYN/ACKs to identify listening port. The method is not described very detailed though.

- On a campus net and the network of research institute with total about 2200 students and researchers, [38] captured the first 64 Bytes. The approach used for identification is, that unknown flows are induced by flows of known traffic. They used a time window to find induced flows. The higher the window the more “identified” traffic resulted. They also stated, that false positives can’t be recognized without checking the payload.

- Flow measurements in the backbone of a large ISP were done in [7] for May 2002 and January 2003. They determined the server port using the IANA[54] port numbers and the more detailed Graffiti [67] port table, giving precedence to well-known ports. Unclassified traffic was grouped in a "TCP-big " class that includes flows with more than 100KB data transmitted in less than 30 minutes. They noted that identified P2P traffic decreased in January 2003 but the TCP-Big traffic dramatically increased (10.5 times for outgoing and 6 times for incoming directions). Moreover they found that the TCP-Big traffic had a strong correlation with P2P traffic.

- In [61] all TCP traffic to port 4662 was captured with tcpdump. Among a general overview about the eDonkey network their results indicate that a majority of the observed traffic is locally and not worldwide distributed.

- Not exactly P2P traffic identification was the goal in [59] but to get new insights of the FastTrack protocol. They were running a Kazaa super node and two ordinary nodes for 12 hours capturing all traffic to these nodes. Results were some interesting facts about how the FastTrack nodes communicate with each other.

- In [55] a layer 4 switch inspected the first few packets to detect FastTrack download traffic. The traffic of an identified host then was redirected to a caching server, that intercepted and analyzed all downloads.

- Commercial software or hardware to block or limit P2P traffic like *P2P WatchDog* [68] or P-Cube's router software [69] use packet analyzing methods too. They constantly have to update their signatures for protocol changes and new P2P clients.

Chapter 5

Conclusion

The goals of this thesis were to find methods to identify P2P traffic in the SWITCH network and to track external peers of the BitTorrent, FastTrack, Gnutella, eDonkey, Overnet and Kademlia networks. To achieve these goals the described approach as a first step identifies peers in the SWITCH network. Therefore the focus of this thesis was on P2P identification which tries to discover hosts that run P2P clients.

P2P Identification

P2P identification is non trivial and has become even more difficult since quite a substantial amount of the P2P clients use non-default ports. Especially the FastTrack client Kazaa incorporates several techniques, including random port number usage and encryption, to hide itself from detection. This in order to evade traffic limitation and legal consequences.

Various techniques that actively or passively detect or verify P2P clients have been presented in this thesis. A passive identification approach using NetFlow data was implemented as an UPFrame plug-in and used for various measurements. The plug-in can process SWITCH NetFlow data about 10 times faster than real-time and delivers various statistics about P2P and general bandwidth usage.

P2P identification in general will become harder in the future since more and more clients will follow the example of Kazaa and try to hide themselves from legal threats and bandwidth restrictions. Packet inspection as a relatively accurate traffic identification method will get more difficult too since there is a trend for P2P clients to implement encryption mechanisms. The implemented approaches for P2P identification which analyze the neighborhood of potential peers have shown to work in general. But as was observed they have their limits. Peers which have only little contact with other peers, are not always reliably identified by the algorithm since it benefits from numerous connections. Moreover the identification results largely depend on the default port usage of a P2P network. Kazaa clients which choose random ports for signalling and download traffic are much harder to detect since their default port usage is around 10% nowadays. Peers of the other considered networks have a higher default port usage which makes them better identifiable.

P2P Verification

An eDonkey specific technique, a polling approach and self-run clients were used to verify the P2P identification results. For all P2P systems that use a two-tier architecture, a crawling program could identify the super nodes. Using the super nodes, ordinary peers can be identified accurately. According to this verification method, 90% of all eDonkey hosts were identified by the implementation. The missing 10% were not recognized most likely since they were only a short time online or since they didn't transmit enough files.

Polling requires a near-real-time identification of peers and their listening ports which imposes new problems concerning the NetFlow data availability. Verification results show that depending on the protocol at least 52%-95% of all peers are correctly identified as true positives. But overall polling was found to be a difficult verification method since a lot of peers are protected by firewalls or NAT devices and thus do not allow incoming connections which are needed to poll them. Furthermore polling requires the listening ports of a P2P client to be known which can't be achieved for all peers.

P2P Bandwidth Usage

The measurements made with PeerTracker have shown that P2P bandwidth estimation could be improved using the approach that first identifies P2P hosts. Further it has been demonstrated that P2P traffic - depending on the network - accounted by the straightforward stateless method was only 15%-60% of the traffic accounted with the improved estimations. In contrast to other publications the approach proposed in this thesis even allows to identify P2P traffic for different P2P networks which use the same download port as is the case with eDonkey, Overnet and Kademlia.

The total P2P bandwidth consumption in the SWITCH network is of substantial size and is almost three times as high as web traffic. This is due to the fact that P2P traffic is less day time dependent. Moreover the P2P users produce a larger amounts of traffic although there are about 7.5 times more web surfing people in the SWITCH network on average. Comparing measurements from June and August also have shown that the longterm number of WWW users and the number of P2P users do correlate. In the holidays 15% less users were active for both groups.

P2P Population Tracking

To track the whole P2P population - with the peers identified in the SWITCH network as starting population - seems difficult for most P2P systems including BitTorrent, FastTrack, eDonkey and Gnutella. A quite substantial amount (more than 50%) of the whole P2P population could be tracked for the Overnet network using the SWITCH NetFlow data. In general there is a chance that tracking the whole population is possible for P2P systems, which have a one-tier architecture and frequently contact their neighbor peers for network maintenance. This includes Overnet and Kademlia. It is believed that tracking the whole P2P population for these two networks is feasible if the NetFlow data of a larger (transit) backbone network could be used. This certainly would need the implementation to be adapted since it was not designed to be used in such an environment.

Tracking of P2P hosts which use a two-tier architecture can be improved by running self-operated super nodes as *honey peers*. A super node is contacted by far more other peers than an ordinary node. Limiting the tracking to one network only is difficult if there are hybrid P2P clients which use the same port numbers to join different P2P networks. Filtering certain hosts which use default ports from other P2P networks is needed in this situation.

Overall this thesis has demonstrated the possibilities and limits of P2P identification and population tracking.

Chapter 6

Outlook

6.1 Future Work and Improvements

The time for this thesis was limited to 26 weeks and although the goals were achieved, there still remains a lot that could be done in a succeeding semester or master's thesis. The development of the implementations was an iterative process where various changes and extensions occurred that needed the measurements to be redone and compared to each other which was quite time consuming. Therefore some of the following features and improvements could not be implemented yet.

6.1.1 Automated Verification

To increase the accuracy of PeerTracker, it could be extended to actively verify P2P hosts by either selectively polling them using an extended version of the script used in Subsection 4.1 or by using a crawler program to track all super nodes of a network. Of course this is only possible when PeerTracker runs online and analyzes the NetFlow data in real-time.

6.1.2 More General Identification

If the evolution of P2P clients continues following the example of FastTrack, port based passive methods won't deliver accurate results anymore. Identification would have to concentrate on more general characteristics of P2P clients. They e.g. could use communications patterns, timings and traffic volumes to perform a classification. Some of these methods were shortly examined during this thesis but since the described approach promised to be more successful they were not analyzed in more detail.

6.1.3 Auxiliary Network Processors

A combination of flow-based identification with a packet analysis method probably could improve the identification accuracy. Analyzing only some of the packets of suspicious hosts may be fast enough even for large networks. A packet processor (an active router, e.g. a Linux box running Promethos[70]) could be used to check and confirm a potential peer.

6.1.4 More P2P Networks

Worldwide there are some dozens of P2P networks up and running. The six quite popular networks considered in this thesis cover a large amount of the worldwide P2P users but some other popular networks like Warez [71], MP2P [72] or DirectConnect [73] could be added too in order to better estimate the total P2P traffic.

6.1.5 Multi Peer identification

There exists a considerably amount of P2P hosts that run more than one P2P client or a hybrid client like eMule. PeerTracker detects just the easiest identifiable of the running clients. If one just is interested in the total bandwidth all the peers consume or the hosts that run at least one P2P client, this approach is adequate. But when the exact number of active peers is of concern, all clients running on a host should be identified.

6.1.6 TCP flags in NetFlow

Unfortunately the NetFlow data provided by the SWITCH border gateway routers does not contain the accumulated *tcp_flags*. These flags, in particular the TCP SYN flag, could be very useful to detect the listening ports of a host which probably allow to identify P2P hosts with a better accuracy. Without them it is rather costly to find out which of two communicating hosts set up the TCP connection although it seems feasible as is shown in [14]. Only the time stamps of two related flows could be used in a work-around to guess which host set the SYN flag. But this would require a buffer that contains several thousand TCP flows to find two related flows and then to find out which host opened the connection.

6.1.7 Continuous Examination

The current implementation of PeerTracker evaluates all the hosts on a periodic basis which is 15 Minutes in the default configuration. An extension to PeerTracker could add a feature that continuously evaluates hosts that provide enough information yet. This generally would result in faster identification times. A periodic evaluation still would be needed though for the population aging.

6.1.8 State Reload

A possibility to save and reload the state of PeerTracker could be very useful to stop or pause the processing, e.g. for hardware maintenance reasons. PeerTracker makes use of several large and many small hash tables that take significant effort to reconstruct.

6.2 Unsolved Problems

One of the reasons - beside the speedup - to make PeerTracker able to run independent from UPFrame, was a non reproducible bug in UPFrame that oc-

curred spontaneously and that made measurements unreliable since NetFlow packets were dropped constantly. It seemed to depend on the *netflow_replay* sending rate and/or non-controllable circumstances like memory/CPU usage of the computing host. The higher the *netflow_replay* sending rate was, the more often this bug did occur.

References

[1] "SWITCH - The Swiss Education and Research Network." http://www.switch.ch/ (July 2004).

[2] E. D. Team and C. Schlegel, "UPFrame: UDP Processing Framework." http://www.tik.ee.ethz.ch/~ddosvax/upframe/ (July 2004).

[3] "SharmanNetworks." http://www.sharmannetworks.com/ (July 2004).

[4] "Digital Piracy - Definitive P2P piracy figures for Year 2003...." http://www.itic.ca/DIC/News/2004/08/11/P2P_piracy_figures_2003.html (July 2004).

[5] A. Wagner and T. Dübendorfer, "DDosVax Project." http://www.tik.ee.ethz.ch/~ddosvax/ (July 2004).

[6] R. K. C. Chang, "Defending against flooding-based distributed denial-of-service attacks: A tutorial," *IEEE Communications Magazine*, pp. 42–51, 2002.

[7] A. Gerber, J. Houle, H. Nguyen, M. Roughan, and S. Sen, "P2P, The Gorilla in the Cable," tech. rep., AT&T Labs - Research, June 2004.

[8] T. Karagiannis, A. Broido, and M. Faloutsos, "File-sharing in the Internet: A characterization of P2P traffic in the backbone," tech. rep., University of California, Riverside Department of Computer Science, November 2003.

[9] S. Sen and J. Wang, "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures," tech. rep.

[10] B. P. Arno Wagner, "Peer-to-Peer Systems as Attack Platform for Distributed Denial-of-Service," tech. rep., 2002.

[11] S. Racine, "Analysis of Internet Relay Chat Usage by DDoS Zombies," Master's thesis, ETH Zurich, Computer and Networks Lab, March 2004.

[12] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," tech. rep., 2001.

[13] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "Analysis of Internet Content Delivery Systems," tech. rep., University of Washington, December 2002.

[14] B. Tellenbach, "Visualisation of Internet Host Client/Server Behaviour," Master's thesis.

[15] "SETI@home: Search for Extraterrestrial Intelligence at home." http://setiathome.ssl.berkeley.edu/.

[16] "Skype." http://www.skype.com/ (September 2004).

[17] "KaZaA Media Desktop." http://www.kazaa.com/ (July 2004).

[18] "eDonkey." http://www.edonkey.com/ (July 2004).

[19] J. Ritter, "Why Gnutella can't scale. No, really.," tech. rep., 2001. http://www.eecs.harvard.edu/~jonathan/papers/2001/ritter01gnutella-cant-scale.pdf (July 2004).

[20] P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. Riley, and E. Zegura, "Bootstrapping in Gnutella: A Measurement Study," tech. rep., Georgia Institute of Technology, Atlanta, April 2004.

[21] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," tech. rep., AT&T Labs - Research, November 2002.

[22] "Suprnova.org." http://www.suprnova.org/ (July 2004).

[23] "AudioGalaxy." http://www.afternapster.com/ (August 2004).

[24] "freshnoise.com: music after napster, the beat goes on...." http://www.afternapster.com/ (August 2004).

[25] "Earth Station 5." http://www.es5.com/ (August 2004).

[26] "Soribada." http://www.soribada.com/ (August 2004).

[27] J. Liang, R. Kumar, and Y. X. adn Keith W. Ross, "Pollution in P2P File Sharing Systems," tech. rep., 2004.

[28] "Overpeer." http://www.overpeer.com/ (August 2004).

[29] P. Jardas, "P2P Filesharing Systems Real World NetFlow Traffic Characterization," Master's thesis.

[30] "Apple iTunes Music Store." http://www.apple.com/itunes/store/ (August 2004).

[31] "RIAA - Recording Industry Association of America." http://www.riaa.com/ (August 2004).

[32] "More on MGM v.Grokster Ruling." http://www.eff.org/IP/P2P/MGM_v_Grokster/ (August 2004).

[33] "P2P and Music Statistics for August 2004." http://www.itic.ca/DIC/News/2004/09/02/P2P_Statistics_August_2004.en.html (September 2004).

[34] "P2P Entrapment - Incriminating P2P Network Users." http://members.ozemail.com.au/~123456789/p2p_entrapment.pdf (August 2004).

[35] "Cisco Netflow Services Solutions Guide." http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm (August 2004).

[36] "IETF - Internet Engineering Task Force." http://www.ietf.org/ (July 2004).

[37] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," tech. rep., October 2003.

[38] R. van de Meent and A. Pras, "Assessing Unknown Network Traffic," tech. rep., University of Twente, Enschede, October 2003.

[39] J.-J. K. Myung-Sup Kim and J. W. Hong, "Towards Peer-to-Peer Traffic Analysis," tech. rep., POSTECH, Korea, October 2003.

[40] G. Zaugg, "A Light Weight Packet Capturer for High-Speed Links," tech. rep., Swiss Federal Institute of Technology (ETH), Zurich.

[41] "Slyck.com." http://www.slyck.com/ (July 2004).

[42] "mlDonkey." http://mldonkey.org/ (July 2004).

[43] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," tech. rep., March 2002.

[44] "LimeWire - No Spyware, No Adware, No Trojan Horse, Just Pure File-sharing." http://www.limewire.com/ (August 2004).

[45] B. Cohen, "Incentives build robustness in BitTorrent," tech. rep. http://bitconjurer.org/BitTorrent/bittorrentecon.pdf (August 2004).

[46] m. Izal, G. Urvoy-Keller, W. Biersack, P. Felber, A. al Hamra, and L. Garcés-Erice, "Dissecting BitTorrent: Five Months in a Torrents Lifetime," tech. rep., Institut Eurecom, France, April 2004.

[47] "TCPDUMP." http://www.tcpdump.org/ (July 2004).

[48] "Ethereal - A Network Protocol Analyzer." http://www.ethereal.com/ (July 2004).

[49] L. Deri, "nprobe." http://www.ntop.org/ (July 2004).

[50] "Kerio." http://www.kerio.com/ (September 2004).

[51] "OpenMosix - An Open Source Linux Cluster Project." http://openmosix.sourceforge.net/ (August 2004).

[52] "TIK Experimental Cluster." http://www.tik.ee.ethz.ch/~ddosvax/cluster/ (August 2004).

[53] "server.met - Server List for eDonkey and eMule." http://ed2k.2x4u.de/ (August 2004).

[54] "IANA." http://www.iana.com/assignments/port-numbers/services/ (July 2004).

[55] N. Leibowitz, M. Ripeanu, and A. Wierzbicki, "Deconstructing KaZaA," tech. rep., June 2003.

[56] A. Klimkin, "pDonkey: Unofficial eDonkey protocol specification and perl tools." http://sourceforge.net/projects/pdonkey/ (July 2004).

[57] "PeerEnabler." http://www.limewire.com/english/content/uastats.shtml (August 2004).

[58] O. Heckmann and A. Bock, "The eDonkey 2000 Protocol," tech. rep., Darmstadt University of Technology, December 2002.

[59] J. Liang, R. Kumar, and K. W. Ross, "Understanding KaZaA," tech. rep., 2004.

[60] "Limewire Network Crawler Statistics." http://www.limewire.com/english/content/uastats.shtml (August 2004).

[61] K. Tutschku, "A Measurement-based Traffic Profile of the eDonkey Filesharing Service," tech. rep., Institute of Computer Science, Würzburg, April 2004.

[62] "Protowall." http://www.bluetack.co.uk/pwhelp/ (August 2004).

[63] "Methlab - Peerguardian." http://www.methlabs.org/methlabs.htm (August 2004).

[64] T. Klingberge and R. Manfredi, "The Gnutella Protocol." http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html (August 2004).

[65] T. Hargreaves, "The FastTrack Protocol." http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL (August 2004).

[66] B. Cohen, "The BitTorrent Protocol." http://bitconjurer.org/BitTorrent/protocol.html (August 2004).

[67] "Graffiti." http://www.graffiti.com/services/ (July 2004).

[68] "P2P Watch Dog 5: The Leader in Detecting Evasive Protocols." http://www.p2pwatchdog.com/ (August 2004).

[69] "P-Cube: Global Leader in Service Control & Bandwidth Management for Service Providers." http://www.p-cube.com/ (August 2004).

[70] Lukas Ruf, "The PromethOS Homepage." http://www.promethos.org, 2001.

[71] "Warez.com - Redefining Warez and Giving Warez a New Meaning." http://www.warez.com/ (September 2004).

[72] "MP2P: Blubster - The Largest Online Music Network." http://www.blubster.com/ (September 2004).

[73] "Direct Connect." http://www.neo-modus.com/ (September 2004).

Appendix A

Full Task Description

The following pages contain the original task description of this master's thesis as of April 2004.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

March 7th, 2004
Arno Wagner,Thomas Dübendorfer

Master's Thesis:

Population Tracking and Traffic Characterisation for Current P2P Filesharing Systems

for Lukas Hämmerle <haemmluk@ee.ethz.ch>

1 Introduction

P2P Filesharing

Peer-to-Peer Filesharing is a new "killer application" for the Internet. The idea is that people install a piece of software, that allows them to offer ("share") files from their computer to the community of people running software for the same system. In turn everybody can download files shared by somebody else.

While legal issues around the sharing of copyrighted works are mostly unsolved, from a technical point of view these sytems are usable today. Recent user statistics suggest that at each point in time several million people are using these systems.

One important characteristic of P2P filesharing systems is that they generate a significant amount of the overall Internet traffic. This thesis will focus on detection and characteristics of the generated traffic and some of its implications.

Internet Worms

Internet worms are by now understood well enough by those willing to create them, that massive infection events in a short time are possible. The compromised hosts can then be used for attacks, such as massively distributed Denial of Service Attacks. Worms cannot only infect operating systems directly, indirect attacks on networked applications are also a real possibility. The classical example of this is email. With the recent emergence of P2P filesharing systems, it is to be expected that worms for these infrastructures will be created in the future. Since these P2P systems create a lot of network traffic during normal operation, actually detecting a worm while it is spreading seems to be quite difficult whithout some monitoring equipment targeted specifically at P2P traffic.

The DDoSVax Project

In the joint ETH/SWITCH research project "DDoSVax"[1] abstracted 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.

2 Thesis Task

We want a P2P traffic observation system that is capable of giving realtime statistics for the traffic of the most popular P2P systems. The main focus for this it the possibility to evaluate overall P2P traffic and to potentially do anomaly detection on this traffic to detect worms spreading in a P2P network. The second motivation is that anomaly detection on other network traffic becomes easier if P2P traffic can be identified and ignored.

The basic survey of P2P filesharing systems is not part of this thesis. Instead results from the now completed Bachelor's Thesis of Philipp Jardas should be used.

P2P Population Tracking

Since a pure momentary identification of P2P traffic seems to be difficult, the primary expected contribution of this thesis are algorithms and prototype implementations of systems that identify and track the current host population in the most popular P2P filesharing systems.

To avoid legal problems, this thesis will focus on techniques that do not need to store any data on the past behaviour of specific nodes. Storage of node identities (IP addresses) will be kept to a minimum. If it can be avoided, there will be no long-term storage of IP addresses of P2P system participants. Ideally the tracking approaches do not store more data long-term than one or more typical P2P clients store anyway.

P2P Trafic Identification and Analysis: Offline and Online

The final goal is the implementation of one or several online tracking systems, based on netflow data and possibly helped by P2P clients operated in addition, should that turn out to be necessary.

As an intermediate step, tracking can be done offline in the DDoSVax netflow data archive, where long-term observation data is available. Such an offline analysis has the advantage that there are no realtime constraints and tracking can be done incremental, i.e. the approach can be run on a specific data set, improved and run again on the same data set.

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

Possible Approaches to Anomaly Detection

If there is time left, the thesis should also try to identify possible algorithms that can be used to detect anomalies, like an increased number of connections, in a P2P system, that can be an indication of a P2P based worm.

3 Deliverables

- Code for offline population tracking.
- Code for online population tracking, designed for UPFrame.
- Demonstration setups (data, scripts) for offline and online population tracking.

Documentation and Presentation

A documentation that states the steps conducted, lessons learnt, major results and an outlook on future work and unsolved problems has to be written. The code should be documented well enough such that it can be extended by another developer within reasonable time. At the end of the thesis, a presentation will have to be given at TIK that states the core tasks and results of this Master's thesis. If important new research results are found, a paper might be written as an extract of the thesis and submitted to a computer network and security conference.

- *Master's Thesis Documentation* A concise description of the work conducted in this thesis (task, related work, environment, results and outlook).
- Code documentation (functionality, interfaced) for all code that is part of the thesis results.
- Users guide (setup and operation) for all implemented tools that are part of the thesis results.

Dates

The Master's thesis starts on March 22th, 2004 and is finished by September 22th, 2004. It lasts 26 weeks in total.

Two informal presentations to Prof. Plattner will be scheduled about 2 Months into the thesis and about 4 months into the thesis.

Around the end of the Thesis there will be a formal presentation of the results.

Supervisors

Arno Wagner, wagner@tik.ee.ethz.ch, +41 1 632 70 04, ETZ G64.1
Thomas Dübendorfer, duebendorfer@tik.ee.ethz.ch, +41 1 632 71 96, ETZ G64.1

Appendix B

P2P Networks Table

The following table shows the most common P2P networks in Europe and the USA. Worldwide there are some dozens P2P networks, most of them limited to a small geographical area.

Properties / P2P Networks	Related to	Supernodes for search	File hashes	Established	Developer/Owner	Rather centralized	Rather decentralized	Shared files	Download protocol	Default Connections	Chunked file downloads	Protocol published	Popular clients	Varia
FastTrack		Dynamically assigned supernodes	Yes, UUHash (weak algorithm)	March 2001	Sharman Networks	No	Yes	Large Files < 2GB	HTTP GET, unencrypted	TCP/1214, UDP/1214	Yes	Encrypted, partially reverse engineered	Morpheus, KaZaA, iMesh, Grokster, giFT (OS), MLDonkey (OS)	Based on Gnutella, proprietary clients automatically download software updates, adware-contained clients, encryption used for searches/queries
iMesh	FastTrack (earlier iMesh)			1999	iMesh	No	Yes	Large Files < 2GB	HTTP GET, unencrypted	TCP/1214, UDP/1214	Yes	Encrypted, partially reverse engineered	Morpheus, iMesh	Had its own network, now part of FastTrack
eDonkey2000		eDonkey Servers (needed to join network)	Yes, MD5	2002	MetaMachine	Yes	No	Large Files < 2GB	eDonkey	TCP/4661 (server connection), TCP/4662 (client connection), UDP/4665 (poll other server)	Yes	Only reverse engineered	Shareaza P2P, Morpheus, iMesh, MLDonkey, eDonkey, eMule, lMule, aMule, xMule	Worlwide about 80 active public servers.
Overnet	eDonkey transfer protocol	No, searches and publishes in a completely decentralized way	Yes, MD5	2002	Jed Mcalleb, MetaMachine	No	Yes	Large Files < 2GB	eDonkey	UDP/Random Port (extensive use of UDP!) TCP/4662	Yes	Only reverse engineered	Overnet, eDonkey	Extension of eDonkey with different way of publishing/finding files, XOR routing policy
Kademlia (eMule)	eDonkey transfer protocol	No, completely decentralized, incompatible to Overnet	Yes, MD5	2004	Emule project	No	Yes	Large Files < 2GB	eDonkey	UDP/4672,4673 (extensive use of UDP!) TCP/4662	Yes	Emule is open source	Emule	Like Overnet uses XOR routing protocol but incompatible to Overnet
BitTorrent		No built-in search facility, torrent links or files are published on websites	Yes, SHA1	July 2002	Bram Cohen	No	Yes, no central resource allocation	Large Files < 5GB	BitTorrent protocol using Bencode	TCP/6969 (Tracker) TCP/6881-6889 (since v. 3.2 up to 6999)	Yes	Yes	The Shad0w's experimental client, Burst!, BitComet, Azureus	Tracker only needed to find other peers, temporary network
Gnutella		Optional Ultrapeers	No	March 2000	Community	No	Yes	< 1GB	HTTP GET	TCP/6346-6347, UDP/6346-6347	No	Yes	Limewire (Java), Morpheus, Shareaza, ...	Scales badly
Gnutella2	Extended Gnutella	Ultrapeers cache file location and hashes of leaf nodes	Yes, SHA1	March 2003	Mainly Michael Stokes (Shareaza)	No	Yes	< 1GB	HTTP GET	TCP/6346-6349, UDP/6346-6349 (used for searches)	Yes	Yes	Limewire, Bearshare, ...	
Napster/OpenNap		Static Servers		September 1999	Shawn Fanning, Napster	Yes	No	Mostly Music, < 10MB		TCP/6699-6702, UDP/6257	No	Yes	Napster	
Soulseek		Static Servers		2002	Nir Arbel, a former Napster programmer	Yes	No	Mostly Music, < 10MB		TCP/2234+5534	No	Yes	Soulseek	Pay for download privileges (about 0.5% of people), inofficial successor of AudioGalaxy

Others

MP2P
DirectConnect
Ares
EarthStation

Appendix C

NetFlow Format

A NetFlow UDP packet consists of a header (see Table C-1) and several records (see Table C-2) contain the information fields of the flows.

An example header looks like this:

```
Header Netflow Version 5:
  count         =           26
  SysUptime     =     24338230
  unix_secs     =   1086817578   [Wed Jun  9 23:46:18 2004]
  unix_nsecs    =          613
  flow_sequence =        18949
  engine_type   =            0
  engine_id     =            0
  reserved      =            0
```

The most important information fields here are the number of following records/flows (depicted in *count*) and the time stamps *unix_secs* and *unix_nsecs* that tell the time this packet was sent.

The actual data records have 20 data fields that look like this:

```
Record Netflow Version 5:
```

Bytes	Contents	Description
0-1	version	NetFlow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	SysUptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current count of seconds since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	reserved	Unused (zero) bytes

Table C-1: NetFlow version 5 header format

Bytes	Contents	Description
0-3	addr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	SNMP index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of layer 3 bytes in the packets of the flow
24-27	First	SysUptime at start of flow in milli seconds
28-31	Last	SysUptime at the time the last packet of the flow was received
32-33	port	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused byte
37	tcp_flags	Cumulative OR of TCP flags (not used in SWITCH routers)
38	prot	IP protocol type
39	tos	IP type of service
40-41	_as	Autonomous system number of the source, either origin or peer
42-43	dst_as	Autonomous system number of the destination, either origin or peer
44	_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused bytes

Table C-2: NetFlow version 5 record format

```
addr =          10.0.1.12
dstaddr =   192.168.1.120
nexthop =         0.0.0.0
input =                0
output =             255
dPkts =                6
dOctets =            438
First =          23419621
Last =           23955906
port =               4662
dstport =            2747
pad1 =                  0
tcp_flags =             0
prot =                 17
tos =                   0
_as =                   0
dst_as =                0
_mask =                 0
dst_mask =              0
pad2 =                  0
```

Since not all of these fields contain the data that they are supposed to be (e.g. the fields *tcp_flags* is always 0 if NetFlows were emitted by the SWITCH

border gateway routers) and only about 10 of them are useful for most purposes, a compact (human/grep readable) one line representation of this form is often more suitable:

```
UDP  pr
10.0.1.12 si
192.168.1.120 di
1214 sp
2747 dp
438 le
6 pk
1086816659.744 st
1086817196.029 en
536.285 du
```

NetFlows can also be generated on Linux hosts using for example the program nprobe [49] and a flow collector. That way one has an easy to set up mean to debug and test algorithms using NetFlows. Using nprobe to produce NetFlows also has the advantage that all NetFlow fields contain the proper data.

Appendix D

SWITCH Network

Since 1987 the Switch network [1] connects almost all universities, research facilities and other educational institutions in Switzerland. Since the connected organizations are merely situated in an academic environment the network usage can't be compared to that of a regular Internet service provider.

As depicted in Figure D-1, four border gateway routers located in Geneva, Zurich and Basel provide access to the rest of the Internet. Approximately 5% of the total Swiss Internet traffic is routed through one of them. The routers also produce the NetFlow data which is used for this thesis and other network monitoring purposes by DDoSVax.

In Figure D-2 is shown how many GBytes per month left the SWITCH network during the past few years. Under the point "SWITCH server" all traffic from the SunSite FTP mirror, NewsServer and Web-Cache (e.g. Akamai) is summed up.

On an average weekday about 25'000 hosts actively send packets over one of the border gateway routers, as can be seen in Figure D-3. About 560'000 external hosts are contacting internal hosts on average as depicted in Figure D-4. In these figures only hosts show up which sent at least one packet over one of the four border gateway routers.

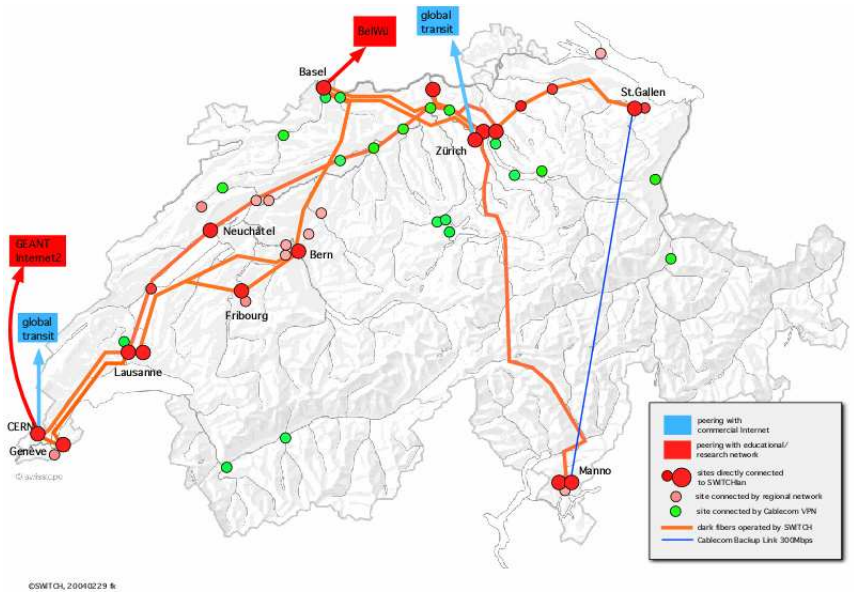


Figure D-1: Switch network

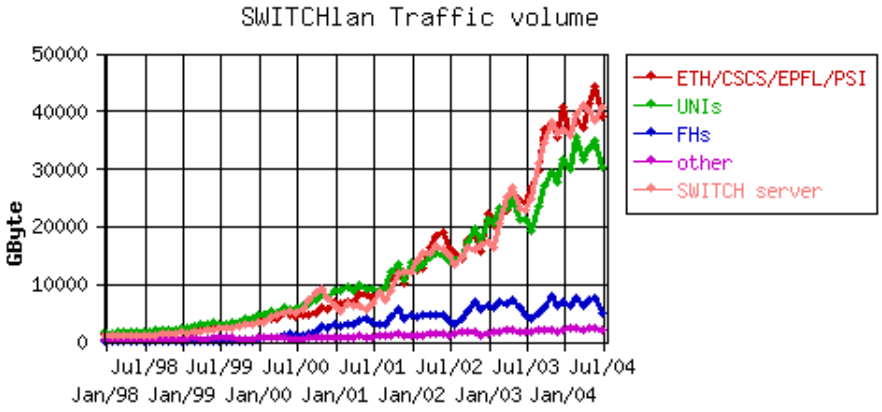


Figure D-2: Switch traffic volume per month since 1998, from [1]

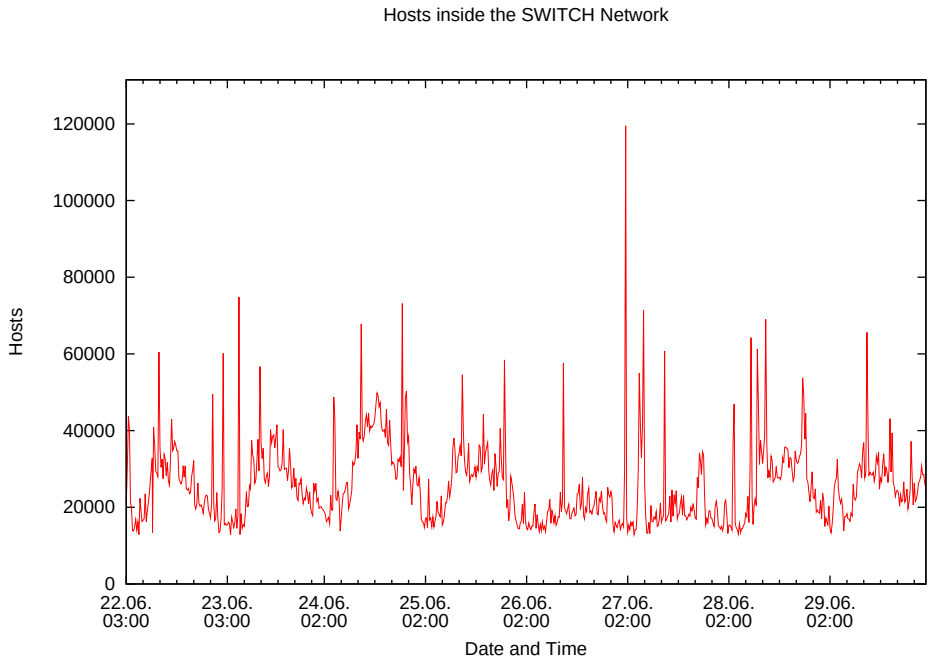


Figure D-3: Active internal hosts in the SWITCH network during 8 days

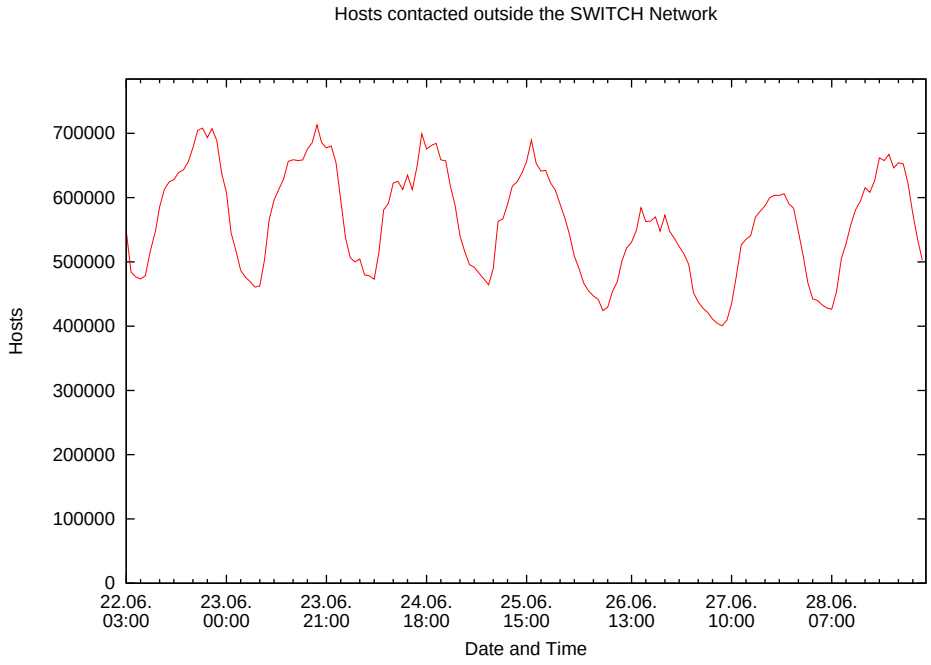


Figure D-4: Active extern hosts contacting hosts within the SWITCH network

Appendix E

P2P Port Usage in the Internet 2

The following data is supposed to demonstrate the P2P camouflage process where more and more P2P clients try to hide themselves using non-default port numbers.

The data is taken from Internet2 statistics pages[1] which produce quite detailed traffic statistics every week since February 2002.

Internet2 is a non-profit consortium led by over 200 US universities and a number of corporate partners from the networking and technology business. Its purpose is to develop and deploy advanced network applications and technologies such as IPv6, IP multi-casting and quality of service.

Looking at Table E-1, one can clearly see, that the amount of unidentified traffic has increased by factor of nine within 30 months. In the same time FastTrack traffic seems to have greatly decreased while the total traffic was almost doubled. Just reading these numbers would lead to the conclusion that the FastTrack network has lost many users during that time. The opposite is true. The FastTrack user base up to date still is the largest of all P2P networks in terms of active users. Another explanation could be that FastTrack traffic was limited or blocked somehow. But since Kazaa, FastTrack's number one client, uses various techniques to evade blocking and since there were transferred considerable 970 GBytes FastTrack traffic in the week of 2004, this explanation looses some evidence.

It is assumed that the measured FastTrack traffic has been declining since Kazaa has started to choose random port numbers, which was in 2002.

Another remarkable fact depicted in Figure E-2 is the huge increase of BitTorrent traffic and a temporary decrease of unidentified traffic in the middle of 2003. These two events seem correlated since BitTorrent traffic has not been identified before April 2003 by Internet 2 routers.

[1] See Internet2 NetFlow: Weekly Reports http://netflow.internet2.edu/weekly/ (September 2004)

Traffic [TB]	Week 2002 02 18	Week 2004 07 19
Total	157.6 (100%)	327.9 (100%)
File-Sharing	80.24 (50.91%)	18.7 (5.73%)
FastTrack	53.74 (34.1%)	0.97 (0.03%)
eDonkey	1.492 (0.95%)	1.998 (0.61%)
BitTorrent	0 (0%)	9.25 (2.82%)
Unidentified	26.68 (18.2%)	71.53 (21.86%)

Table E-1: Internet 2 traffic statistics comparison

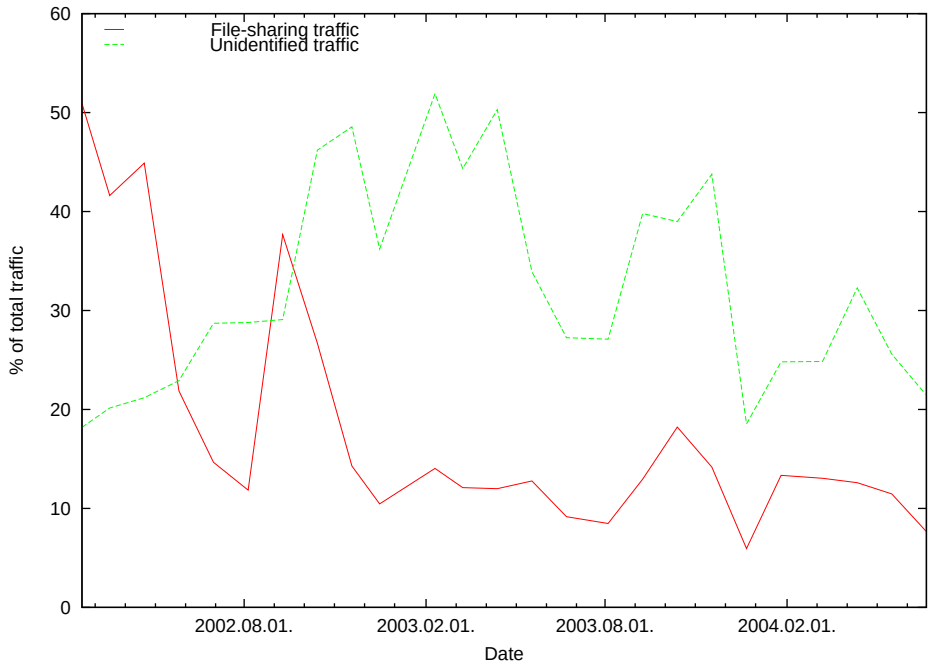


Figure E-1: Internet 2 traffic graph

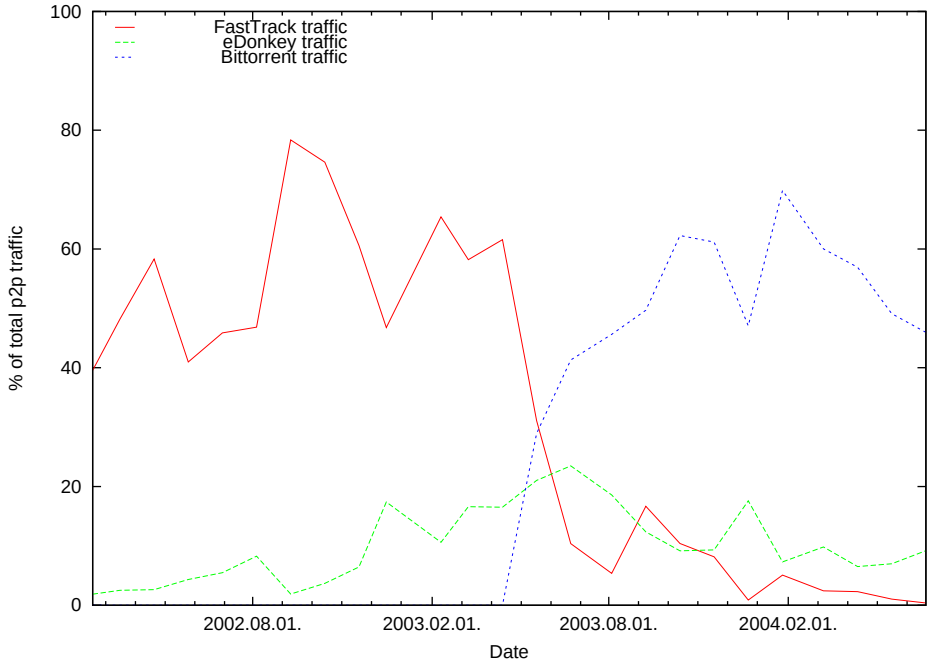


Figure E-2: Internet S2 P2P traffic graph

Appendix F

Identification Code

The following code sample shows the part which actually determines to which P2P network a host belongs to.

```
/*------------------------ Peer type detection --------------------------*/
// Try to identify type of peers

// Precedence:
      NonPeers, Overnet, Kademlia, eDonkey, Gnutella,  BitTorrent, FastTrack
// Overnet: udpremote random, tcpremote 4662
// Kademlia: udpremote 4672/4673, tcpremote 4662
// eDonkey: (udpremote 4665), tcpremote 4662
// Gnutella: tcp/udp ports 6346-6349
// Bittorrent: udpremote none, tcpremote 6881-6889
// Fasttrack: udpremote 1214 (few), tcpremote 1214 (few)

// Non peers
// We first must check if a host is a peer at all

// Mark as non peer if
// - Too less connections after probation period
// - Too less susp. flows/tcpbigflows
if (
        tcpremotetotalcount < MinTCPConn
        || curHost->data->susp < MinSuspConn
        || curHost->data->tcpbig < MinTCPBig
   )
{
        curHost->status = NONPEER;
        return;
}

// Check most used ports
// Most important property for identification
if (
        (
                tcpremoteports[0] == 4662
                && tcpremoteportcounts[0] > MinPeerPortThreshold
        )
        ||
        (
                tcplocalports[0] == 4662
                &&
                tcplocalportcounts[0] > MinPeerPortThreshold
        )
```

```
42     )
43 {
44
45         // Overnet, Kademlia, eDonkey
46         if (TrackKademliaPeers)
47                 goto KADEMLIAID;
48
49 KADEMLIAID:
50         if (!TrackKademliaPeers)
51                 goto EDONKEYID;
52
53         if ( // Kademlia default port usage
54                 (
55                         udplocalports[0] == 4672
56                         &&
57                         udplocalportcounts[0] > MinPeerPortThreshold
58                         )
59                 ||
60                 (
61                         udpremoteports[0] == 4672
62                         &&
63                         udpremoteportcounts[0] > MinPeerPortThreshold
64                 )
65            )
66         {
67                 curHost->type = KADEMLIA;
68                 curHost->status = ACTIVE;
69                 return;
70         }
71
72 EDONKEYID:
73         if (!TrackeDonkeyPeers)
74                 goto OVERNETID;
75
76         if ( // eDonkey uses 4665 to query other servers
77                 (
78                         udpremoteports[0] == 4665
79                         &&
80                         udpremoteportcounts[0] > MinPeerPortThreshold
81                 )
82                 ||
83                 (
84                         udplocalports[0] == 4665
85                         &&
86                         udplocalportcounts[0] > MinPeerPortThreshold
87                 )
88
89            )
90         {
91                 curHost->type = EDONKEY;
92                 curHost->status = ACTIVE;
93                 return;
94         }
95
96 OVERNETID:
97         if (!TrackOvernetPeers)
98                 goto GNUTELLAID;
99
100        if ( // Overnet random port distribution
101                udpremotetotalcount > MinPeerPortThreshold
102                && udpremoteportcounts[0] < MaxPeerPortThreshold
103             )
```

```
104         {
105                 curHost->type = OVERNET;
106                 curHost->status = ACTIVE;
107                 return;
108         }
109
110
111         // If a host as survived to this point it is an eDonkey host
112         curHost->type = EDONKEY;
113         curHost->status = ACTIVE;
114         return;
115 }
116
117 GNUTELLAID:
118 if (!TrackGnutellaPeers)
119         goto BITTORRENTID;
120
121         if ( // Gnutella uses mainly default ports
122                 (
123                         (udplocalports[0] >= 6346 && udplocalports[0] <= 6349 )
124                         &&
125                         udplocalportcounts[0] > MinPeerPortThreshold
126                 )
127                 ||
128                 (
129                         (udpremoteports[0] >= 6346 && udpremoteports[0] <= 6349)
130                         &&
131                         udpremoteportcounts[0] > MinPeerPortThreshold
132                 )
133                 ||
134                 (
135                         (tcplocalports[0] >= 6346 && tcplocalports[0] <= 6349 )
136                         &&
137                         tcplocalportcounts[0] > MinPeerPortThreshold
138                 )
139                 ||
140                 (
141                         (tcpremoteports[0] >= 6346 && tcpremoteports[0] <= 6349)
142                         && tcpremoteportcounts[0] > MinPeerPortThreshold
143                 )
144             )
145 {
146         curHost->type = GNUTELLA;
147         curHost->status = ACTIVE;
148         return;
149 }
150
151 BITTORRENTID:
152         if (!TrackBitTorrentPeers)
153         goto FASTTRACKID;
154
155         // BitTorrent
156         // BitTorrent connections often have src/dst port above suspicous
157         // range and these connections are not stored in ip/port queue
158         // So there possibly are only a few flows for a bittorrent client
159         // why we also use an additional method for identification
160
161         if (
162                 (
163                         tcpremoteports[0] >= 6881
164                         &&
165                         tcpremoteports[0] <= 6889
```

```
166                             &&
167                             tcpremoteportcounts[0] > MinPeerPortThreshold
168                     )
169                     ||
170                     (
171                             tcplocalports[0] >= 6881
172                             &&
173                             tcplocalports[0] <= 6889
174                             &&
175                             tcplocalportcounts[0] > MinPeerPortThreshold
176                     )
177             )
178 {
179         curHost->type = BITTORRENT;
180         curHost->status = ACTIVE;
181         return;
182 }
183
184 float probability = 0;
185 int defaultportcount = 0;
186 if (curHost->data->t6969)
187 {
188         probability += 0.2; defaultportcount++;
189 }
190 if (curHost->data->t6881)
191 {
192         probability += 0.25; defaultportcount++;
193 }
194 if (curHost->data->t6882)
195 {
196         probability += 0.1; defaultportcount++;
197 }
198 if (curHost->data->t6883)
199 {
200         probability += 0.1; defaultportcount++;
201 }
202 if (curHost->data->t6884)
203 {
204         probability += 0.05; defaultportcount++;
205 }
206 if (curHost->data->t6885)
207 {
208         probability += 0.05; defaultportcount++;
209 }
210 if (curHost->data->t6886)
211 {
212         probability += 0.05; defaultportcount++;
213 }
214 if (curHost->data->t6887)
215 {
216         probability += 0.05; defaultportcount++;
217 }
218 if (curHost->data->t6888)
219 {
220         probability += 0.05; defaultportcount++;
221 }
222 if (curHost->data->t6889)
223 {
224         probability += 0.05; defaultportcount++;
225 }
226
227 if (curHost->data->t6969 && defaultportcount >= 3)
```

```
228 {
229         probability += 0.4;defaultportcount++;
230 }
231
232 if (defaultportcount > 6)
233 {
234         probability += 0.4;
235 }
236 else if (defaultportcount > 4)
237 {
238         probability += 0.3;
239 }
240 else if (defaultportcount > 2)
241 {
242         probability += 0.2;
243 }
244
245 if (probability >= 0.5)
246 {
247         curHost->type = BITTORRENT;
248         curHost->status = ACTIVE;
249         return;
250 }
251
252 FASTTRACKID:
253 if (!TrackFastTrackPeers)
254         goto ENDID;
255
256         // FastTrack
257         // These are hardest to detect since only about 5-15% of them
258         // use default port. So we first try to use the murp method and
259         // if that does not work we check if the most used tcp port is
260         // the same as the udp port.
261       // Then we check for a random port distribution.
262       // If that is the case we use the absolut number
263         // of default ports in combination. This is not really a good
264         // method alone, but together with the the random portlist usage
265         // it helps improving detection.
266
267 if (
268         tcpremoteports[0] == 1214
269         &&
270         tcpremoteportcounts[0] > MinPeerPortThreshold
271    )
272 {
273         curHost->type = FASTTRACK;
274         curHost->status = ACTIVE;
275         return;
276 }
277
278 if (
279         tcplocalports[0] == 1214
280         &&
281         tcplocalportcounts[0] > MinPeerPortThreshold
282    )
283 {
284         curHost->type = FASTTRACK;
285         curHost->status = ACTIVE;
286         return;
287 }
288
289 // Check if local udp/tcp ports are the same, but not gnutella port
```

```
290  if (
291          tcplocalports[0] == udplocalports[0]
292          &&
293          tcplocalportcounts[0] > MinPeerPortThreshold
294          &&
295          udplocalportcounts[0] > MinPeerPortThreshold
296     )
297  {
298          curHost->type = FASTTRACK;
299          curHost->status = ACTIVE;
300          return;
301  }
302
303
304  // Check for almost random tcp remote port distribution
305  // and occurence of tcp port 1214
306  if (
307          tcpremoteportcounts[0] < MaxPeerPortThreshold
308          &&
309          curHost->data->tcpremote->searchPort(curHost->data->tcpremote, 1214)
310          &&
311          udpremoteportcounts[0] < MaxPeerPortThreshold
312          &&
313          curHost->data->udpremote->searchPort(curHost->data->udpremote, 1214)
314     )
315  {
316
317          // Now check how many of the "default" ports
318          // of a kazaa clients are used
319          if (
320                  (curHost->data->u1214 && curHost->data->u3531)
321                  || (curHost->data->t1214 && curHost->data->u3531)
322                  || (curHost->data->t1214 && curHost->data->t3531)
323             )
324          {
325                  curHost->type = FASTTRACK;
326                  curHost->status = ACTIVE;
327                  return;
328          }
329  }
330
331  ENDID:
332  // If peer could not be identified, curHost remains candidate
333  return;
```

Appendix G

PeerTracker Usage Instructions

Installation

Installation is straightforward.

1. If not yet done, install UPFrame[2]. Upframe is needed because PeerTracker uses some of its libraries, even for stand-alone mode.

2. Install the bz2 development files. E.g. under Debian you simply use *'apt-get install libbz2-dev'*

3. Get PeerTracker source code (on the CD that came with this documentation or ask Arno Wagner, wagner@tik.ee.ethz.ch)

4. Uncompress with *'tar -zxvf peertracker.tgz'* and move the directory *'peertracker'* inside the UPFrame directory

5. UPFrame must have been compiled successfully before PeerTracker can be compiled (*'make all'* in UPFrame directory)

6. Modify PeerTracker's makefile if you didn't install UPFrame in its default location which is *'/home/upframe'*. In that case adapt the value *'INSTALLDIR'* in the PeerTracker makefile.

7. Change working directory to PeerTracker directory and call *'make peertracker'*. This will also compile netflow_sreplay that can be used to feed NetFlow data to PeerTracker in stand-alone mode

8. Finally run *'make install'* which will copy the binary and script files to the right places.

Together with the PeerTracker binary and the configuration file, some shell scripts to start PeerTracker are placed in the UPFrame directory tree. The scripts are adapted for usage in SWITCH network, but it should not be a problem to modify them. One will have to adapt the scripts in UPFrame's *'etc'* directory anyway. Commenting out the warning may do for a default setup.

Command Line Options

Calling the plug-in with the -h option will output:

```
peertracker -h
Usage:
 -w <SCRIPT>       Enable watchdog, restart command
 -e <SCRIPT>       Script that shall be executed after a
                   new statistics file has been written.
 -t <TIME>         Watchdog timeout
 -m                Enable memory exceeding check
 -f <FILENAME>     Path to fifo
 -d                Generate detailed statistics
 -a <FILENAME>     Address list containg <netip> <CIDR> of home net
 -l <FILENAME>     General UPFrame plugin log
 -o                Plugin log to stdout (overrides -l)
 -n <FILENAME>     Plugin log to stdout (overrides -w, -e, -t, -f)
 -p <FILEPREFIX>   Prefix to use for peer dump files
 -s <FILEPREFIX>   Prefix to use for p2p statistics files
 -c <FILENAME>     Config file
 -h                Print usage hints
```

See Table G-1 for a more precise description of the command line options. After you installed and configured PeerTracker, a simple run of could look like this:

1. Start *'upframe/etc/ptstart.sh'*
2. Then call *'upframe/etc/ptsendflows.sh'* with some flows as arguments, e.g. *'/netflow/archive/*'*

PeerTracker will output the log information to STDOUT or to a file, depending on the arguments in the *'ptstart.s*h' script. It will output some status information every 2 Million flows or if something special happens.

Table G-2 shows the netflow_sreplay command line usage. Based on the original netflow_replay by Arno Wagner it extends its functions in the way that it is possible to delay packets (modification by Bernhard Tellenbach), to send two NetFlow files synchronized (therefore netflow_sreplay) and to feed a named pipe on the local host instead of sending the data to UPFrame. Sending two NetFlow files together is quite useful since the SWITCH NetFlow data for one hour comes in two separate files. The pipe feed allows processing data directly - without UPFrame - by any program that can handle pipes. This has the advantage that processing speed is dynamically adapted and in general as fast as possible while it would be constant when UPFrame is used. Moreover there won't be any packet loss as there could be with UPFrame.

Configuration

The standard PeerTracker configuration file *'peertracker.cfg'* that per default should be in *'upframe/etc'* directory looks like this:

Switch	Detailed description
-w <SCRIPT>	Executed whenever UPFrame's watchdog process recognizes that PeerTracker is hanging or crashed. The script should contain a restart command for PeerTracker.
-e <SCRIPT>	This post-processing script is executed after statistical files (P2P statistics or peer dump files) were written. The script will be called with the name of the statistical file as first argument.
-t <TIME>	Time interval the watchdog should check the heartbeat of PeerTracker.
-f <FILE>	Path to UPFrame's first-in-first-out queue. Usually *'upframe/var/fifo'*.
-d	Not only gather statistics about P2P systems but also some general traffic statistics. Slows down processing a bit.
-a <FILENAME>	File where subnets and masks are saved of the form '*net mask*' where mask is a number in range 0-32 that stands for the subnet mask. Example '*192.168.0.0 24*'.
-l <FILENAME>	Filename of the log file that PeerTracker uses.
-n <FILENAME>	Use stand-alone mode. This option tells PeerTracker to read the NetFlow data from a named pipe (FIFO) instead getting the data from UPFrame. This mode is faster and more secure than the UPFrame plug-in mode. The named pipe will be deleted on exit.
-o	Instead of logging to a file, log to standard output.
-p <FILEPREFIX>	Filename prefix of the peer dump files. Appended by '*_YYYYMMDDhhmm.dat*' time stamp.
-s <FILEPREFIX>	Same as above but for the P2P statistic files. Appended by '*_YYYYMMDDhhmm.dat*' time stamp.
-c <FILENAME>	Filename of the PeerTracker configuration.
-h	Show short usage info.

Table G-1: PeerTracker command line options

Switch	Description
-i <IP>	Target IP address of the host running UPFrame, default is 127.0.0.1. Only needed when feeding UPFrame.
-p <PORT>	Target UDP port of the host running UPFrame, default is 19999. Only needed when feeding UPFrame.
-d <USEC>	Delay in micro seconds between packets. Does not include time for reading, decompressing and sending the packet, but is a purely additional delay.
-x <NUMBER>	Every x-th packet, the specified delay takes effect. Default is 0
-c <PACKETCOUNT>	Number of NetFlow packets that are sent. Default is all.
-b <BUFFERSIZE>	Send buffer size. Default is 1024KB.
-f <FILE1>	Read from <FILE1> instead of STDIN. Giving argument '-' will still read from STDIN. If <FILE1> ends in '*.bz2*', bzip2 uncompression will be done on it. Note: Only <FILE1> can be read from STDIN.
-g <FILE2>	Read from <FILE2> instead of STDIN. Giving argument '-' will still read from STDIN. If <FILE2> ends in '*.bz2*', bzip2 decompression will be done on it. This argument is optional to make netflow_sreplay backward compatible.
-n <NAMEDPIPE>	Instead of sending the NetFlow packets to UPFrame put them in named pipe <NAMEDPIPE> on the local host. This allows faster and more secure computation since there won't be any packet loss. If this argument is not given data is sent using UDP.

Table G-2: netflow_sreplay command line options

```
# Configuration file of PeerTracker
# The following setup corresponds to the hardcoded default values
# which should be reasonable for most purposes

# Comments start with a #
# Format is '%ParameterName%  %Number%'

# Unknown %ParameterName% is ignored
# %Number% must be an unsigned integer including 0

# Time in seconds after that all the hosts gets evaluated
EvaluationInterval 900

# Suspicous portrange end port, start port is a 1024
SuspPortRange 30000

# TCP big flow length in bytes
TCPBigLength 102400

# TCP big flow minimum duration in seconds
TCPBigDuration 10

# Number recently contacted neighbours that are stored in
# ip_port_hashed_queue
# All parameters beginning with Min or Max are relational
# to this value
NeighbourHosts 100

# Period in seconds that a host at least is observed
# before it gets deleted or assigned 'active peer'
# or 'dead peer' status
ProbationPeriod 400

# Maximum time in seconds an intern (home net) peer
# may be inactive before it gets assigned 'dead peer'
# status
# Must be greater than Probation Period
MaxPeerInactivity 600

# Maximum time in seconds a host can be candidate
MaxCandidateLife 1800

# Maximum time in seconds an extern host can be
# inactive before deletion
MaxExternPeerInactivity 10800

# Maximum time in seconds a 'dead peer' still
# remains in hosts pool
MaxAfterLife 3600
```

```
# Maximum number of neighbours (out of
# NeighbourHosts) that have a
# non-default port as most used port
MaxPeerPortThreshold 6

# Minimum number of neighbours (out of
# NeighbourHosts) that must use the
# default port to consider a host to be part
# of a network
MinPeerPortThreshold 8

# Minimum occurrence (out of NeighbourHosts) of
# a port to be considered as incoming UDP/TCP port
MinIncomingPortThreshold 60

# Minimum number of TCP flows for a host to be
# considered as peer at all
MinTCPConn 3

# Minimum number of suspicous flows for a host to
# be considered as peer at all
MinSuspConn 3

# Minimum number of TCP big flows for a host to
# be considered as peer at all
MinTCPBig 3

# Wether to track extern hosts
# If 0, only peers in home net are tracked
OnlyTrackHome 0

# Wether to dump eDonkey peers
# If 0, eDonkey peers wont be tracked
TrackeDonkeyPeers 1

# Wether to dump Overnet peers
# If 0, Overnet peers wont be tracked
TrackOvernetPeers 1

# Wether to dump Kademlia peers
# If 0, Kademlia peers wont be tracked
TrackKademliaPeers 1

# Wether to dump FastTrack peers
# If 0, FastTrack peers wont be tracked
TrackFastTrackPeers 1

# Wether to dump Gnutella peers
# If 0, Gnutella peers wont be tracked
TrackGnutellaPeers 1
```

```
# Wether to dump BitTorrent peers
# If 0, BitTorrent peers wont be tracked
TrackBitTorrentPeers 1

# Dump hosts periodically after each
# evaluation interval
DumpHosts 0

# How many hosts should be dumped,
# set to 0 for no limit
DumpLimit 1000

# Wether to dump the non peers after each
# evaluation step
# If 0, non peers won't be dumped
DumpNonPeers 0

# Wether to show all hosts or only hosts in home net
# If 0, all hosts (including extern hosts) will be dumped
# Use this with caution,  since there may be
# millions of extern hosts to dump
DumpOnlyHomeHosts 1

# Wether to dump candidate peers
# If 0, candidate peers won't be dumped
DumpCandidatePeers 0

# Wether to dump active peers
# If 0, active peers won't be dumped
DumpActivePeers 1

# Wether to dump dead peers
# If 0, dead peers won't be dumped
DumpDeadPeers 1

# Wether to dump overdead peers
# If 0, overdad peers won't be dumped
DumpOverdeadPeers 0
```

The format is *'ParamaterName Value'*. The parameter names are equivalent to the variable names in PeerTracker. The values of this file correspond to the default values used for the measurements in this thesis. These values would be reasonable for most setups.

In the UPFrame etc directory there are two shell scripts called *'ptreloadconfig'* and *'ptdumppeers.sh'*. As their names let assume, they can be used to make PeerTracker reload its configuration file during runtime. This works by sending the PeerTracker process the *USR1* signal. Running *'ptdumppeers.sh'* will make PeerTracker dump all its stored hosts according to the configuration file (obey-

ing *DumpLimit* and other parameters) using the *USR2* signal. Depending on which hosts and how many hosts are dumped, this can take quite some time and my fill up your hard drive (approximately 2.7 KBytes per host), so be careful when using this command during plug-in mode. It could result in packet loss.

Output

P2P statistics files and host dump files are saved in a Perl format which means that each value can be read in by a Perl script by applying the *eval()* function to every line. An example of a partial P2P statistics file looks like this:

```
# This is a perl data dump file, use eval to import data

$P2PStats{'1087859714'}{'infos'}{'datavalid'} = 'yes';
$P2PStats{'1087859714'}{'infos'}{'flowtime'} = 1087859714;
$P2PStats{'1087859714'}{'infos'}{'processingtime'} = 912;
$P2PStats{'1087859714'}{'infos'}{'flowcounter'} = 11000000;
$P2PStats{'1087859714'}{'infos'}{'flowpacketcounter'} = 379279;
$P2PStats{'1087859714'}{'infos'}{'flowpacketloss'} = 0;
$P2PStats{'1087859714'}{'config'}{'evaluationinterval'} = 900;
$P2PStats{'1087859714'}{'config'}{'suspportrange'} = 7000;
$P2PStats{'1087859714'}{'config'}{'tcpbiglength'} = 102400;
$P2PStats{'1087859714'}{'config'}{'tcpbigduration'} = 10;
$P2PStats{'1087859714'}{'config'}{'neighbourhosts'} = 100;
$P2PStats{'1087859714'}{'config'}{'probationperiod'} = 400;
...
```

Using the post-processing script the statistics files for example can be compressed and then transmitted to a storage server where eventually another scripts could generate statistical graphics.

If the *-d* switch is enabled there are 171 values written in the P2P statistics file, including the configuration information and some PeerTracker specific information like packet loss or the flow counter. For host dumps, 59 data fields are written out for each host together with the processing time and the flow time stamp. The field names are mostly self explaining.

The data is saved in a Perl hash array whose keys are the Unix time stamps and whose values are sub hash arrays containing the actual information. This way it is easy to read in several statistic files to analyze the contained data. See Appendix H for scripts that process the statistics files to output the data in a human readable format.

Appendix H

Used Software

Written/Modified software

- PeerTracker: Plug-in for UPFrame that identifies and tracks P2P hosts. Can also be used in stand-alone mode without UPFrame.

- {show|analyze}{P2PStats|Peers} Perl scripts: Read files or directories of PeerTracker output files and either show their information in a human readable format or create gnuplot data files, scripts and graphics.

- netflow_sreplay: Extended version of netflow_replay that can replay two NetFlow files at once, preserving synchronization. Moreover this program can be used if PeerTracker is used in stand-alone mode. NetFlow data then is written to a named pipe on the local host where it can be read by PeerTracker.

- isSwitch: C version of isSwitch (Perl), tells if an IP is within SWITCH network.

- ip_port_hashed_queue: Data structure described in Section 3.3.2, there is also a test program for verification purposes.

- nls: Perl script to show/symlink NetFlow file names in a human readable format.

- {eDonkey|Overnet|Kademlia|FastTrack\BitTorrent}Tracker: Perl scripts to identify and track internal/external P2P hosts using different methods.

- Safe{eDonkey|Overnet|Kademlia}Tracker: Perl scripts to identify safe P2P hosts, which have a high probability to be a peer of a certain network.

- verifyPeer: Perl script that reads a PeerTracker peer dump file and actively polls the identified hosts in order to verify them.

- verifyWWW: Perl script to verify WWW hosts.

DDoSVax related

- UPFrame 0.2: UDP processing framework to do online measurements
- netflow_to_text: To convert NetFlow data into a human/grep readable format
- udp_counter_single.pl: As a flow collector for nprobe

Measurement related

- nprobe 3.0.3: To create NetFlow files on a Linux box
- gnuplot 4.0: To visualize the measurement data

Programming related

- valgrind: To check for memory leaks in code
- gprof: To analyze plug-in execution profile

P2P client related

- BitTorrent 0.332
- TorrentSpy-0.2.4.26-win32: To examine torrentfiles
- Azurueus 2.0.8.4
- eMule 0.4.2e
- iMesh V4
- Kazaa Multimedia Desktop 2.7
- KaZuperNode s147
- eDonkey Overnet 0.5.2