Simulating BitTorrent

Karel De Vogeleer
Blekinge Institute of Technology
kdv@bth.se

David Erman
Blekinge Institute of Technology
david.erman@bth.se

Adrian Popescu
Blekinge Institute of Technology
adrian.popescu@bth.se

ABSTRACT

IP Television (IPTV) and other media distribution applications are expected to be one of the next Internet killer applications. One indication of this is the corporate backing that the IP Multimedia Subsystem (IMS) is getting. However, the bandwidth utilization of these applications is still an issue, as the volume of multimedia grows due to larger image resolution and higher bitrate audio. One way of managing this increase in bandwidth requirements is to use existing end-host bandwidth to decrease the load on the content server in a Peer-to-Peer (P2P) fashion. One of the most successful P2P applications is BitTorrent (BT), a swarming file transfer system. This paper presents an implementation of a BT simulator intended for future use in investigating modifications to the BT system to provide streaming media capabilities. The simulator is validated against real-world measurements and a brief performance analysis is provided.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

Keywords

P2P, simulation, BitTorrent, streaming, multimedia, video-on-demand

1. INTRODUCTION

The advent of P2P applications has instigated a significant amount of research into many aspects of these applications. A large number of measurement, analytical and simulation studies have been performed on a wide variety of P2P applications and protocols. P2P applications appear in a multitude of forms, but the most common is some form of file sharing or resource location application. These applications have shown to be very efficient in both decreasing server load as well as providing a simple way of locating information. P2P applications impose their own routing and forwarding on top of the Internet infrastructure, in effect forming overlay networks.

While P2P applications are the de-facto most prominent contributors to Internet traffic today, there is another group of applications expected to challenge this situation. This group consists of IPTV, Voice over IP (VoIP), Video-on-Demand (VoD) and other multimedia distribution applications. By the amount of attention placed upon it, large-scale multimedia streaming is expected to be the next killer application. Indications of this are the corporate backing that, *e. g.*, the IMS has achieved as well as the amount of research put into mechanisms for compression, coding, distribution and related domains. Previous attempts to provide streaming services using IP Multicast (IPMC) have to a large extent been unsuccessful. This is due to a number of reasons: IPMC lacks buffering capabilities, a lack of applications using IPMC, poor deployment of IPMC and reluctance of Internet Service Providers (ISPs) to allow IPMC forwarding across their networks. Overlay Multicast (OLMC) has been investigated as a possible solution to these issues. Systems such as End-System Multicast (ESM) [19], PeerCast [16], dPAM [18], oStream [7] and others have been proposed to address the issues with IPMC in the context of media streaming.

One of the most popular P2P applications is BT [6], a swarming file distribution system. BT has shown to be very efficient in distributing large files without increasing the load on the original server of the files being distributed. While originally a file distribution system, research has been done on adapting the core BT algorithms to facilitate VoD. Solutions such as BASS [8] and BiToS [21] have been proposed that modify the original BT algorithms to be able to handle streaming scenarios. Streaming media places significantly different demands on both the network and the distribution mechanism from those of non-streaming file transfers. The most obvious difference are the temporal demands, *i. e.*, delay sensitivity and the associated timeliness of data packet arrival. In addition to packet timeliness, the ordering of the packets is important. The data of a video object is played back in a linear fashion, but in BT, data segments are downloaded in a random fashion. As for the delay demands of VoD, it is very difficult to provide any true Quality of Service (QoS) for these demands, and the BT protocol does not provide any means for this. The work we present in this paper is part of the Routing in Overlay Networks (ROVER) project at Blekinge Institute of Technology (BTH) to rectify

QoSim 2008, March 3, 2008, Marseille, France.

these issues with BT, so as to make it a viable and cost-effective solution for VoD. While testing on real and active BT networks would be beneficial, the risk of adversely affecting the network during development of our modifications is too high. Thus, we have designed a simulator in which the core BT algorithms can easily be exchanged for alternative algorithms. Which is used for investigating modifications to the BT system to provide streaming media capabilities [10].

The rest of this paper is organised as follows. In section 2, we briefly describe the BT system. This is followed by a survey of related work on P2P simulators and simulation frameworks in section 3. In section 4, we describe our BT simulator, and present some results regarding the validation of the simulator against real-world traces of BT traffic. The paper is concluded with a discussion of future work in section 5.

2. BITTORRENT

BT is a swarming distribution system originally developed to distribute large files. The system comprises two network entity types: *peers* and *trackers*. A peer is an end host participating in the distribution of a specific set of data, and we denote a local peer a *client*. Peers can be either *seeds* or *leechers*. A seed is a peer that possesses all the pieces of the data, while a leecher is a downloading peer. A tracker is a peer arbitrator that keeps track of what peers are currently participating in a swarm. The trackers and set of peers participating in distribution are collectively known as a *swarm*.

BT employs a fairness system designed to provide incentives for peers to reciprocate up- and downloads between each other. Data is referenced in fixed-size byte ranges, called *pieces*. Requests for data among peers are made in parts of a piece, known as *blocks*. A piece is thus made up of several blocks. Once a client has downloaded all blocks of a piece correctly, all peers currently connected to are notified of this. This informs the peers that the piece is now available at the client.

The characteristics of BT are primarily governed by two main algorithms: the piece selection algorithm and the peer selection (or *choking*) algorithm. A peer may run any version of the algorithm, as the protocol does not dictate any semantics regarding these.

The piece selection algorithm in the reference BT client employs four distinct policies when deciding what block to request next. When at least one block of any given piece is downloaded, the remaining blocks from this piece are prioritized.[1] This policy is known as the *strict priority* policy. When fewer then four pieces have been downloaded, and there are no active pieces, a random piece is selected for download. This policy is known as the *random-first* policy. When at least four pieces are downloaded, the least replicated piece in the swarm, *i. e.,* the piece that is available at the fewest peers is selected for download. This policy is known as the *rarest-first* policy. When every piece has been requested or downloaded, *i. e.,* when the only active policy is the strict priority policy, the remaining blocks are requested from all connected peers and are downloaded from the first responding peer. This policy is known as the *end-game mode*.

[1]A piece with at least one block downloaded is known as an *active* piece.

In order to have a fair level of upload and download reciprocation, the *choking* algorithm was introduced in BT. Two states are assigned to each peer: *choked* by a peer and *interested* in a peer. When a peer is *choked* it will not be allowed to download from the peer it is choked by. A peer is *interested* in another peer when the latter peer has pieces that the former doesn't have. The choking algorithm differs in leecher and seeder state. However, in both cases there can only be 4 peers unchoked by a peer and interested in that specific peer at the same time. Every peer starts as *choked* by a peer. The policy to get unchoked by a leecher is as follows. Every 10 seconds, the peers that are interested in the leecher are ordered according to their download rate to the leecher. The 3 fastest peers are unchoked by the leecher. The remaining peers that are unchoked are choked. Additionally, every 30 seconds another random interested peer gets unchoked. The latter gives new peers, that don't have any pieces yet, the possibility to start their first download. The choking policy in seeder state is as follows. Every 10 seconds, the unchoked peers are ordered according to the time they were last unchoked. The first two consecutive periods of 10 seconds the fourth most recent unchoked peer is choked and an additional random choked and interested peer is unchoked. The third period of 10 seconds the 4 most recent unchoked peers are kept unchoked. This way each leecher in the swarm gets the opportunity to download from a seeder.

More detailed descriptions of the BT system are available in [5, 6, 9].

3. RELATED WORK

The popularity of P2P applications has resulted in a large amount of research activities in terms of developed simulators. These simulators range from general frameworks to application specific. A brief overview of P2P simulators is presented below. For a more complete review, the work in [15] is more comprehensive.

An ambitious general framework is OverSim [1], which extends the OMNeT++ [20] simulation framework. OverSim provides a modular system for implementing both structured and unstructured P2P protocols as well as various types of underlays, *i. e.,* transport protocols such as TCP or UDP. Depending on the selected underlay, simulations can be made on realistic scenarios when needed, but be run without considering, *e. g.,* link delay or transport protocol details, when simulation speed is more important. The OverSim framework also supports connecting real-world applications to the simulation engine. Several overlay protocols such as Chord, Pastry and GIA are provided with the OverSim distribution, and several more are planned for implementation.

PeerSim [12] is a Java P2P simulator comprising two distinct simulation engines. One is an efficient cycle-based engine, which does not take into account many parameters in the protocol stack. The other is a more accurate event-based engine, and thus significantly slower but allows for more realistic simulations.

As PeerSim, GPS [22] is another Java simulator. Also, as OMNeT++, it is an event-driven, message-oriented simulator. GPS incorporates a GUI and logging features, in addition to the core simulation components. The simulator has been used to model the BT protocols, primarily the peer protocol, also known as the peer wire protocol. The authors

also report on running the reference client in a small-scale LAN topology and compare measurements from this scenario with an identical simulation scenario. The results indicate some similarities between the measurement and simulation, but nothing conclusive. Additionally, the authors present a scalability analysis of the simulator, which indicates that the wall-clock simulation time increases exponentially with the number of simulated nodes.

One of the earliest attempts to simulate BT-like scenarios is the swarming simulator described in [13]. The simulator does not implement the full BT protocol, and development seems to have stopped. Additionally, the simulator abstracts the BT network entities in a rather unorthodox way, making extending the simulator more complex and difficult.

Another BT-specific simulator is the one used for the work presented in [2] and [3]. It is written in the C# language and implements the majority of the BT mechanisms. Being implemented in the C# language and making use of the Microsoft .NET runtime makes platform independence an issue, and, as with [13], development of the simulator seems to largely have stopped.

4. A BITTORRENT SIMULATOR

In this section, we describe the BT simulator developed as part of the ROVER project at BTH. We discuss the motivation for writing another simulator as well as provide a short description of the simulator itself.

4.1 OMNeT++

Our BitTorrent simulator makes use of the OMNeT++ framework [20]. OMNeT++ is a public-source, component-based, modular, open-architecture discrete event simulation environment. Simple modules are written in C++ and defined in the NED-language. NED is a simple language developed to easily insert modules into the simulator. Modules interact with each other by means of messages sent through gates and over channels. Figure 1 shows a visualisation of a simulation run. The OMNeT++ GUI makes it intuitive to debug and validate simulation scenarios. The simulation executable can also be compiled as a command-line variant to get higher simulation speeds. Statistics are conveniently managed through specific classes provided by the OMNeT++ framework designed to collect simulation data.

The modular architecture and structure of the OMNeT++ framework makes it quick and easy to extend the core BT algorithms. Furthermore, the fact that OMNeT++ is written in C++ has the added advantage of platform independence, and allows us to run simulations on a wide variety of operating systems and architectures. Additionally, OMNeT++ is free for academic and non-profit use and has proven useful success in other projects, *e. g.*, OverSim [1].

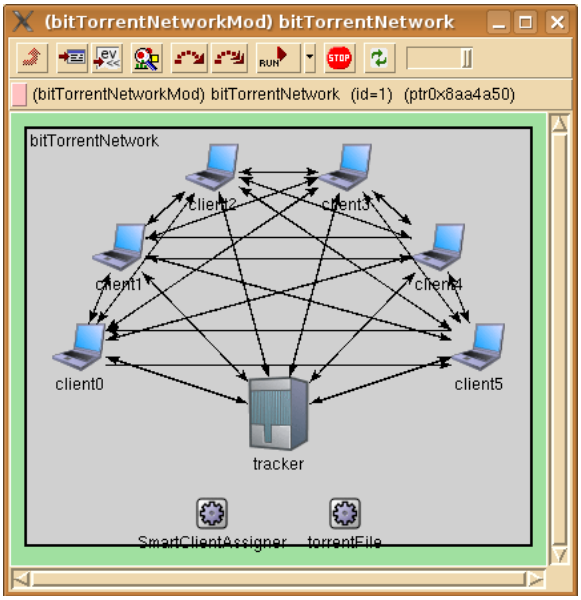


Figure 1: The BitTorrent simulator showing a simulation with 6 peers.

4.2 Design

The BT simulator is comprised of four distinct modules. The *TorrentFile* module loads data from a real *.torrent*-file and makes it available for the other modules. This data is primarily the file size and piece length. The *ClientAssigner* module creates the necessary amount of BT clients at the start-up of the simulation by dynamic module creation. It assigns the *seeder*/*leecher* states, session arrival and seeding time to each BT client. The two remaining modules are the *BTClient* and *BTTracker* modules, representing the corresponding BT entities.

A large number of parameters are accessible through an external initialisation file, loaded during the simulation start-up. These parameters are:

- amount of clients that join the swarm during the simulation run.
- amount of seeders initially present in the swarm; these stay present during the whole simulation.
- session interarrival time, the time between arrivals of successive BT clients.
- seeding time of the clients when they turn into seeds. This implies that a client only leaves the swarm after it is turned into a seeder.
- delay and bandwidth of the links between nodes in the swarm.
- an indicator whether links between nodes in the swarm are asymmetrical or not.
- desired piece selection algorithm that the BT clients execute.
- number of peers a client wants to receive when exchanging information with the tracker.

- maximum number of active connections between nodes in the swarm at any time. This represents the maximum amount of TCP connections at each client.
- amount of pieces a client has when it joins the swarm.
- time between the arrival of a piece request and the actual transmission of that piece.
- block length.

Any of these parameters can be drawn from an arbitrary distribution available through the OMNeT++ framework.

The simulation ends when every scheduled BT Client has turned into a seeder. This is only possible if there is always at least one seeder present in the swarm. This is also a verification of whether the simulator works properly. If the simulation does not end automatically, it means that certain leechers could not download all the pieces of the content.

4.3 Implementation

The implementation of the BT protocol is based upon the *mainline* client [4], version 4.0.2 released in May 2005. The *mainline* client is considered the reference implementation of the BT protocol. As there exist many different implementations and extensions of BT, we have restricted our implementation to this client. The choking- and rarest first algorithm are implemented as they were presented in [14], which are the algorithms used in the *mainline* client v4.0.2. The BT protocol itself is rather simple, but the associated algorithms can be implemented in many different ways. Therefore, some implementation patterns must be highlighted.

All messages are responded immediately. This implies that the processing time for a message is zero (in simulation time), except for piece requests, which have a response delay. This delay is settable in the initialisation file. Consequently, a leecher starts downloading from another peer at the moment it is unchoked by that specific peer. A new block is requested immediately after a block has arrived from a peer, provided that the client is not choked in the mean time by that peer. Handshakes and bitfields are also exchanged without processing delay. As piece requests are the most occurring messages needing a response, it is a fair approximation to only use response delay when handling piece requests. This response delay is also necessary to implement the end-game mode.

As the simulator was written to be easily extendable for piece selection algorithms, it is highly desirable to insert an easy to use mechanism to select a piece selection algorithm. This is realised by means of a parameter setting accessible in the initialisation file. A new algorithm must be added to the source code and the internal simulator code then selects the proper algorithm.

The built-in topology generator makes it easy to simulate different swarm sizes by changing only one parameter in the initialisation file. The BT clients are then created dynamically upon start-up of the simulation. The maximum swarm size of a simulation is not explicitly defined. However, the swarm size can be altered by two parameters: the amount of clients connecting during the simulation run and the session interarrival time.

4.4 Simulation Results

In this section, we discuss some preliminary simulation results achieved with the simulator.

4.4.1 Parameters

As most of the parameters for the simulation have not been measured before, we make assumptions regarding the distributions. For instance, the exponential seeding time and symmetric link assumptions are also used in [17]. The parameters for the simulations presented in this section were set as follows:

- The swarm interarrival time distribution is an exponential distribution with a mean of 180 s.
- The seeding time distribution is also an exponential distribution with mean 180 s.
- Each peer requests 10 peers from the tracker.
- The piece response time distribution is an exponential distribution with mean 100 ms.
- The initial piece distribution per peer is a uniform distribution, with values between 0 % and 50 %.
- Links between peers are assumed to be symmetric, and the link delay distribution is a uniform distribution, with values between 50 ms and 400 ms. The link bandwidth distribution is also the uniform distribution, with values between 2 Mbps and 10 Mbps.
- We ran 8 sets of experiments, with one initial seed and 5, 10, 50, 100, 500, 1000, 2000 and 3000 peers, respectively. Each simulation was run once.

The simulator is able to handle large swarm sizes. The major boundary when computing these is the physical memory of the device used for simulation. A BT client in the simulator occupies around $\approx 2{,}5$ kB of memory space. This number depends on the simulation settings, mainly the maximum amount of connections. Thus, a network size of 5000 nodes minimally needs a memory space larger than 12,5 MB plus additional memory space to store swarm information at each peer and the OMNeT++ simulation core. Consequently, it is possible to run large simulations on a device with an average amount of resources. Studies have shown that a real-world swarm usually doesn't exceed a swarm size of 3-500 nodes except during flash crowds [11]. During flash crowds, peer numbers are one order of magnitude larger. However, the swarm size parameter in our simulator doesn't refer to the actual instantaneous swarm size but rather the total number of peers connecting during the entire life time of the swarm.

Figure 2 shows the simulation time as a function of the number of clients connecting to the swarm. The course of the curve tend to be linear, which means that the additional computational overhead of adding a new peer during runtime is low for the specific torrent file and parameter set.

4.5 Simulator Validation

To validate the simulator's behaviour we compare simulation data with real-world measurements.

4.5.1 Piece selection

Ideally, the rarest-first piece selection algorithm should tend toward a uniform piece distribution of pieces. That is, each piece should be replicated about the same number of times in the swarm. This means that each piece will also be *requested* about the same number of times.

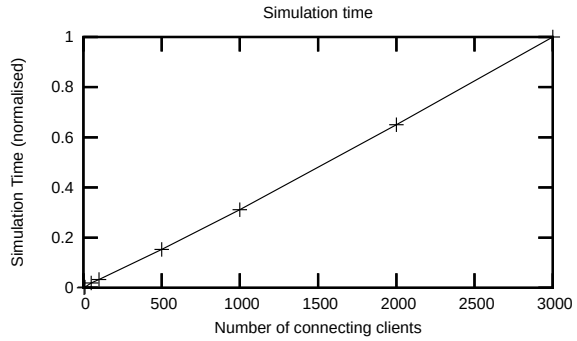


Figure 2: Evolution of the simulation time as a function of the amount of connecting nodes during a simulation run.

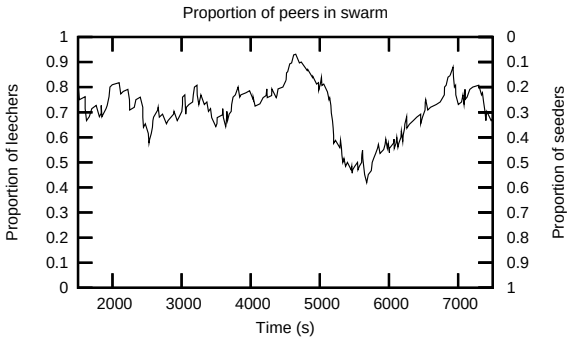


Figure 4: Simulation data representing the proportion of seeders and leechers in the swarm during a simulation run.

In Figure 3, we show the Empirical Probability Density Function (EPDF) for the piece distribution for a simulation scenario with 1000 peers. The grey line is the theoretical density for a uniform distribution with minimum value 0 and maximum value 1204, while the black line represents the proportion of requests of the associated piece.

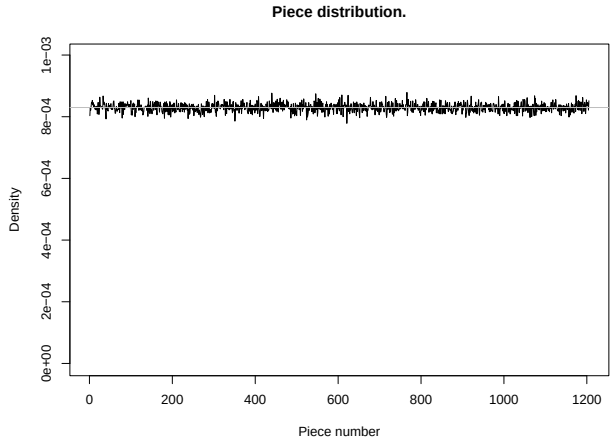


Figure 3: Piece distribution for 1000 peer scenario.

As Figure 3 shows, the simulated piece selection algorithm quite clearly tends toward a uniform selection of pieces.

4.5.2 Choking algorithm

The proper working of the choking algorithm was checked by tracing what decisions each peer made while running the choking algorithm. No irregularities were observed during our simulation test-runs.

4.5.3 Number of peers in swarm

Figure 4 shows the amount of seeders and leechers in the swarm, taken from a simulation run. There are no noticeable errors observed in this plot, and the results shows similarities with the corresponding results in [11].

4.5.4 Session duration

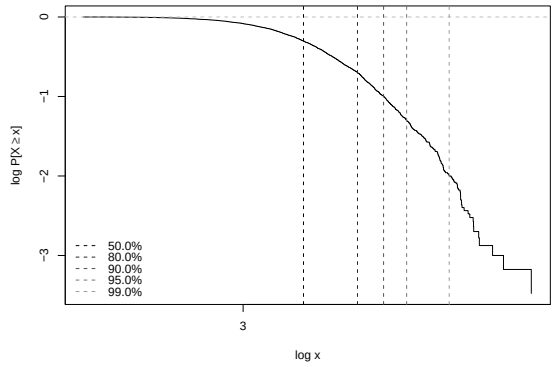
Realistic session arrivals and session duration time are fundamental for a realistic simulation. The session arrival rate is set in the initialisation file. The session duration time, however, is partially dependant on the amount of peers in the swarm and the initial amount of pieces of the peers as well as the piece response delay and available bandwidth. In order to be able to compare similar data of simulation results and real-world measurements we achieved a simulated swarm size similar to the one observed in the real-world measurements [9].

Figure 5(a) shows a graph of the session duration Complementary Cumulative Distribution Function (CCDF) of all peers for simulation run 8, *i. e.*, with 3000 peers in total. Figure 5(b) shows the session duration for one of the measurements reported in [9]. When fitting a log-normal distribution to the simulated results, the Anderson-Darling (AD) statistic for the fitted parameters of the data is 0.89, indicating a very good measure of fit. This is an interesting result, as in [9], results indicated that session durations are log-normally distributed. This validates the functionality of our simulator with regards to session duration.

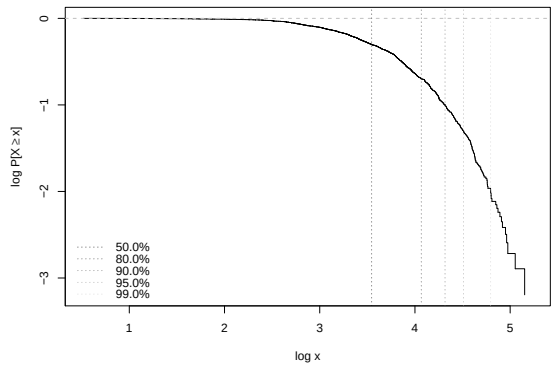
5. FUTURE WORK

While our simulator is capable of handling a wide range of scenarios, there is still some functionality missing. For instance, modular peer selection is not implemented, nor is peer snubbing, *i. e.*, dropping peers that do not respond quickly enough. Also, as the BT protocol allows for extensions, several clients have used this capability to add new features to the protocol, *e. g.*, a trackerless Distributed Hash Table (DHT) protocol, encryption, super seeding. Additionally, some clients have extended the torrent file format to contain more information, such as current download status, connected peers and QoS information. None of these non-standard extensions have been implemented in our simulator. In the near future, we plan to add both snubbing and modular peer selection capabilities to our simulator, as well as extend the protocol with QoS-related messaging.

Furthermore, points of improvement can be carried out to improve the performance of the BT simulator. The current version of the simulator loads all the BT clients that will connect during the simulation at the simulation start-up. We will change this to loading each client when the session arrives in simulation time. This way we will be able to simulate larger and longer simulations. Further, an overloadable piece selection algorithm would be preferred instead of a sim-



(a) Simulated results.



(b) Measured result.

Figure 5: BT session duration.

ple hard-coded switch. Also, the *TorrentFile* module needs improvement so it can load more *.torrent*-files.

Additionally, we will run more and longer simulation runs, as well as repeat runs with varying initial random number seeds to achieve confidence intervals for the simulation results.

6. ACKNOWLEDGMENTS

We would like to thank the Swedish Internet Infrastructure Foundation (IIS) and Euro-NGI for granting and supporting the ROVER project during 2006 and 2007.

7. REFERENCES

[1] I. Baumgart, B. Heep, and S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium*, pages 79–84, May 2007.

[2] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Some observations on bittorrent performance. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 398–399, New York, NY, USA, 2005. ACM Press.

[3] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent network's performance mechanisms. In *INFOCOM 2006. Proceedings of the 25th IEEE International Conference on Computer Communications.*, pages 1–12, 2006.

[4] BitTorrent, Inc. Bittorrent. http:www.bittorrent.com. URL verified on April 10, 2007.

[5] B. Cohen. BitTorrent protocol specification. http://www.bitconjurer.org/BitTorrent/protocol.html, February 2005.

[6] B. Cohen. BitTorrent. http://www.bittorrent.com/, March 2006.

[7] Y. Cui, B. Li, and K. Nahrstedt. oStream: Asynchronous streaming multicast in application-layer overlay networks, 2003.

[8] C. Dana, D. Li, D. Harrison, and C.-N. Chuah. BASS: Bittorrent assisted streaming system for video-on-demand. In *Proceedings of IEEE 7th Workshop on Multimedia Signal Processing*, pages 1–4, October 2005.

[9] D. Erman. Bittorrent traffic measurements and models, October 2005. Licentiate thesis, Blekinge Institute of Technology.

[10] D. Erman. Extending bittorrent for streaming applications. In *Proceedings of the 4th Euro-FGI workshop on "New Trends in Modelling, Quantitative Methods and Measurements"*, May/June 2007.

[11] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. In *Passive and Active Measurements (PAM2004)*, 2004.

[12] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. Peersim. http://peersim.sourceforge.net, October 2007.

[13] P. Korathota. Investigation of swarming content delivery systems. Master's thesis, Sydney University of Technology, http://me55enger.net/swarm/thesis.pdf, November 2003.

[14] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *ACM SIGCOMM/USENIX ICM'2006*, October 2006.

[15] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37(2):95–98, 2007.

[16] PeerCast.org. Peercast – p2p casting for everyone. Online at http://peercast.org. URL verified on March 27, 2007.

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

[18] A. Sharma, A. Bestavros, and I. Matta. dPAM: A Distributed Prefetching Protocol for Scalable Asynchronous Multicast in P2P Systems. In *Proceedings of Infocom'05: The IEEE International Conference on Computer Communication*, Miami, Florida, March 2005.

[19] The ESM Project. ESM – end system multicast. Online at http://esm.cs.cmu.edu/. URL verified on March 26, 2007.

[20] A. Vargas. OMNeT++ discrete event simulation system. http://www.omnetpp.org.

[21] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for supporting streaming applications. In *9th IEEE Global Internet Symposium (GI2006)*, Barcelona, Spain, April 2006.

[22] W. Yang and N. Abu-Ghazaleh. Gps: A general peer-to-peer simulator and its use for modeling bittorrent. *mascots*, 00:425–434, 2005.