Traffic Measurements of P2P Systems

Dragos Ilie, David Erman, Adrian Popescu, Arne A. Nilsson

Dept. of Telecommunication Systems
School of Engineering
Blekinge Institute of Technology
371 79 Karlskrona, Sweden
{dragos.ilie, david.erman, adrian.popescu, arne.nilsson}@bth.se

Abstract— **The paper reports on a measurement infrastructure developed at the Blekinge Institute of Technology (BIT) with the purpose to do traffic measurements and analysis on Peer-to-Peer (P2P) traffic. The measurement methodology is based on using application logging as well as link-layer packet capture. This offers the possibility to measure application layer information with link-layer accuracy. Details are reported on this methodology, together with description of the BIT measurement infrastructure. The paper also reports on traffic measurements done on BitTorrent and Gnutella protocols from an end-client perspective, together with some measurement results of salient protocol characteristics. Preliminary results show a high degree of variability of the BitTorrent and Gnutella traffic, where in the case of Gnutella a large contribution is given by the signaling traffic.**

I. Introduction

Over the last years, P2P file sharing systems have evolved to be some of the major traffic contributors in the Internet [13]. P2P applications have now become immensely popular to the Internet community, due to characteristics like communication among equals (computers are acting as both clients and servers) as well as pooling and sharing of exchangeable resources such as storage, bandwidth, data and CPU cycles. Although an exact definition of ”P2P systems” is still debatable, such a system typically represents a distributed computing paradigm where a spontaneous, continuously changing group of collaborating computers act as equals in supporting applications such as resource redundancy, content distribution, and other collaborative actions. The roles of the computers is determined based on the perceived system performance.

In most cases the peers act from the network’s edge instead of it’s core, and they can dynamically join and leave the network, discover each other and form ad-hoc collaborative environments. Each of the participating peers is sharing and exploiting the resources brought collectively to the network pool. The resources needed for the execution of a specific (application) task are dynamically aggregated for the required time period, e.g., by swarming techniques. Beyond that, the allocated resources return to the network pool. These features allow the P2P system to still provide services even when losing resources, in contrast to the classical client-server concept where failures in the system may completely disrupt the service.

There are currently a number of architectural designs for P2P systems, which follow different strategies used for resource discovery and content distribution [16], [20], [27]. For instance, resource discovery can be done either with the help of a centralized directory (e.g., Napster [18]) or with the help of a decentralized directory (e.g., KaZaa/FastTrack [14]) or with the help of query flooding (e.g., Gnutella [15]). On the other hand, content distribution can be done either in a distributed way (so-called ”pure P2P” system, e.g., Gnutella) or server mediated (so-called ”hybrid P2P”, e.g., Napster, BitTorrent [3]) or based on a hybrid client/server model (e.g., SETI [26], GRID [8]) or even based on a pure client/server model (e.g., WWW). Specific advantages and drawbacks are associated with every architectural design, as reported in [16], [20].

A serious consequence however is related to the high traffic volumes caused by P2P systems, which are due to both signaling traffic and data traffic. Furthermore, another serious consequence is related to the high variability introduced by P2P systems in the Internet traffic patterns, with fluctuations that strongly variate in time and space. For instance, recent measurement studies showed that P2P traffic may come up to 80% of the total traffic in a high speed IP backbone link carrying TCP traffic towards several ADSL areas and that Long Range Dependence (LRD) properties and the degree of traffic self-similarity in traffic seem to reduce with the predominance of P2P traffic [2]. Other open problems in P2P systems are related to the appearance of ”mice” (short transfers) and ”elephants” (long transfers) phenomenon in Internet traffic, scalability, expressiveness, efficiency and robustness of search mechanisms as well as security issues.

Measurement studies and analysis of P2P traffic have been rather limited so far. The reason is because of the complexity of this task, which involves answering hard questions related to data retrieval and content location, storage, data analysis and modeling of traffical and topological characteristics as well as privacy and copyright issues. Furthermore, the appearance of what we call the second generation P2P protocols, best exemplified by BitTorrent, further complicated the picture. Compared to the first generation P2P protocols, BitTorrent relies on swarming techniques in combination with the “tit-for-tat” mechanism for creating incentive in content distribution. No search functionality is built into the protocol, and the signaling is geared towards an efficient dissemination of data [6]. On the contrary, the first generation P2P protocols (e.g., Gnutella) are intensively using signaling traffic as well as the exchange of user resources [9]. This protocol diversity further challenges the task of P2P traffic measurements.

Some of the most relevant measurement studies on the first generation P2P protocols are provided in [1], [12], [17], [22]–[25], [28]. A large part of the reported research is focused on P2P signaling traffic, especially generated by the Gnutella and Napster systems. Usually, P2P crawlers have been set up on the Internet to collect P2P topology information. The drawback however is related to the fact that this is an ”active probing” approach, with the well-known associated problems (e.g., bandwidth-intensive, difficulties in mapping large P2P systems, difficulties in collecting temporal dynamics of P2P systems). Another drawback with some of the studies is because the estimates and statistics of the collected P2P traffic are unreliable, due to P2P systems actually having the ability to use arbitrary ports to camouflage their existence.

There are actually only a few measurement studies done on the second generation protocol BitTorrent [4], [10]. This is because the protocol is quite new, about two years old. Traffic has been collected from ”tracker” as well as with the help of modified clients. The drawback however is related to the accuracy of the timestamps at the application level, which is directly depending on the type and version of the computer hardware, OS and application software.

The paper is reporting on a measurement infrastructure developed at the BIT with the purpose to do traffic measurements and analysis of P2P traffic. The measurement methodology is based on using application logging as well as link-layer packet capture. This offers the possibility to measure application layer information with link-layer accuracy. Details are reported on this methodology, together with description of the BIT measurement infrastructure. Furthermore, we also report some of the results obtained in our measurements, which are in the form of salient characteristics of Gnutella and BitTorrent protocols.

The reasons for doing traffic measurements on Gnutella are as follows. Gnutella is a system under strong development and the source code is widely available. Due to its distributed nature, the system is very appealing for developments and improvements. Especially during the last two years the system has undergone various modifications and enhancements. Some of the most important improvements are the development of HTTP-like peer handshaking, traffic compression, better query routing, minimized flood of queries, swarming, partial file sharing, file magnets and UDP signaling [15]. Arguably, the recent changes warrant a new look at how Gnutella behaves in the network as well as invalidate some of the old studies. By doing new measurement studies with higher accuracy, we have the choice of better understanding the Gnutella system as well as calibrating it against old data. Furthermore, we do traffic measurements on BitTorrent because the BitTorrent system is very new, but also because we want to understand the differences between the two generation systems, best represented by Gnutella and BitTorrent. In spite of differences in purposes and architectural designs, we believe that it is important to understand the differences between these systems in terms of file sharing efficiency and efficiency of incentives to prevent free-riding (end-user point of view) but also in terms of traffic variations and characteristics, scalability, and peer evolution (teleoperators point of view). We are targeting at the development of traffic heuristics to recognize P2P traffic at non-standard ports as well. In a first phase, we measure the traffic generated by these systems separately. In a later phase however, we will do traffic measurements in a mixed environment to understand how the traffic created by the two systems interact with each other. This is however a difficult task as it is the one of classifying traffic aggregates.

The rest of the paper is as follows. In Section II we report the measurement methodology used in our measurements. Section III presents the BIT measurement infrastructure. Details about traffic measurements done on Gnutella and BitTorrent are reported in Section IV. Section V reports some of the results obtained in our measurements. Finally, Section VI concludes the paper.

II. Measurement Methodologies

A mixed methodology for traffic measurements of P2P systems has been developed at BIT, which is based on a combination of instrumentation at the application layer with transport flow identification and extraction of packets captured at the link-layer. This solution allows accurate measurements on both generations of P2P protocols.

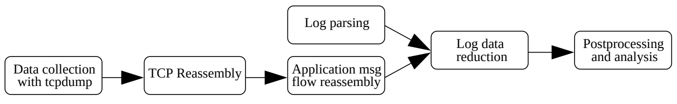
Log parsing
Data collection with tcpdump
TCP Reassembly
Application msg flow reassembly
Log data reduction
Postprocessing and analysis

Fig. 1. Measurement procedures

A. Application logging

Application layer instrumentation involves modifying the specific P2P protocol client to provide logs of pertinent protocol events. This provides access to internal states of the application otherwise not readily available, e.g., download completion ratio and number of connected peers. The internal states may then be mapped to various protocol events, and written into a logfile. Depending on the complexity of the application, this methodology can be more or less difficult. For instance, for the reference BitTorrent client, the adding of logging functionality is fairly easy because the client is written in an interpreted language. On the contrary, the Gnutella client is a larger, and much more complex piece of software, and the consequence is that this type of logging is considerably more difficult.

The process of generating data suitable for analysis is generally a two- or three-stage process. The stages are: log parsing, log data reduction and postprocessing and analysis. If the logs are non-local, i.e., the log creation is not controlled by the application (e.g., for the case of publically available logs), they may contain errors and inconsistencies. Non-locally generated logs thus usually require more stringent error checks for the input data, with the consequence of stricter constraints on the log parser. For locally generated logs, the log format can be predefined and made suitable for the data reduction step.

We used application logging for doing traffic measurements on BitTorrent. We modified the BitTorrent client to generate a log of all major state changes in the client. These logs were written to compressed XML files for metric extraction. By using XML files, the issue of creating a solid parser is made moot, as there are several high-quality XML parsers readily available. Additional details on the setup are presented in [6].

P2P applications may use arbitrary ports, as does BitTorrent [12]. These ports are configurable at run-time and they are often changed from the default port ranges, e.g., to avoid firewalls and facilitate the applications properly functioning behind NAT boxes. By logging using application instrumentation, the issue of detecting which port the application is using is circumvented. When using both application logging and link-layer capture, we also have the opportunity to post-filter the captured link-layer frames to reduce the amount of non-relevant traffic that needs to be parsed and discarded during the application stream reassembly.

An important issue regarding application logging is that the accuracy of the event timestamps is directly linked to the accuracy of the timers in the application runtime environment. Also, as the events are collected only after processing in both the TCP/IP stack in the operating system and the application itself, this further affects the accuracy of the timestamps. In essence, the timestamps can be viewed as being affected by an additive stochastic process, whose major components are governed by applications running, and the amount of processing needed in the TCP/IP stack.

B. Link-layer packet capture

There are situations where application logs may not be feasible, e.g., when the source code for the peer software is not available or when the complexity of the client makes it difficult to localize all places in the source code required to achieve consistent application logging. For instance, most of Gnutella signaling messages are actually compressed, with the consequence that the relationship between the traffic volumes at the application and network layers cannot be captured. Furthermore, to capture and to decode arbitrary flows of Gnutella traffic means that one must examine all TCP flows that exist at the link-layer at the particular time moment. Application logging does not offer this possibility, as is it limited to only own flows and thus cannot be used to capture and decode all Gnutella flows crossing for instance a router.

The major advantage of using link-layer packet capture lies in the ability of capturing a snapshot of the entire network activity on a LAN segment for an arbitrary length of time usually limited by the amount of available storage. The snapshot can then be used to analyze the traffic flow generated by an application and also to search for various correlations that may exist between flows from different applications. Furthermore, link-layer packet capture provides a higher degree of timestamp accuracy than application logs, but at the price of fairly complex stream reassembly software needed to facilitate extraction of the application data stream. The current accuracy of the timestamps is about 10μs, which is almost one thousand times better than that provided by the application logs. Even better accuracy can be obtained through the use of specialized hardware. A pertinent drawback with packet traces is the need for massive amounts of storage. For longer measurements of the order of weeks, data sizes in the terabyte range are not uncommon. The measurement procedure also requires more steps in addition to those used for application logs, as it is shown in Fig. 1. The first three steps are: data collection, TCP reassembly, and application flow reassembly. The last two steps are identical to the ones for application logs.

Data collection was done by using `tcpdump` [11] on the peer nodes. In order to avoid packet loss in `tcpdump` all unnecessary services on the BIT peer nodes were turned off. In addition to that, `tcpdump` was instructed to accept only TCP traffic. Furthermore, the Gnutella and BitTorrent measurements were run sequentially in order to avoid interference between the two types of traffic. Altough combinations of different types of peer-to-peer traffic may be interesting to observe, this was outside the scope of this paper and it is currently planned to be done in the future.

The TCP reassembly module is based on `tcptrace` [19]. `Tcptrace` has some basic TCP reassembly functionality but that was not enough for our purposes. Instead, a new TCP reassembly engine was written similar to the one used by FreeBSD as described in [21]. Among the features present in the engine is the ability to handle out-of-order segments as well as forward and backward overlapping between segments. Our experiments have shown that the module is sensitive to missing segments due to frames dropped by `tcpdump` when the link was under heavy load. Furthemore, the process is very CPU intensive, which makes it unsuitable for real-time processing.

The application flow reassembly module extracts peer messages from each reassembled TCP connection. This process involves finding the peer connection setup, the message boundaries in the TCP stream and optionally data decompression in the case of compressed streams [9]. Furthermore, data reduction can be achieved by compressing the logs using the `zlib` library [7] as well as by data aggregation over time. During postprocessing the logs are demultiplexed based on user chosen constraints: message type, IP address, port number, etc. The output format is usually a two-column table suitable for tools such as `MATLAB` and `gnuplot`.

III. BIT Measurement Infrastructure

The P2P measurement infrastructure developed at BIT consists of peer nodes and protocol decoding software [5]. `Tcpdump` [11] and `tcptrace` [19] are used for traffic recording and protocol decoding. Although the infrastructure is currently geared towards P2P protocols, it can be easily extended to measure other protocols running over TCP as well. Furthermore, we plan to develop similar modules to measure UDP-based applications as well.

The BIT measurement nodes run the Gentoo Linux 1.4 operating system, with kernel version 2.6.5. Each node is equipped with an Intel Celeron 2.4GHz processor, 1GB RAM, 120GB hard drive, and 10/100 FastEthernet network interface. As shown in Fig. 2, the network interface is connected to a 100Mbit switch in the lab at our department, which is further connected through a router to the GigaSUNET backbone.

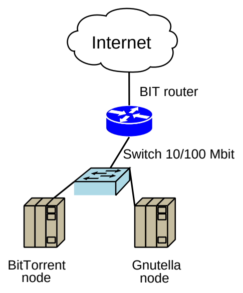

Fig. 2. Measurement setup

Our experience with the current setup has been that the traffic recording step alone accounts for about 70% of the total time taken by measurements. Protocol decoding is not possible when the hosts are recording traffic. The main reason is the protocol decoding phase, which is I/O intensive and requires large amounts of CPU power and RAM memory. To overcome this problem we are developing the distributed measurement infrastructure shown in Fig. 3.

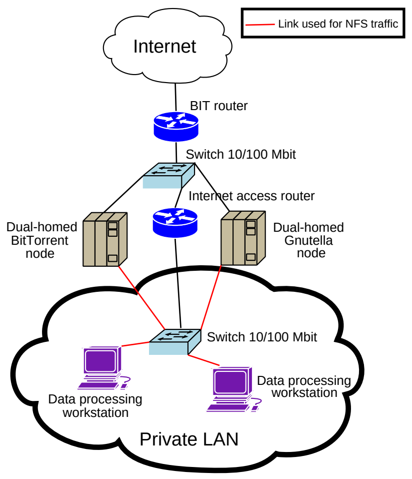

Fig. 3. Distributed Measurement Setup

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

IV. Traffic Measurements

The measurement infrastructure described in section III has been used to collect three sets of measurements of the P2P protocols BitTorrent and Gnutella [5].

A. *BitTorrent measurements*

Two sets of BitTorrent measurements have been performed. The first set used the instrumented version of the reference BitTorrent client as the main measurement tool, with only partial packet capture to determine timestamp accuracy. The second set involved full packet capture and stream reassembly in addition to application logging.

The traffic for the first set of measurements was collected at two different locations over a three-week time period starting on May 3rd 2004. One location was the networking lab at BIT (100 Mbps Ethernet) and the other one was a local ISP with a 5 Mbps link. The measurements represent 12 different runs (with lengths of 2 to 12 days) of the instrumented client, 3 of which were run as the only active application. This was done so as to establish a point of reference without applications competing for available bandwidth. To measure more realistic scenarios, the rest of the runs were done with some temporal overlap [6]. A total of 20GB of uncompressed XML logs were collected in the first set of measurements. After postprocessing, the amount of logs was over 25GB. The logs contain approximately 100 million protocol messages from almost 300000 individual sessions. The BitTorrent log files contain a list of client software states, e.g., tracker announcements, new connections, choke, unchoke, interested, uninterested, along with the timestamps when the state change took place.

The second set of traces were collected as `tcpdump` traces at the BIT networking lab during one week, starting June 4th, 2004. A single instance of the reference client was run as the only application on the measurement node. The set contains 150GB of data, out of which 143GB are `tcpdump` traces. The rest of the data are application logs and postprocessed logs. Approximately 22 million messages were transmitted in 53000 sessions during the second measurement set.

An important issue regarding traffic measurements in P2P networks is the copyright issue. The most popular content in these networks is usually copyrighted material. To circumvent this problem, we joined BitTorrent swarms distributing several popular Linux operating system distributions.

Traffic metrics like peer swarm size, download time, number and types of messages, host persistence and piece popularity have been measured [6]. Some of the most interesting results follow. Figure 4 shows the evolution of the number of connected peers during the downloading (also known as the *leech* phase) for measurement number two. The peer is preconfigured to keep 50 connections up, and the figure clearly indicates a healthy swarm. The client finds the preconfigured number of peers in less that 5 minutes, and does not drop below this number during the entire downloading phase. A short downloading phase with a stable number of peers are two key indicators of swarm health.

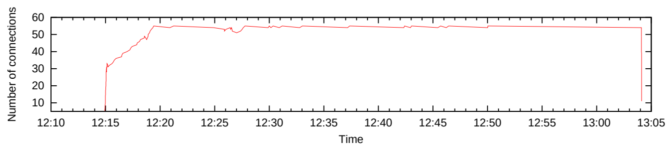

Fig. 4. Swarm size for leech phase

Depending on the type of data being shared in a BitTorrent swarm, the participating peers display different download dynamics. Figure 5 shows the reaction of a swarm to the release of a new version of the RedHat Linux distribution. The leftmost time series is from a swarm sharing the final test release of the distribution, and the rightmost is from a swarm sharing the final release. The final release version was released on May 18th, 2004. The swarm clearly reacts to the release of a newer version, as the swarm size has been halved in less than one day.

Another key issue for a resource sharing system of the incentive, or "tit-for-tat", type is the amount of time peers remain in the swarm after their download is complete. Figure 6 shows the empirical density of the session lengths for the measurement set 2. Note that the x-axis is in hours. The predominance of very short sessions is due to peers connecting to find out whether there is any interesting data on any given peer. A connection attempt is necessary for this, as this information is exchanged during the protocol handshake.

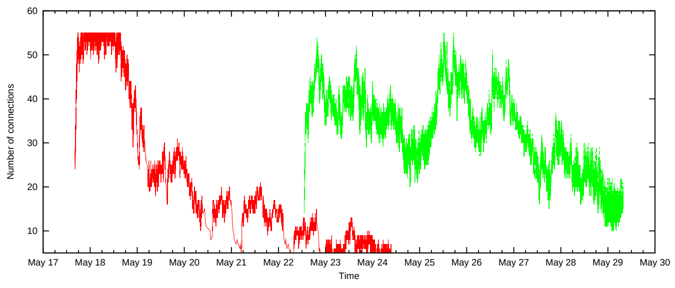

Fig. 5. Evolution of swarm size in reaction to changing resource availability

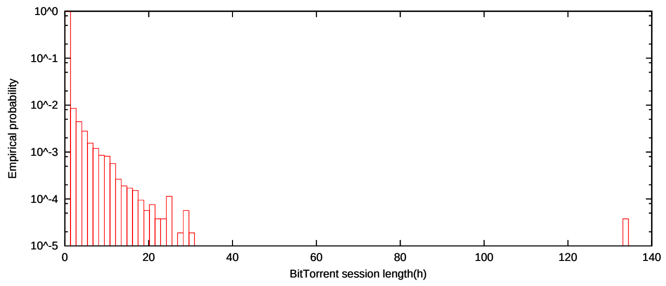

Fig. 6. Empirical density function for BitTorrent session length

B. *Gnutella measurements*

The third set of measurements consists of a one-week long link-layer packet trace of Gnutella signaling traffic. The observed traffic was flowing across one of the peer nodes at BIT running as a Gnutella ultrapeer node. Ultrapeers are Gnutella hosts with more CPU power and connection to high-capacity links. The ultrapeers shield regular nodes from large volumes of signaling traffic.

The trace was collected using the approach described in Section II-B. The total amount of PCAP data is approximately 75GB. The `tcpdump` data generated approximately 9.2GB of compressed log files, which is equivalent to approximately 90GB uncompressed log files. The recorded traffic contains 282 million IP datagrams. The log files show 773 thousand Gnutella sessions that were used to exchange 763 million Gnutella messages. The Gnutella log files use a simple format, where each line contains the timestamp, the source and the destination, message type as well as various details related to the message in question.

Since the software is able to decode Gnutella traffic down to individual message elements both *session metrics* and *message metrics* can be obtained. The session metrics include, e.g., session interarrival time, session duration, number of messages in a session and number of bytes in a session. Some of the message metrics that can be obtained are message type, size, duration, rate and byte rate.

Details of the performed measurements are reported in [9], some of which are reported below.

Fig. 7 shows the empirical density function (i.e., histogram normalized to unity) for the size of Query Routing Protocol (`QRP`) messages. It appears that the density function is bi-modal, although more thorough analysis is required. Furthermore, it appears that most

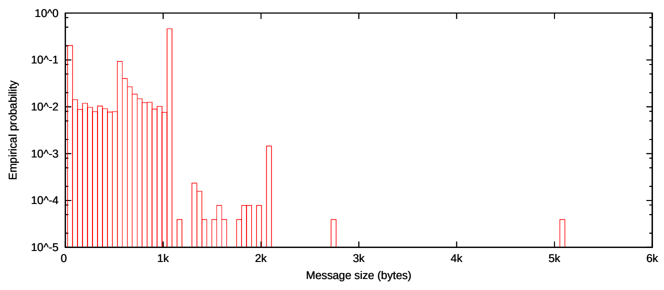

Fig. 7. Empirical density function for `QRP` message size

`QRP` messages have a size of about 1kB. The observed maximum size of the `QRP` messages is slightly above 5kB. This is greater than the 4kB size recommended in [15].

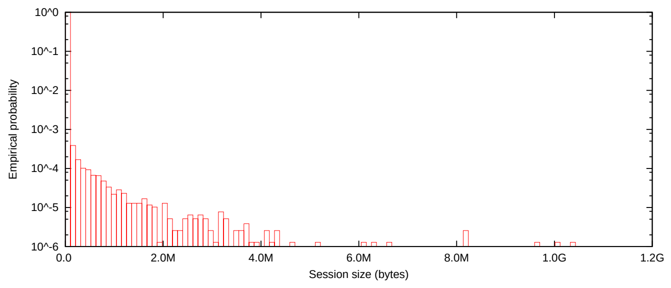

Fig. 8. Empirical density function for number of session bytes

The measurement sample displayed in Fig. 8 shows the empirical density function of the number of bytes transferred due to peer signaling for the duration of a Gnutella session. Most sessions tend to transfer very little data. This could be explained by a large number of unsuccessful Gnutella handshakes due to peers being busy or off-line.

V. Conclusions

A measurement infrastructure developed at BIT with the purpose to do traffic measurements and analysis on P2P traffic has been reported. The measurement methodology is based on using application logging as well as link-layer packet capture. Details have been reported on both methods, together with description of the BIT measurement infrastructure. Furthermore, details regarding traffic measurements done on BitTorrent and Gnutella protocols, together with some measurement results of salient protocol characteristics, have been reported as well.

Our future work will be about further development of the measurement infrastructure towards a distributed measurement setup as well as further analysis of the obtained results. The goal is to find structural similarities or differences among characteristics of the protocols as well as to search for possible invariant characteristics across object flows.

References

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

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

[3] Cohen B. Incentives build robustness in bittorrent. http://bitconjurer.org/BitTorrent.

[4] Qiu D. and Srikant R. Modeling and performance analysis of bittorrent-like peer-to-peer networks. Technical report, University of Illinois at Urbana-Champaign, USA, 2004.

[5] David Erman, Dragos Ilie, and Adrian Popescu. Peer-to-peer traffic measurements. Technical report, Blekinge Institute of Technology, Karlskrona, Sweden, 2004.

[6] David Erman, Dragos Ilie, Adrian Popescu, and Arne A. Nilsson. Measurement and analysis of BitTorrent traffic. In *NTS 17*, August 2004.

[7] Jean-loup Gailly and Mark Adler. zlib. http://www.gzip.org/zlib.

[8] Grid. Grid. http://www.globus.org.

[9] Dragos Ilie, David Erman, Adrian Popescu, and Arne A. Nilsson. Measurement and analysis of Gnutella signaling traffic. In *IPSI 2004*, September 2004.

[10] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *PAM2004*, 2004.

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

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

[13] Thomas Karagiannis, Andre Broido, Nevil Brownlee, Claffy K, and Michalis Faloustos. File sharing in the internet: A characterization of p2p traffic in the backbone. Technical report, University of California, Riverside, USA, 2003.

[14] KaZaa. Kazaa. http://www.kazaa.com.

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

[16] James F. Kurose and Keith W. Ross. *Computer Networking, A Top-Down Approach Featuring the Internet*. Addison-Wesley, 2003. ISBN: 0-201-97699-4.

[17] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages p 233–242. ACM, 2002.

[18] Napster. Napster. http://www.napster.com.

[19] Shawn Ostermann. Tcptrace. http://www.tcptrace.org.

[20] Larry L. Peterson and Bruce S. Davie. *Computer Networks, A Systems Approach*. Morgan Kaufmann, 2003. ISBN: 1-55860-833-8.

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

[22] Stefan Saroiu, Krishna P. Gummadi, Steven D. Gribble Richard J. Dunn, and Henry M. Levy. An analysis of internet content delivery systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

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

[24] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems Journal*, 8(5), 2002.

[25] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking*, 12(2), 2004.

[26] Seti. Seti. http://www.seti.org.

[27] Dimitrios Tsoumakos and Nick Roussapoulos. A comparison of peer-to-peer search methods. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, pages San Diego, California, USA, 2003.

[28] Demetris Zeinalipour-Yatzi and Theodoros Folias. A quantitative analysis of the gnutella network traffic. Technical report, University of California, Riverside, 2002.