Measurement and Analysis of BitTorrent Signaling Traffic

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

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

Abstract

BitTorrent is a second generation Peer-to-Peer application that has been recently developed as an alternative to the classical client-server model to reduce the load burden on content servers and networks. The protocol relies on the use of swarming techniques for distributing content. No search functionality is built into the protocol, and the signaling is geared only towards an efficient dissemination of data. The paper reports on measurement and analysis of BitTorrent traffic collected at the Blekinge Institute of Technology (BIT), Karlskrona, Sweden. We measure and analyze data from local BitTorrent client sessions at BIT. The characteristics of the signaling traffic exchanged among the participating peers in a BitTorrent distribution swarm are investigated. A dedicated approach based on combining instrumentation at the application layer with flow identification and extraction at the transport layer is used for traffic measurements.

1 Introduction

The popularity of peer-to-peer (P2P) networking has lead to a dramatic increase of the volume and the complexity of traffic generated by P2P applications [6, 9, 10]. Applications like Napster, Gnutella, Kazaa and eDonkey have challenged some of the most fundamental concepts that have guided the design of the Internet, e.g., the end-to-end principle [1]. Conflicting interests of different nature have been created, such as the inability to trust the behavior of individual users vs the demand for trustworthy operation. Several important reasons behind the popularity of P2P networking are related to the increased robustness through information redundancy (dissemination), ability to easily share own resources on demand, presence of overlay routing and the

associated sharing of routing functions with the IP routing, fine-grained access control policies, anonymity and encryption.

Over the last years a second generation of P2P applications and associated protocols have appeared, which is best exemplified by the BitTorrent protocol [2]. BitTorrent has been designed with the goal of efficient distribution and replication of content such as CD-images of software and software updates. BitTorrent is a P2P content distribution system designed to quickly, efficiently and fairly replicate data. The system utilizes a central entity, the tracker, to register participating peers. There is no search functionality built into the protocol, so the signaling is geared only towards an efficient dissemination of data. The system relies on using swarming techniques for distributing content, thus reducing the load on the content servers as well as on the networks.

There are actually only a few papers reporting on traffic characteristics of BitTorrent [4, 8]. This is because the protocol is very new, about two years old. Our paper is a contribution towards a better understanding of the BitTorrent protocol and the associated traffic.

The paper reports on measurement and analysis of BitTorrent traffic collected at the Blekinge Institute of Technology (BIT), Karlskrona, Sweden. We measure and analyze data from local BitTorrent client sessions at BIT. The characteristics of the signaling traffic exchanged among the participating peers in a BitTorrent distribution swarm are investigated. A specific approach is used for traffic measurements that is based on combining instrumentation at the application layer with flow identification and extraction at the transport layer.

Packet traces have been collected with the help of the well-known *tcpdump* application [5] and a specific solution has been developed to filter out non-BitTorrent traffic prior to decoding and analysis. The difficulty here is to solve the problems related to the use of arbitrary ports and to identify the relevant BitTorrent sessions. Further, a *tcptrace* module [7] has been developed to decode the BitTorrent signaling traffic. Collected data includes, among others, packet sizes, packet interarrival times, handshake-rate, response times, overlay size and session duration of the P2P protocol.

The motivation is to build up stochastic models for key elements of the protocol that can be further used to build up integrated performance testbed for conducting experiments on P2P traffic control and engineering.

The rest of the paper is organized as follows. In Section 2 we provide a short review of the BitTorrent system and the associated protocols. In Section 3 we report the measurement infrastructure and data collection methodology. Section 4 presents the traffic metrics used in the evaluation of BitTorrent signaling. Section 5 reports on some of the most salient traffic characteristics made available through our measurements. Finally, Section 6 concludes this paper.

2 The BitTorrent Protocol

BitTorrent is a P2P protocol for content distribution and replication designed to quickly, efficiently and fairly replicate data [2]. In contrast to other P2P protocols, the BitTorrent protocol does not provide any resource query or lookup functionality, but rather focuses on fair and effective replication and distribution of data. The sig-

naling is geared towards an efficient dissemination of data only. The protocol is fair in the sense that peers exchange content in a tit-for-tat fashion. Non-uploading peers are not allowed to download. The protocol operates over TCP and uses swarming, i.e., peers are downloading parts, the so-called *pieces*, of the content from several peers simultaneously. The consequence of this is efficient network utilization. The size of the pieces is fixed on a per-resource basis and it can not be changed.

A peer interested in downloading some content by using BitTorrent must first obtain a set of metadata, the so-called *torrent* file, to be able to join a set of peers engaging in the distribution of the specific content. In the following we use the term *swarm*, or *distribution swarm*, to define a set of metadata together with the associated network entities. The metadata needed to join a BitTorrent swarm consists of the network address information (in BitTorrent terminology called the announce URL) of the tracker and resource information such as file size and piece size. An important part of the resource information is a set of Secure Hash Algorithm One (SHA-1) hash values, each corresponding to a specific piece of the resource. These hash values are used to verify the correct reception of a piece. The resource information is also used to calculate a separate SHA-1 hash value, the *info* field, used as an identification of the current swarm. The hash value appears in both the tracker and peer protocols. The metadata does not contain any information regarding the peers participating in a swarm.

A BitTorrent distribution swarm can be partitioned into three network entities and two protocols. The first network entity is a centralized software, the so-called *tracker*, which keeps lists of connected peers as well as information about their evolution. The tracker replies to peer requests for other peer addresses and ports as well as records simple statistics about the evolution of the swarm. The second entity is the set of active peers, which can be further divided into *seeds* and downloading peers, or *leechers*. A seed is defined to be a peer that has already retrieved an entire file or amount of data, and has stopped downloading data from other peers. A seed however may continue to serve other peers. Also, an initial seed is necessary for peers to be able to start replicating the content. Finally, the third network entity is a server, usually a webserver, which provides the metadata required for joining a specific swarm. The distribution of the metadata is not necessarily done via HTTP, but it can be done in any manner. Any way of distributing the torrent file is valid.

The BitTorrent protocols (except the metadata distribution protocol) are the tracker protocol and the peer wire protocol. The tracker protocol uses HTTP. Peers make HTTP GET requests and the tracker sends responses in the returning HTTP response data. The purpose of the peer request to the tracker is to locate other peers in the distribution swarm and to allow the tracker to record simple statistics of the swarm. The peer sends a request containing information about itself and some basic statistics to the tracker, which responds with a randomly selected subset of all peers engaged in the swarm.

The peer wire protocol operates over TCP, and uses in-band signaling for peer communication. Signaling and data transfer are done in the form of a continuous bi-directional stream of length-prefixed protocol messages. A P2P session is equivalent with a TCP session, and there are no protocol entities for tearing down a BitTorrent session beyond the TCP teardown itself. Connections between peers are single TCP sessions, carrying both data and signaling traffic. Once a TCP connection between two

peers is established, the initiating peer sends a handshake message containing the peer id and info field hash (Figure 1). If the receiving peer replies with the corresponding information, the BitTorrent session is considered to be opened and the peers start exchanging messages across the TCP streams. In other cases, the TCP connection is closed. Immediately following the handshake procedure, each peer sends information about the pieces of the resource it possesses. This is done only once, and only by using the first message after the handshake. The information is sent in a *bitfield* message, consisting of a stream of bits, with each bit index corresponding to a piece index.

A peer maintains two states for each peer relationship, namely *interested* and *choked*. If a peer is choked, then it will not receive any data unless unchoking occurs. Usually, unchoking is equivalent with uploading. The *interested* state indicates whether other peers have parts of the sought content. Interest should be expressed explicitly, as should lack of interest. That means that a peer wishing to download notifies the sending peer (where the sought data is) by sending an *interested* message, and as soon as the peer no longer needs any other data, a *not interested* message is issued. Similarly, for a peer to be allowed to download, it must have received an *unchoke* message from the sending peer. Once a peer receives a *choke* message, it will no longer be allowed to download. This allows the sending peer to keep track of the peers that start downloading when unchoked. A new connection starts out choked and not interested. A peer with all data, i.e., a seed, is never interested.

The choke/unchoke and interested/not interested mechanism provides fairness in the BitTorrent protocol. As it is the transmitting peer that decides whether to allow a download or not, peers not sharing content will be reciprocated in the same manner. To allow peers that have no content to join the swarm and start sharing, a mechanism called *optimistic unchoking* is employed. From time to time, a peer with content will allow even a non-sharing peer to download.

Data transfer is done in parts of a *piece* (called *sub-piece*) at a time, by issuing a *request* message. The sub-pieces are typically of size 16384 or 32768 bytes. To allow TCP to increase the throughput, several requests are usually sent back-to-back. Each request should result in the corresponding sub-piece to be transmitted. If the sub-piece is not received within a certain time (typically one minute), the non-transmitting peer is snubbed, i.e., it is punished by not being allowed to download, even if unchoked. Data transfer is done by sending a piece message, which contains the requested sub-piece (Figure 2). Once the entire piece, i.e., all sub-pieces, has been received, and the SHA-1 hash of the piece has been checked, a *have* message is sent to all connected peers.

3 Traffic Measurements

A specific approach has been used for traffic measurements that is based on combining instrumentation at the application layer with flow identification and extraction at the transport layer. The objectives are to capture the type and sizes of protocol objects, to test for the presence of heavy-tailed properties, to find out structural similarities or differences among characteristics of sessions as well as to search for possible invariant characteristics across object flows.

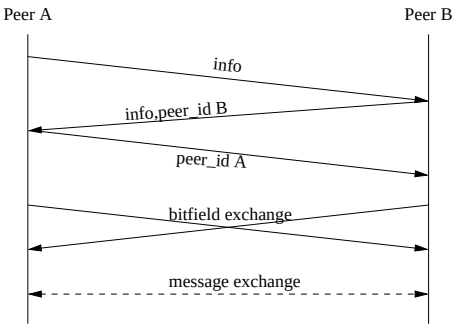

Figure 1: BitTorrent handshake procedure

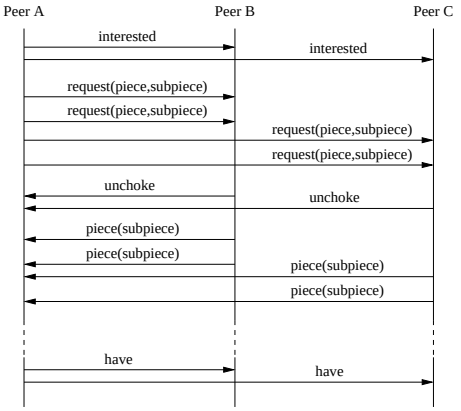

Figure 2: BitTorrent protocol exchange

In this study, statistics about *new connections* as well as about the following protocol messages have been collected: *bitfield*, *request*, *choke*, *unchoke*, *interested*, *not interested*, *piece*, and *have*.

A specific measurement infrastructure has been developed for the collection of P2P traffic, with both passive measurements and active participation of a BitTorrent client in a real BitTorrent swarm [3]. Passive measurements involves flow identification and extraction at the transport layer. To this end we have developed extensions to the popular tcp flow reassembly software *tcptrace* [7], to obtain message timings from the TCP flows and separate the captured tcpdump traces on a per-peer or per-statistic basis. For the other part of traffic measurements, we have instrumented the reference BitTorrent client to provide us with application layer data not readily available in the transport layer, e.g., download completion ratios and number of connected peers. The application instrumentation generates a verbose log of all major state changes in the BitTorrent client and writes these changes to an XML file for later metric extraction. Combining these two approaches provides us with a powerful set of performance evaluation tools. For this paper, the tcpdump traces have been used only to estimate the accuracy of the application level timestamps.

In addition to the modified versions of *tcptrace* and BitTorrent client, specific software has been developed for postprocessing the application logs and create suitable output for further calculations in Matlab as well.

The BitTorrent system uses, like many other P2P applications, arbitrary ports [6]. Even though the protocol has a predefined default port range for connections, clients are free to modify the ports used to suit network parameters like firewall settings and NAT status. Since it is not possible to dynamically change capture filters during runtime with tcpdump, that means that all traffic on non-reserved ports must be collected. Since BitTorrent uses in-band signaling, the entire link layer frame must be therefore collected to capture all protocol messages. The obvious drawback is related to the generation of large tracefiles. We use the application logs to extract the IP addresses and TCP ports used during the client sessions in the network. In a later phase, we

post-filter the main trace to obtain the specific addresses and ports. By modifying the client and taking measurements at the application level, we circumvent the issue of arbitrary port numbers, although the price is that we lose accuracy in the process [3].

The reported measurements have been done by having instances of the BitTorrent client software join several distribution swarms. An instrumented version of the reference BitTorrent client has been used to avoid potentially injecting non-standard protocol messages in the swarm. The client was instrumented to log all incoming and outgoing protocol messages together with a UNIX timestamp. The BitTorrent client is implemented in Python, an interpreted programming language [11]. The drawback is that the accuracy of the timestamps is reduced compared to the actual arrival times of the carrying IP datagrams. By comparing the actual timestamps of back-to-back messages at the application level with the corresponding TCP segments, we have found that the accuracy was approximately 10ms.

The application logs have been collected as XML files, giving about 18 GBytes of log files. By this, much flexibility is obtained when postprocessing the logged data as well as that this is less error prone than the common practice of log parsing since robust XML parsers already exist. After post-processing and extraction of statistics the total amount of data was over 20 GBytes.

The traffic reported in this paper has been collected over a three week time period at two measurement points in Blekinge, Sweden [3]. The first measurement point was the networking lab at BIT, Karlskrona, which is connected to the Internet through a 100 Mbps Ethernet network. The second measurement point was placed at a local ISP with 5 Mbps link. Both measurement points were running the Gentoo Linux operating system, on standard PC hardware. A number of twelve measurement have been done, each of them with a duration of two to seven days (Table 1).

For the first measurement point, no significant amount of different software was running simultaneously with the BitTorrent client. At the second measurement point, the BitTorrent client was running as a normal application, together with other software such as web browsers and mail software. The first measurement point can be viewed as a dedicated BitTorrent client, while the second corresponds to normal desktop PC usage patterns.

The measurements 1 through 3 (Table 1) have been done with a single instance of the instrumented BitTorrent client running. As TCP is known to be very aggressive in using the network, this has been done to minimize the effects of several clients competing for the available bandwidth and to establish a point of reference for the rest of client sessions. Measurements 4 through 8 were started and done simultaneously, as were measurements 11 and 12. The other measurements were done with some temporal overlap, as shown in Figure 3.

An important issue regarding traffic measurements in P2P networks is the copyright. 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. Notably, we joined both the RedHat Fedora Core 2 (FC2) test and release versions. The FC2 'Tettnang' version was released on May 18th, while the rest of the content was available at the start of the measurements. This gave us an unique opportunity to study the dynamic nature of the FC2 swarms. The contents of the measured swarms are reported in Table 2. Two of the swarms

Number	Records	Start	Duration	Location
1	10770695	2004-05-03	2 days, 20 hours	BIT
2	10653466	2004-05-06	3 days, 19 hours	BIT
3	10990569	2004-05-12	4 days, 4 hours	BIT
4	12567283	2004-05-17	7 days	BIT
5	13691459	2004-05-17	7 days	BIT
6	11754838	2004-05-17	7 days	BIT
7	1943636	2004-05-17	7 days	BIT
8	7321166	2004-05-17	7 days	BIT
9	687046	2004-05-13	3 days, 7 hours	ISP
10	2881803	2004-05-18	5 days, 23 hours	ISP
11	9252170	2004-05-22	7 days	ISP
12	5599997	2004-05-22	7 days	ISP

Table 1: Measurement summary

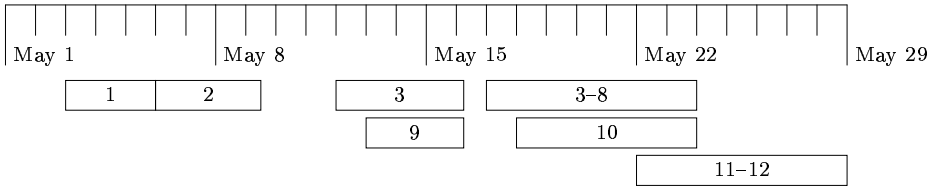

Figure 3: Temporal structure of measurements

have been measured from both measurement points to allow for comparisons, one with temporal overlap, and another without overlap.

Content	Pieces	Size	Measurements
RedHat Fedora Core 2 test3 CD Images	8465	2.2 GB	1–3
RedHat Fedora Core 2 test3 DVD Image	16708	4.3 GB	6, 10
Slackware Linux Install Disk 1	2501	650 MB	4
Slackware Linux Install Disk 2	2627	670 MB	5
Dynebolic Linux 1.3	2522	650 MB	7, 9
Knoppix Linux 3.4	2753	700MB	8
RedHat Fedora Core 2 ‘Tettnang‘ CD Images	8719	2.2 GB	12
RedHat Fedora Core 2 ‘Tettnang‘ DVD Image	16673	4.3 GB	11

Table 2: Content summary

4 Traffic Metrics

The BitTorrent client application logs are in essence timestamped protocol events. This means that metrics like interarrival and interdeparture times are readily available by simple calculations. The possibility does exist to compute detailed statistics on several levels of aggregation as well. Most notably, this offers the possibility to look

into potential burstiness on timescales that are decided by the timestamp accuracy.

Out of substantial amounts of logged data, specific software has been written to extract several important statistics and metrics, to characterize the peer behavior only, and not the entire swarm [3]. To measure the true size of the swarm, active probing of the tracker is necessary. This is, however, subject for future work. The goal is to use accurate characterization and modeling of the behavior of a peer in modeling entire swarms.

A number of metrics have been used for the characterization of the BitTorrent signaling traffic [3]. The most important ones are as follows:

Download time
This is the time it takes for the modified client to do a complete download. This time also provides information about the peer changes from being both a downloading and uploading peer to being a seed, thus offering the possibility to collect statistics about the seed and leecher states.

Session duration and size
A BitTorrent session is equivalent with a TCP session, given that the BitTorrent handshake is completed. As BitTorrent protocol messages are fixed-length messages, there is a one-to-one mapping between the messages sent and received during a session and the session size. A BitTorrent session time is given by the TCP session time, whereas the session size is given by the amount of data transmitted during the TCP session.

Number and type of messages
We count the number of messages of each type in both upstream and downstream directions. Together with the session duration and size, this gives us valuable insights into the behavior of a peer.

Host persistence
We also count the number of unique host IP addresses and peer client IDs. If a given host IP address has a one-to-one mapping to a peer ID and we have a long session time, the peer is considered to be persistent. Persistent peers indicate a healthy swarm in the sense that new peers are more likely to find a larger number of seeds in a swarm with many persistent peers than in one with less persistent peers.

Peer swarm size
The peer swarm size refers to the number of peers observed by the measuring client at any given time. This is not the size of the entire swarm, i.e., the total number of collaborating peers, but the number of peers to which the measuring peer is connected. Information about the total swarm size is only available at the tracker, and therefore it is not considered in the reported measurements.

Piece response times
The piece response time is defined to be the time elapsed between the moment of the initial *request* for any subpiece belonging to a given piece to the moment of the transmission of the associated *have* message. This parameter gives us the possibility to estimate the downstream bandwidth usage.

Piece popularity
The popularity of a piece is given by the number of requests for any subpiece of a given piece. This gives an indication of the effectiveness of the piece selection algorithms of the requesting peers.

5 Traffic Characteristics

Table 3 reports summaries of the measured download times and rates. It is observed that the time before our peer went into the seeding mode varies from roughly 20 minutes up to 6.5 hours. As the content sizes vary with each measurement, we also provide the average download rate for the entire content, i.e., the size/time quota. The download rates also show large disparity, with rates ranging from just over 120 kBytes to over 1.3 MBytes. The measurements 1 through 3 are clearly the most demanding ones in terms of bandwidth utilization.

#	Download	
	Time (s)	Rate (bps)
1	1930	1149520
2	1932	1147908
3	1681	1319445
4	2607	251424
5	3397	202644
6	23000	190416
7	1237	534282
8	6005	120153
9	2723	242776
10	23475	186570
11	19431	224927
12	9106	250989

Table 3: Summaries of download time and average download rate

A summary of the session sizes and durations is reported in Table 4. We also report the number of sessions and unique peer IPs and peer client IDs. It is observed that measurement 6 is different, with regards to both mean session size and session length. Further, the maximum session size for this measurement is more than twice than that of any other measurement. The mean session size is observed to be about twice than that of the corresponding measurement of the same content (measurement 10). As measurements 6 and 10 have the largest session sizes, it is very likely that the session size is related in this case to the total content size (4.3GB).

The minimum session lengths are all set to 0, indicating that all of them are shorter than the accuracy provided for by the application logs. These very short sessions are also indicated in the minimum session sizes, and they correspond to a session containing only a handshake or an interrupted handshake.

Another pertinent feature is the quota of the number of unique IPs to the number of unique peers for the case of measurement 8. The quota for this measurement is observed to be slightly above 0.25, while none of the other measurements are below 0.5. This indicates that either users are stopping and restarting their clients several times, or that users are sharing IPs, such as peers behind a NAT box.

#	Sessions	Session length (s)				Session size (MB)				Peers[a]	
		Mean	Max	Min	Std	Mean	Max	Min[b]	Std	ID	IP
1	29712	343	98991	0	2741	27.49	647.26	73	70.65	2024	1314
2	46022	233	117605	0	2316	27.15	646.03	73	64.05	1876	1394
3	28687	465	171074	0	3614	28.54	539.20	73	61.70	1913	1319
4	13493	750	143707	0	3942	49.88	671.99	73	100.65	1813	1143
5	12354	910	180298	0	4504	57.08	668.53	73	116.10	1747	962
6	10685	1207	223235	0	7016	74.25	3117.79	73	247.74	1033	619
7	4444	218	46478	0	1642	49.96	431.13	78	76.48	279	184
8	17287	231	87026	0	1972	33.11	695.94	73	109.31	1656	406
9	3043	294	29163	0	1719	21.62	408.05	78	42.27	193	166
10	9701	652	267497	0	5907	37.78	1499.85	73	109.08	444	305
11	43939	448	141509	0	3791	17.22	475.86	73	52.73	1841	1067
12	68288	197	292241	0	2580	8.31	987.89	73	30.63	2177	1152

Table 4: Summary of sessions and peers

[a]Unique peer client IDs and IP addresses
[b]This column measured in bytes.

Table 5 summarizes the number of messages received on a per-message basis. In addition, column 5 shows the number of the incoming connection requests collected in our measurements.

The *request* and *have* messages clearly dominate in terms of number of messages sent, while the *interested* and *not interested* messages are the least common ones. This is observed to be valid for all measurements, except for measurement 2, which has almost 5 times more incoming *interested* messages than the measurement with the second highest number of *interested* messages.

The high number of *request* and *have* messages found in our measurements is an expected result, as the peer is acting as a seed for most of the time spent in the swarm. When seeding, a peer never receives *piece* messages, and the downloading peer must request data by the *request* message. The *have* messages are accounted for by the fact that every completed piece download results in such a message being transmitted.

#	request	not int.	piece	new conn.	bitfield	unchoke	have	int.	choke	cancel
1	3316470	504	135615	29746	28024	27120	3651835	2905	26314	6500
2	3044768	489	135797	46047	45054	19117	3984881	14602	18061	9059
3	3276644	493	135682	28714	27092	40705	3941658	2430	39955	7628
4	5596270	406	40167	13502	12935	29628	1206000	2041	28640	14643
5	6163605	401	42176	12364	11827	32325	1197813	2059	31452	11508
6	4501907	191	277261	10688	9659	24239	2090892	2147	23639	6244
7	810019	52	40371	4445	4370	290	198885	230	122	1255
8	3347256	766	44328	17292	16623	9270	404038	2012	8579	18999
9	217336	37	40426	3045	2996	1114	139472	259	956	3061
10	838379	79	268429	9703	9181	13015	570367	692	11936	9085
11	1835910	470	268575	43957	42848	54090	4713440	2573	52458	17313
12	1118110	348	139943	68297	67373	37925	2619333	3242	36872	25047

Table 5: Summary of downstream protocol messages

The summary of the outgoing messages reported in the Table 6 again shows the very low number of *interested* and *not interested* messages. The major bulk of the outgoing messages is however accounted for by the *piece* messages. This is again an expected result, as *request* messages generate a *piece* message in response. The absence

of transmitted *choke* messages for the measurement 7 indicates that there has been a continuous exchange of data between peers. Regarding the *request* and *have* messages, these are tightly coupled to the number of pieces present in the content. The higher number of *request* messages is explained by the fact that these messages correspond to only a single subpiece.

#	request	piece	not int.	unchoke	bitfield	int.	have	choke	cancel
1	137007	3251948	63	11792	29714	68	8465	9553	970
2	137271	2964836	63	17471	46020	70	8465	13301	894
3	136738	3189175	62	16545	28682	64	8465	14085	1011
4	42709	5468908	76	25476	13489	86	2501	22740	855
5	44862	6032599	146	25759	12353	157	2627	23749	725
6	291200	4394389	91	23166	10661	197	16708	18943	555
7	40497	808844	18	4445	4444	18	2522	0	140
8	47413	3296616	100	19380	17281	136	2753	8672	423
9	40906	213693	16	3192	3042	19	2522	193	220
10	285650	753074	71	21304	9673	214	16708	15222	611
11	281921	1660868	67	35698	43927	157	16673	31279	812
12	145517	960802	76	49093	68271	125	8719	34570	701

Table 6: Summary of upstream protocol messages

Finally, Figure 4 shows an example of evolution of the swarm size obtained in the measurement 2. It is observed for instance in the leech figure the short download time and high download rate that are typical for the swarm. By quickly connecting to the preconfigured number of peers, the client instantly has a large number of peers from which to download, increasing thus the probability of fast downloading. For the seeding phase, the number of connected peers seems to display a daily variation pattern for this measurement.

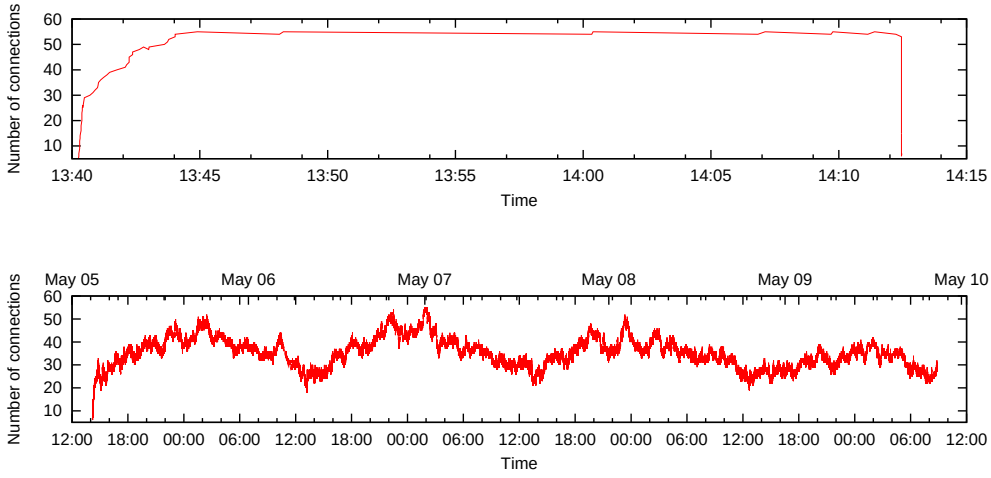

Figure 4: Swarm size evolution for leech phase (up) and seed phase (down)

6 Conclusions

A measurement study of BitTorrent signaling traffic has been reported for traffic collected at the Blekinge Institute of Technology (BIT), Karlskrona, Sweden. The characteristics of the signaling traffic exchanged among the participating peers in a BitTorrent distribution swarm have been investigated. A specific approach has been used for traffic measurements that is based on combining instrumentation at the application layer with flow identification and extraction at the transport level.

Our future work will be about further analysis of the obtained results, to find out structural similarities or differences among characteristics of peers and swarms as well as to search for possible invariant characteristics across object flows.

References

[1] Blumenthal M. S. and Clark D. C., *Rethinking the Design of the Internet: the End-to-End Arguments vs. the Brave New World*, ACM Transactions on Internet Technology (TOIT), Vol. 1, No. 1, August 2001.

[2] Cohen B., *Incentives Build Robustness in BitTorrent* , http://bitconjurer.org/BitTorrent

[3] Erman D., Ilie D. and Popescu A., *Peer-to-Peer Traffic Measurements*, Technical Report, Blekinge Institute of Technology, 2004.

[4] Izal M., Urvoy-Keller G., Biersack E. W., Felber P. A., Al Hamra A. and Garces-Erice L., *Dissecting BitTorrent: Five Months in a Torrent's Lifetime*, 5th Passive and Active Measurement Workshop, PAM2004, Antibes, France, 2004.

[5] Jacobson V. et al, *tcpdump* software, ftp://ee.lbl.gov, June 1989.

[6] Karagiannis T., Broido A., Brownlee N., Claffy K. and Faloustos M., *File-Sharing in the Internet: A characterization of P2P traffic in the backbone*, technical report University of California, Riverside, USA, 2003.

[7] Ostermann S., *TCPTRACE: A TCP connection analysis tool* , http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html

[8] 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.

[9] Saroiu S., Gummadi K. P. and Gribble S. D., *Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts*, Multimedia Systems Journal, Vol. 9, No. 2, August 2003.

[10] Sen S. and Wang J., *Analyzing Peer-To-Peer Traffic Across Large Networks*, IEEE/ACM Transactions on Networking, Vol. 12, No. 2, April 2004.

[11] van Rossum G. et al, *Python*, http://www.python.org