Efficient Simulation of Large-Scale P2P Networks: Packet-level vs. Flow-level Simulations

Kolja Eger[*]
Hamburg University of Technology (TUHH)
Institute of Communication Networks
Germany

Tobias Hoßfeld, Andreas Binzenhöfer
University of Würzburg
Institute of Computer Science
Germany

Gerald Kunzmann
Technical University of Munich
Institute of Communication Networks
Germany

Abstract

Peer-to-peer (P2P) networks can reduce the distribution cost of large media files for the original provider of the data significantly. Thereby, the BitTorrent protocol is widely used in the Internet today. Most research work studies the protocol analytically, by simulations at the flow-level or real world experiments. Thereby, for flow-level simulations the influence of neglecting packet-level characteristics is not yet quantified. Therefore, this paper compares packet-level simulation results with flow-level values and analytically derived bounds. Our findings show that BitTorrent is near to optimal at flow-level for different scenarios. Naturally, packet-level results deviate more from the optimal values but differences are at most around 30% in our simulations. Furthermore, we show that the propagation delay can significantly influence the download performance of BitTorrent.

1 Introduction

The Peer-to-Peer (P2P) paradigm offers obvious advantages for the fast distribution of large content in the Internet. While in the client/server architecture the total load must be carried by the server(s), it is distributed among the users in a P2P network.
A popular example is the BitTorrent protocol [7]. Basically, with two simple ideas this protocol increases efficiency and reduces free-riding, which was a severe problem in the first P2P networks [3]. Firstly, a file is fragmented into a number of smaller pieces, called chunks. When a peer completes the download of a chunk, it can already upload it to other peers. Secondly, each peer controls to whom it uploads data. This is called choking/unchoking in BitTorrent (see Section 2 for details).
BitTorrent networks cause large amounts of traffic in today's Internet and have gained much interest in research and practice. Three types of approaches can be found frequently in the literature to analyze its performance. These are analytical studies like [16], simulations at the application-layer [5] and the analysis of real traces [12]. Whereas analytical models are based on simplifying assumptions, simulations at the application-layer take only the access link bandwidth into account and model the data transfers as flows. These flow-level simulations overestimate the performance of BitTorrent because characteristics of the packet-level are ignored. Real world experiments suffer from the lack of knowledge about the properties of all peers in the network.
Because of the high simulation complexity packet-level simulations are rarely conducted for P2P networks. However, for BitTorrent the number of active peers is much smaller than in other P2P networks, because only one file is shared per overlay network. By looking at statistics for popular torrents we observed up to 10000 peers per overlay network for very popular data. But this number decreases sharply to 1000 peers and below for less demanded data. Thus, medium-sized BitTorrent networks can also be simulated at packet-level.
Packet-level simulations are not only helpful to validate simplified simulation models but also to study the influence of the lower level protocols on the BitTorrent performance. Since data in BitTorrent is transferred over TCP, one of the reasons to simulate at packet-level is to take the exact behavior of TCP into account. The influence of TCP

[*]Corresponding author: eger@tuhh.de

on BitTorrent's unchoking algorithm can be notable: TCP throughput depends, among others, on the round trip time (RTT) between the peers. Thus, the achieved throughput increases with decreasing RTT. So it can be reasoned that a peer unchokes other peers, which are near to it with respect to the delay between peers. This means that the performance of a peer does not only depend on its upload capacity, but also on the RTT to other peers. To prove or disprove such cross-layer interactions packet-based simulations are inevitable. Another reason for packet-level simulations is to study the traffic aggregation of P2P applications at the edge and core routers of a network for developing and validating new traffic models which incorporate P2P traffic.

To the best of our knowledge this paper presents the first simulation results at packet-level for BitTorrent-like P2P networks. Further contributions are the comparison to analytical results as well as simulations on flow-level for two scenarios (flash crowds and networks with constant number of peers). Furthermore, we show that the download performance of a peer depends on the delay to other peers.

The paper is structured as follows. Section 2 presents aspects of the BitTorrent protocol which are relevant for this paper. Details about the simulation methodology used in this paper are presented in Section 3. Simulation results are discussed in Section 4. Section 5 concludes the paper.

2 BitTorrent Protocol

The BitTorrent protocol [7] was implemented with the objective to disseminate one large file (or a composition of large files) to a large number of users in an efficient way. Therefore, for each file an overlay network is created. The file sharing is based on the swarming principle, which is also denoted as multi-source download. Here, the file of interest is fragmented into chunks. When a peer completes the download of a single chunk, it offers it to other peers which so far have not downloaded this chunk. Thus, peers exchange chunks with each other although they did not finish the download of the complete file. Therefore, the resources in the P2P network are used more efficiently and the network also scales for large peer populations with respect to download time.

According to the original BitTorrent specification, each overlay network consists of two different kinds of peers, the seeds and the leechers, and a so-called tracker. A seed holds the complete file and uploads to others altruistically, whereas a leecher is still downloading the file. The tracker is a centralized component which stores information about all peers. A new peer, which enters the network, asks the tracker for a list of active peers in the overlay. The tracker returns a random subset to the requesting peer. Furthermore, an active peer contacts the tracker from time to time to obtain information about new peers in the network. An extension [2] of the protocol also incorporates the exchange of information about other peers in the network between connected peers. This is often stated as trackerless BitTorrent. This paper focuses on the original version with a tracker.

BitTorrent specifies the messages between the tracker and a peer and between peers themselves. Furthermore, it implements two important algorithms which are run by each peer. These are the peer selection and the piece selection algorithm. The peer selection process is called choking/unchoking in BitTorrent. Each peer controls to whom it uploads data. When a remote peer is selected for upload an *UNCHOKE* message is sent. A upload is stopped with a *CHOKE* message. Each peer uploads to a fixed number of other peers (the default value is four). Thereby, a peer chooses to upload to other peers from which it has the highest download rates. As a seed unchoking is based on the download rate of the connected peers rather than the upload rate. By default this tit-for-tat strategy is run every ten seconds by every peer, whereby the download rates are determined by a moving average over the last 20 seconds. To discover new peers with better performance a so-called optimistic unchoke is done additionally. Here, one of the peers is unchoked independently of its rate. The optimistic unchoke is changed every 30 seconds to provide enough time to be possibly unchoked by the remote peer in return. Another rule in BitTorrent is to choke a peer when it has sent no data message in the last minute. This is called anti-snubbing.

The piece selection algorithm determines which file fragment is requested when a peer is unchoked by a remote peer. The decision process follows the following rules: Firstly, when some bytes are received from a specific chunk the remaining parts of that chunk are requested. This scheme is called strict priority. Since peers forward only complete blocks (where data integrity is verified) to other peers, this mechanism ensures that blocks are completed fast. When strict priority is not applicable, the rarest block is requested. Since a peer has only a local view of the network it can only estimate rarity based on the chunk information of its neighbors. These information are available to the peer by the BitTorrent messages *BITFIELD* and *HAVE* (see [1] for details).

When a peer has no chunk at the beginning of the download, BitTorrent deviates from the rarest-first policy and the new peer requests a block randomly. This is intended to ensure a faster completion of the first block such that the upload bandwidth of that peer can be used by others.

Normally, one *REQUEST* message asks for a data portion which is smaller than the chunk size. The default values in the original implementation are 256 KB as chunk size and 16 KB per request. To prevent that the sender runs out of requests and has to wait for a new request from another

peer, the first requests after an unchoke are sent as a batch. By default the batch size is 5 requests. In normal mode a peer requests each part only once. This can become a problem at the end of the download. When the rest of the file is requested at a very slow peer, the downloading peer has to wait long although other peers may handle the request faster. Therefore, a peer can switch to the endgame mode, where it requests the same parts at multiple peers. Although, a peer can cancel requests at remote peers the endgame mode can consume additional bandwidth by transferring redundant parts.
The BitTorrent protocol is neither standardized nor fixed and a large number of different applications, which use the BitTorrent protocol, are available. Especially, the implementation of the peer and the piece selection is implemented in different ways, because it is not part of the protocol but in fact part of the first specification. For example, [5] proposed a smart seed to ensure that the original provider of a file uploads the whole file once before sending duplicated data. Similar, [1] describes the super-seeding method, which tries to overcome the same problem at start-up. Also to improve fairness the tit-for-tat strategy can be replaced by a bandwidth trading scheme [8].

3 Simulation Methodology

The objective of this work is to estimate the differences between packet-level and flow-level simulations for BitTorrent-like P2P networks. We denote it as BitTorrent-like, because we do not intend to implement a specific version of BitTorrent, but aim at assessing the differences between a full simulation of all network layers and simplified simulations on the application-layer. Thus, some functionalities were simplified and others were omitted. In detail, in our simulator no torrent file is used and the downloaded data is not checked for data integrity by hash values. The HTTP tracker protocol is not implemented. That is, all tracker traffic is directly given to the peers rather than being transmitted over the network. We use this simplification because we are predominantly interested in the efficiency of the data transfers between the peers. Each peer runs the basic unchoking algorithm in [7]. Thereby, anti-snubbing and the endgame-mode are neglected. We omitted the endgame mode in our implementation because it is not clearly specified when a peer switches to the endgame mode. Hence, different implementations realize it differently. Furthermore, anti-snubbing was omitted because it can result in situations where a peer does not contribute its upload bandwidth although it can transfer data to other peers. This can cause inefficiency in the network.
However, we implemented the super-seeding functionality [1] because it improves considerably the performance for the flash crowd scenario (see Section 3.4).

3.1 Packet-level Simulator

We used ns-2 [14] for the packet-level simulations and added four classes. These are BitTorrent application, BitTorrent tracker, BitTorrent connection and BitTorrent message. Furthermore, minor changes were done to the implementation of the node, agent and FullTcp implementation in ns-2 to handle a BitTorrent application [1].
ns-2 provides different versions of the TCP protocol. We used FullTcp since it supports bidirectional data transfers. Normally, ns-2 does not transfer application data but similar to the TcpApp implementation for a web cache we extended the code to handle application data. Therefore, we added the class BitTorrent connection, which buffers the application data at the sender. On the other end of the connection the TCP agent informs the BitTorrent application with an upcall when new data is received. If a full BitTorrent message is received, the application accesses the data at the sending BitTorrent connection and handles it.
The FullTcp implementation does not support a receiver advertised window. Since autotuning of the buffer size is available (e.g. Linux kernel 2.6.17 and later) we assume that the buffer size at the receiver is not the bottleneck in the network. Interested readers are referred to a Gnutella implementation [11] for ns-2 which incorporates the advertised window.
The BitTorrent implementation is modular. That is, the peer and the piece selection algorithms can be replaced by alternatives. Thus, different implementations of BitTorrent-like networks can be compared easily by simulations.

3.2 Flow-level Simulator

It is widely assumed that the bottlenecks in P2P networks are the access links of the users. This seams reasonable because most peers are home users who are connected e.g. with DSL or cable modems to the Internet. Thus, most of their uplink bandwidths are in the range of 64kbps to 1024kbps. Furthermore, the core of the network shows a low utilization of the available bandwidths [15] and small packet loss rates due to congestion [9].
Often also the TCP behavior is ignored and the simulation takes only the application-layer into account and uses the access link bandwidth as the speed of the bottleneck. In our flow-level simulator we also neglect the downlink and assume the uplink is the bottleneck in the whole network. Since home users often have asymmetric access links (e.g. ADSL) and/or set the upload bandwidth in the application lower than the physical capacity, this assumption is reasonable. When altruistic behavior is noticeable the download capacity must also be taken into account since in this case a

[1]The source code of our BitTorrent implementation will be available from the website of the corresponding author

peer receives data from numerous other peers. A model for this case is discussed in [17].
The flow-level simulator is a derived class of the packet-level simulator. This enables the reuse of large parts of the code and a correct comparison between both simulation approaches. Only the way how BitTorrent messages are handled is changed. While in the packet-level simulator messages are handled by the TCP agent, the flow-level simulator differentiates between control and data messages. The control messages are delivered directly to the receiver whereas the *PIECE* messages are queued in an upload queue. To minimize the number of generated events the upload queue of a peer is handled before the unchoking algorithm is run. That is, a peer computes the amount of data which it transmits between two consecutive runs of the peer selection algorithm, denoted as unchoking interval in the following, and dequeues the number of allowed messages from the upload queue. Then, these messages are directly handled by the receiver. To ensure a fair sharing of the upload bandwidth requests are not pipelined in the flow-level simulations.
The flow-level simulator uses a single recurring event per peer. This reduces the computational complexity significantly. Further details on efficient discrete event simulations can be found in [6].

3.3 Topologies

A network topology is only used for the packet-level simulator. Based on the assumption that the bottleneck of the network is at the access links of the users and not at the routers, we use a simplified topology in our simulations. We model the network with the help of access and overlay links. Each peer is connected with an asymmetric link to its access router. All access routers are connected directly to each other modeling only an overlay link. This enables us to simulate different upload and download capacities as well as different end-to-end (e2e) delays between different peers. This topology is denoted as the overlay topology in the following.
One disadvantage of the overlay topology is the required number of links. For P peers in the network the number of links in the topology is $Z = (P+1)\cdot P/2$, because P links are needed to connect each peer with an access router and $(P-1)P/2$ links are needed to connect all routers with each other. Thus, the number of links increases quadratically with the number of peers in the network potentially causing a memory problem for large peer populations.
One possible solution to overcome this problem is to neglect the differences in e2e delay between the peers. This would result in omitting the overlay links in the already simplified topology resulting in a star topology with $Z' = P$ links.
Another approach is to take a real topology and connect peers randomly to the routers of the core network. We used the German Network [4] in this work. A more sophisticated solution for simulations with different e2e delays between peers is presented in [13].
The performance of the overlay, star and German network topologies are compared by simulations in Section 4.3.

3.4 Simulation Scenarios

To model the user behavior in BitTorrent networks we have to consider, most importantly, the leecher arrival process and the seed leaving process. For a detailed representation also the download pauses of a leecher and its leaving process have to be taken into account. Thereby, the leaving process is initiated by an error or by an user, who is not satisfied with the progress of the download. Additionally, seeds also rejoin the network although a user has no motivation to do that. This is due to the implementation of specific clients, where the software automatically connects to the network after start-up and serves the file. For the sake of simplicity we concentrate in the following on the leecher arrival process and the seed leaving process and neglect the others.
The first experiment studies the worst case scenario for file dissemination, which is called the flash crowd effect. Thereby, initially only one seed and a number of leechers are in the network. This represents an extraordinary burden on the network because only one peer can upload data to others at the beginning. We assume new seeds stay in the network until all peers have finished their download. For the case of peers with the same upload capacity C the download time can be estimated analytically. We denote the file size, the chunk size and the number of parallel uploads as S_F, S_C and U, respectively. After S_F/C the whole file is available in the network. The last U chunks uploaded by the seeder are the rarest chunks in the network as each of them is only available at the seeder and one other peer. For a uniform dissemination of the rarest chunks the seed uploads each rarest chunk once. The other peers upload the rarest chunk they hold U times. Thus, the number of peers which hold a rarest chunk increases by $\lceil (1+1/U)(U+1)^i \rceil$, where i is the number of time intervals it takes to upload a full chunk to U peers. This time interval can be computed with $U \cdot S_C/C$. With P peers in the network the total download time is

$$t_{\text{flashcrowd}} = \frac{S_F}{C} + \frac{US_C}{C}\left\lceil \log_{U+1}\left\lfloor \frac{P}{1+\frac{1}{U}} \right\rfloor \right\rceil, \quad (1)$$

where $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ denotes the ceiling and the floor function, respectively. Equation (1) holds when P is a multiple of $(1+1/U)(U+1)$. For other values of P it is exact if each chunk is downloaded from a single source.

In the second experiment we simulate a constant number of peers in the network. We assume S seeds and L leechers are present in the network. Furthermore, a leecher leaves the network when its download is completed. To keep the peer population constant a new leecher with no chunks enters the network. At the beginning of the simulation the leechers have a random set of chunks. This simulation setup studies the performance of peers with different download progress. Furthermore, it can be easily studied analytically. The total capacity in the network is $\sum_{s=1}^{S} C_s + \sum_{l=1}^{L} C_l$, where C denotes the upload capacity of a peer. The objective of the tit-for-tat strategy in BitTorrent is to upload to those peers which upload in return. Thus, in the optimal case the download bandwidth achieved by trading chunks with other peers is equal to the peer's own upload bandwidth. Another portion of the download bandwidth is contributed by the seeds in the network. If these resources are allocated in a fair manner a peer has a download bandwidth x of

$$x_l = C_l + \frac{\sum_{s=1}^{S} C_s}{L} \tag{2}$$

Equation (2) overestimates the download performance of a peer because it neglects that a peer needs a chunk which is of interest for other peers. This is not always the case. Particularly, at the start a peer has no complete chunk and cannot barter with its own bandwidth. Thus, its download bandwidth depends only on the altruistic behavior of others.

4 Simulation Results

4.1 TCP behavior

Before simulating the BitTorrent network we were interested in the TCP performance in general. We measured the aggregated goodput of four TCP connections which share one bottleneck link. Thereby, four corresponds to the default number of data uploads in BitTorrent. Since the bottleneck link represents the access link to the ISP we varied the capacity from 64 kbps to 1024 kbps and set the delay to 1 ms. Each TCP connection traverses another link where the one-way propagation delay is determined according to a uniform distribution between 1 ms and 100 ms. With a capacity of 100 Mbps no packet losses occur at the second link. We used the FullTcp/NewReno agent in ns-2. Packet size was 1460 Bytes excluding IP and TCP headers. A good queue limit was determined for each upload capacity beforehand. We measured the goodput over two time intervals. Thereby, 10 s is exactly the interval of the peer selection algorithm of BitTorrent. The mean values and 95% confidence intervals over 10 runs are depicted in Figure 1.
The utilization of the upload bandwidth increases with increasing upload capacity. For measurements over 100 s the utilization is larger than 90% for 128 kbps and higher. Only

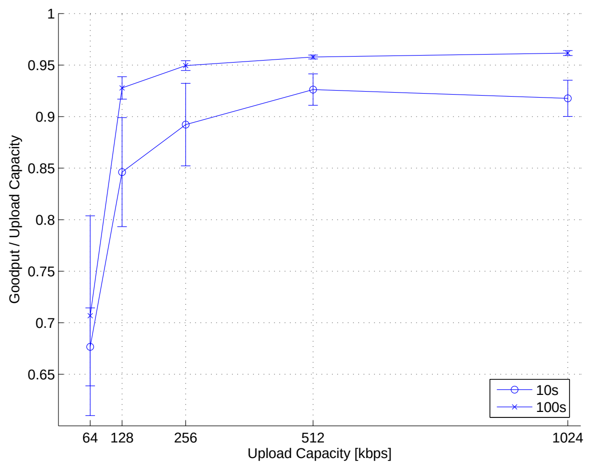

Figure 1. Normalized TCP goodput

for 64 kbps the utilization is poor for both measurement intervals. Here the bandwidth-delay-product is so small that the TCP connections cannot operate in congestion avoidance state for this packet size. But also for 128 kbps TCP performance varies strongly influenced by the different round-trip-times (RTT) of the connections for a measurement time of 10 s.
These results indicate the influence of TCP on the BitTorrent performance. Because the utilization of TCP increases with increasing upload capacity, we expect that the differences between packet-level and flow-level simulations of BitTorrent become smaller. Furthermore, by looking at the differences for the two measurement intervals in Figure 1, the time intervals, for which peers are unchoked, have an influence on the utilization of the available bandwidth.

4.2 Flash crowd

For the flash crowd scenario we measured the total download time until all peers have finished their download. We simulated a 100 MB download with varying number of peers, which have an upload capacity of $C = 1024$ kbps. Figure 2 shows the mean and the 95% confidence intervals for 5 simulation runs. The theoretical values are computed with equation (1). For packet-level simulations the star topology is used. To simulate asymmetric access links the downlink capacity is set 8 times higher than the uplink speed. The propagation delay of each link is determined randomly between 1-50 ms. Super-seeding mode is used for both simulation types.
The original content provider needs at least 819 s to upload the file to the network. Theoretically, the distribution of the data to all peers in the network is very fast and scales with

	64 kbps	128 kbps	256 kbps	512 kbps	1024 kbps
Theoretical [s]	13500	6750.2	3375.1	1687.6	843.78
Flow-level [s]	16161±78	7791±122	3637±49	1769±5.7	897±6.7
Packet-level [s]	17645±201	8549.3±72.7	4220.2±34	2043.3±6.4	1028.1±4.3

Table 1. Total download time for flash crowd of 100 peers for different upload capacities (and 95% confidence intervals)

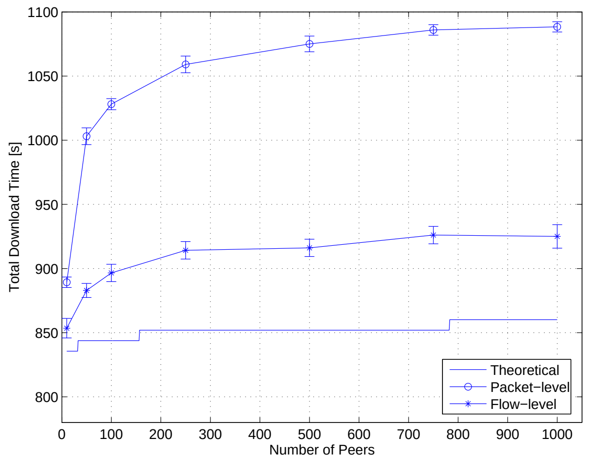

Figure 2. Total download time of flash crowd with different number of peers

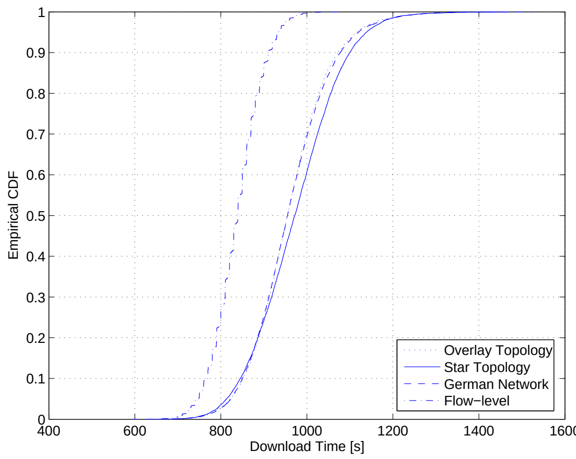

Figure 3. Constant population of 100 peers for flow-level and different topologies for packet-level

an increasing number of peers. Also the simulation results reveal that the content distribution with BitTorrent scales well. At flow-level the results are only up to 9% higher as the theoretical values. Packet-level results are up to 27% higher.

At flow-level nearly the same download time is observed for 250 peers and more. Although the increase of the download time at packet-level is small, it increases by 6% from 100 to 1000 peers.

The super-seeding mode improves the download performance considerably. Without it we measured up to 70% higher download times, because the original seed frequently uploads chunks, which were already available in the network instead of the missing ones. Similar results are also reported in [5] for flow-level simulations.

In our simulations super-seeding mode is left when the seed has received HAVE messages for every chunk. This was around 830 s and 890 s for flow-level and packet-level simulations, respectively.

Further results with different upload capacities are summarized in Table 1 for a network of 100 peers and several upload capacities. Here, the differences between theoretical values and flow-level and packet-level results are at most 20% and 30%, respectively. With larger uplink bandwidth the differences are smaller (around 6% and 15% for 1024kbps). Because this behavior is observed for both simulation models, not only the TCP behavior discussed in Section 4.1 accounts for it but also the BitTorrent implementation itself. Since with smaller uplink bandwidth a peer uploads less data per unchoking interval, a downloader has to be unchoked for multiple unchoking intervals until it can complete a chunk. E.g. with 128 kbps only 160 KB are transferred at most in an unchoking interval of 10 s. Thus, a peer has to be unchoked for more than 6 intervals to complete a single chunk. On the other hand with 1024 kbps up to 5 chunks can be uploaded in one unchoking interval.

4.3 Constant Peer Population

In this section we present results for the second experiment discussed in Section 3.4 for 1 seed and 99 leechers. All peers have an upload capacity of 1024 kbps. Figure 3 shows the results for the flow-level and the three topologies

discussed in Section 3.3 for packet-level simulations for in total 10000 simulated peers.
To ensure a fair comparison between the different topologies we set the propagation delay appropriately. We choose the propagation delay randomly between 1-100, 1-50 and 1-18.5 ms for the overlay, star and German network topology, respectively. (The German network has an average distance of 2.7 hops [4].)
The results in Figure 3 show similar performance for the three topologies. The mean download times are 960, 973 and 961 s for the overlay, star and German network topology, respectively. Thus, the packet-level results are less than 20% above the theoretically optimal value of 811 s, which can be computed by dividing the file size of 100 MB with equation (2). Moreover, the mean download time at flow-level is with 838 s only 3% larger than the optimal case.
One reason for this good performance is that peers finish their first chunk fast and thus can provide their upload bandwidth to others. For the packet-level simulations it takes a mean value of around 15 s until a peer finishes the download of its first chunk. Since uploads are done for the flow-level simulations only once per unchoking interval, the mean is around 22 s to finish the first chunk in this model.

4.4 Delay

It is well known that the TCP throughput depends on the RTT (e.g. [10]). Thus, if two TCP flows compete for the resources of the same bottleneck link, the connection with a smaller RTT will receive a higher bandwidth share than the other one. Since a BitTorrent peer uploads to those peers from which it downloads with high rates, peers on links with large delays experience a worse performance. To confirm this claim we ran a simulation using the star topology. Thereby, one set of peers is connected over a link with 10 ms delay and the other set with 100 ms delay. The cumulative distribution functions for 3 runs are shown in Figure 4.
The results from the different runs are nearly identical. With a smaller delay peers download much faster. The mean download time over all runs is 857 s for peers connected on a link with 10 ms delay and 1237 s connected by a link with 100 s delay. Thus, the mean download performance deteriorates by 44% for peers with higher delays.

4.5 Simulation Complexity

In general, the complexity of simulations is much higher on the packet-level than on the flow-level. For example, the simulations described in Section 4.3 for 100 active and in total 10000 peers take, depending on the topology, between 8-10 hours at packet-level and less than 1 hour at flow-level with an Athlon 64 x2 Dual Core 4400+ processor. Furthermore, simulation time increases significantly with increasing number of active peers.

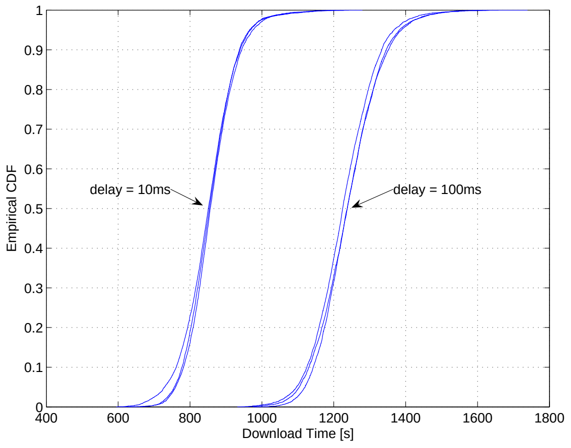

Figure 4. Download times for peers with different delays

For all simulations we used the heap structure for the event scheduler of ns-2. Although, theoretically a calendar queue has a hold time of $O(1)$ and outperforms a heap with $O(\log(n))$ (see e.g. [6] for details), we observed better performance using the heap structure [2].
Except for the overlay topology, which already exhausts the RAM of 3GB for 1000 peers, the memory consumption is acceptable. The star topology and the flow-level consume 8% and 3.5% of the available RAM for 1000 peers only.

5 Conclusion

This paper determines the differences between packet-level and flow-level simulations for BitTorrent-like P2P networks and compares the results with simple analytical models. The results for the flow-level simulations are near to the optimal values indicating that the BitTorrent protocol works efficient for the discussed scenarios. Naturally, packet-level simulations deviate more from these values but in the conducted experiments the download performance is at most 30% worse as compared to the analytical results.
Although packet-level simulations are more complex as on flow-level, they are required for studying cross-layer interactions. We showed by simulations that the download performance with BitTorrent depends on the delay to other peers. This stems from the usage of TCP as transport protocol. Future work will study these cross-layer interactions in more detail.

[2]The patch from http://netlab.caltech.edu/∼weixl/technical/ns2patch/ for ns-2 did not speed up our simulations.

Acknowledgements

The authors would like to thank Prof. Ulrich Killat, Prof. Phuoc Tran-Gia, and Prof. Jörg Eberspächer for enabling and supporting this work.

References

[1] *Bittorrent Protocol Specification v1.0*. http://wiki.theory.org/BitTorrentSpecification.

[2] *Experimental Draft: BitTorrent Trackerless DHT Protocol Specifications v1.0*. http://www.bittorrent.org/Draft_DHT_protocol.html.

[3] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), Oct. 2000.

[4] A. Betker, C. Gerlach, R. Hülsermann, M. Jäger, M. Barry, S. Bodamer, J. Späth, C. Gauger, and M. Köhn. Reference transport network scenarios. MultiTeraNet Report, 2003.

[5] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving BitTorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, 2005.

[6] A. Binzenhöfer, T. Hoßfeld, G. Kunzmann, and K. Eger. Efficient simulation of large-scale P2P networks: Compact data structures. In *Workshop on Modeling, Simulation and Optimization of Peer-to-peer environments (MSOP2P) in conjunction with Euromicro (PDP 2007)*, Naples, Italy, Feb. 2007.

[7] B. Cohen. Incentives build robustness in BitTorrent. In *Proc. 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, June 2003.

[8] K. Eger and U. Killat. Bandwidth trading in unstructured P2P content distribution networks. In *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P2006)*, pages 39–46, Cambridge, UK, Sept. 2006.

[9] S. Floyd. *Measurement Studies of End-to-End Congestion Control in the Internet*. http://www.icir.org/floyd/ccmeasure.html.

[10] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Trans. Netw.*, 7(4):458–472, 1999.

[11] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto. Mapping peer behavior to packet-level details: A framework for packet-level simulation of peer-to-peer systems. *mascots*, 00:71, 2003.

[12] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. In *Passive and Active Mesurements*, pages 1–11, April 2004.

[13] G. Kunzmann, R. Nagel, T. Hossfeld, A. Binzenhöfer, and K. Eger. Efficient simulation of large-scale P2P networks: Modeling network transmission times. In *15th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2007)*, Naples, Italy, Feb. 2007.

[14] *ns-2 (The Network Simulator)*. Sources and Documentation from http://www.isi.edu/nsnam/ns/.

[15] A. Odlyzko. Data networks are lightly utilized, and will stay that way. *Review of Network Economics*, 2(3):210–237, September 2003.

[16] R. Qiu, D. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. *Computer Communication Review*, 34(4):367–378, 2004.

[17] D. Schlosser, T. Hoßfeld, A. Binzenhöfer, K. Eger, and G. Kunzmann. Efficient simulation of large-scale P2P networks: Modeling bandwidth. submitted for publication.