ChunkSim: Simulating Peer-to-Peer Content Distribution

Jussi Kangasharju Uwe Schmidt Dirk Bradler Julian Schröder-Bernhardi
Department of Computer Science, Darmstadt University of Technology
Hochschulstrasse 10, 64289 Darmstadt, Darmstadt, Germany
Email: jussi@tk.informatik.tu-darmstadt.de

Keywords: Peer-to-peer, content distribution, BitTorrent

Abstract

Peer-to-peer content distribution is currently one of the main sources of traffic on the Internet. In spite of the extreme popularity of systems like BitTorrent, only a little attention has been paid to such systems in the research world. We believe one cause behind this lack of attention is the lack of suitable tools. Peer-to-peer content distribution aims at extremely large systems, consisting of several thousands or even millions of peers, which makes simulation a logical choice in studying them. In this paper, we present *ChunkSim*, a simulation framework for investigating peer-to-peer content distribution. We identify the key functionality of a peer-to-peer content distribution system and have implemented these requirements in ChunkSim. Furthermore, ChunkSim is an extensible framework and we demonstrate how it can be extended and discuss some common extensions.

1. INTRODUCTION

Peer-to-peer file sharing systems typically compose about 70% of current Internet traffic [11, 27], and the systems are highly popular with many users. Content being shared on these networks typically includes music files, videos, and software. In a file sharing network, peers join the network and offer some files for download. Other users can search for files with keyword queries and can download any file being offered in the system. Examples of popular file sharing networks are eDonkey [9] and Gnutella2 [14]. Although the main use for such networks in the past has been copyright-infringing sharing of files, newer systems have also fully legal uses.

Examples of the newer systems are the so-called *peer-to-peer content distribution systems*, such as BitTorrent [6]. In contrast to file sharing networks, these content distribution networks build *one network* for each file that is being distributed. Peers which join this one network are all interested in the same file, and if a peer wants to download two different files, it has to join two different networks. Another difference between file sharing and content distribution is that whereas in file sharing networks users have to search for the file they want, in content distribution networks each network is uniquely associated with exactly one file. Peer-to-peer content distribution is widely used for software distribution, such as Linux [15] or patches [28].

Peer-to-peer content distribution has been shown to be very effective way of distributing large files to a large number of interested users [8,13,30]. Of a particular interest has been the study of how efficiently files can be distributed in a peer-to-peer content distribution network [3, 5, 10, 18, 19, 24, 26]. The most common approach has been to model the systems analytically, with several simplifying assumptions, or by simple simulations. Although each of the cited studies has its merits, the community would greatly benefit from a unified evaluation framework, in order to allow an objective comparison of algorithms and ideas. Given the need for large scale evaluations and the considerable effort required to set up real-world test systems, simulation appears to be the approach of choice for most of the above-mentioned work.

In this paper, we present *ChunkSim*, a simulator specifically designed for simulating peer-to-peer content distribution systems. Although other peer-to-peer simulators exist, such as [22, 23], they are often hard to tailor into the specifics of peer-to-peer content distribution. This is because they attempt to model the whole overlay network, but in content distribution the details of the network are typically irrelevant to the main issues and questions that we seek to answer. Instead, it is important to be able to add new distribution strategies, peer selection strategies, etc.

Our contribution is an extensible simulation framework, specifically designed for investigating peer-to-peer content distribution systems. It incorporates all the key features of such systems: chunk selection, peer selection, peer arrivals and departures, as well as a simple but sufficient network connectivity model.

This paper is organized as follows. First, we provide an overview of chunk-based peer-to-peer content distribution systems, using BitTorrent as an example. We also discuss the main characteristics of such systems. We then present ChunkSim, show its architecture and discuss extension possibilities. Later we discuss related work on peer-to-peer simulators, and conclude the paper and provide directions for future work.

2. CHUNK-BASED CONTENT DISTRIBUTION SYSTEMS

We consider the so-called *chunk-based* content distribution systems as representative systems for peer-to-peer content distribution. In such systems, the file to be distributed is divided into several, equal-sized *chunks* or blocks and a

peer needs to retrieve all the chunks of a file to download the complete file. Typically, a peer would get different chunks from different peers. This is the approach taken by BitTorrent, which we will discuss in more detail below. Similar approach is taken by the Avalanche system [12]. However, Avalanche is based on the use of network coding [2], which lets peers to create new chunks as linear combinations of other chunks. From a content distribution point of view, there difference between these two approaches is minimal.

2.1. BitTorrent

BitTorrent [6] is a new and popular form of peer-to-peer content distribution. It is especially used for delivering large files, not only (illegal) multimedia content, but also for large software packages, such as Linux distributions [15].

A BitTorrent network works as follows. Originally, the file to be distributed is available from one server, called *seed*. In addition to the seed, there is a *tracker* server which keeps track of all the clients in the network. A client who wants to download the file, needs to get the so-called `.torrent`-file which contains metadata about the file (length, chunks, checksums, etc.) and the address of the tracker for that file.

The client then contacts the tracker and receives a list of peers who are currently downloading that file (also called *leechers*) or who have already downloaded it and are still connected to the network (known as *seeds*). The client then picks some peers from this *peer set* and starts downloading chunks from them. BitTorrent uses a tit-for-tat policy, so that a client serves chunks to other peers who are serving chunks to it. A client will try to find the best set of peers from which to download, by choking and unchoking peers. Peers explore the peer set through a process called *optimistic unchoking*, where a peer randomly picks a peer from its peer set and uploads a chunk to this peer. This process has been observed to improve the performance of the download. For a detailed description of how BitTorrent works, please see [15].

The peer knows which chunks each of the peers in its peer set has. To achieve this, each peer periodically checks with the tracker and updates the list of which chunks it has and updates the lists for the other peers in its peer set.

2.2. Important Parameters

We now discuss what are the important parameters which capture the essential properties of a chunk-based content distribution system. A chunk-based content distribution system requires that a peer, who wants to download something, make two choices. First, it needs to select which chunk to download next, and second, after it has chosen the chunk, it must select from which peer it downloads that chunk. The selected chunk could be available from several peers in the peer set, thus requiring the peer to select one of them.

There has been a considerable amount of work on which chunk to select next [3, 5, 10, 18, 26], in order to maximize some performance metric. A relatively tight lower bound and a discussion on the possible metrics are given in [18]. Many of the existing works propose several algorithms and compare them against the algorithms in the same paper. However, a good comparison between the algorithms in different papers has not been presented, thus making it hard to understand the fundamental differences between the algorithms.

In contrast, the problem of selecting the peer to download from has received much less attention. Intuitively, it would be suitable to select the nearest of all the possible peers, but in practice this may not be easy to achieve. Regardless, it has been shown [8, 13], that a good selection of the peer can have significant impact on the amount of network traffic sent during the distribution of the file. BitTorrent attempts to select the peers which offer the fastest downloads, through the above-mentioned choke mechanism. This mechanism has been found to yield satisfactory performance [15, 20], but there are no works comparing different peer selection policies.

A third important parameter concerning the distribution is the arrival and departure of peers. The arrival process determines when new peers join the distribution network and defines the number of peers in the network at any given time. The arrival process is typically assumed to be Poisson. The departure process, on the other hand, determines when peers leave the system, is much harder to define. First, some peers might leave *before* they have completed the download, because the user has grown impatient. Second, after a peer has completed the download, it might still stay on as an additional seed, for whatever the reason. Most of the existing work tends to assume that peers download the complete file and then stay on as seeds with a certain probability.

A final parameter concerns the network between the peers. Because the chunks are typically large, on the order of a few hundred KB, we are only concerned with the average network performance over longer timescales. Short-term details—such as TCP congestion windows, exact routing, network loss, etc.—are not really relevant, when comparing the performance of content distribution systems. This allows us to abstract the network between the peers into a simple number, which states how much bandwidth over long timescales is available between two peers. Note that this number can change during the distribution, but it can certainly be assumed to remain constant for several consecutive chunks. Similar approach is taken in [4].

3. CHUNKSIM

In this section, we will present ChunkSim. First, we present the general architecture of ChunkSim, then we discuss the detailed settings and configuration possibilities. Finally, we

show how it can be extended to include new algorithms and strategies.

3.1. Main Characteristics

ChunkSim is built as a discrete event simulator and it models the main entities in the system (i.e., the peers) and the relationships between these entities (i.e., chunk transfers between peers). This model yields the two main data structures of ChunkSim, namely peers and chunks.

Peers

Peers in ChunkSim are divided into *peer groups*, where each peer in the group has the same properties as the other peers in the group. There are no limitations on the number of groups or peers. Peers can be configured in terms of their networking capabilities and the user behavior.

Networking capabilities are modeled by setting the up- and downstream bandwidths for the peer, and how many concurrent upload and download connections the peer is willing to handle. The bandwidths abstract out the network level details and reflect the long-term average bandwidth from or to that peer from any point in the network. Since most peer-to-peer networks run on the actual user PCs, it is highly likely that the user's access link's bandwidth is the bottleneck, in particular for uploads for users with DSL or cable modem connections. Later we will explain how this simple network model can be extended to take into account more characteristics of the network.

The number of concurrent connections determines how many other peers a peer is willing to serve, or from how many peers a peer attempts to download chunks at any time. BitTorrent typically attempts to maintain 4 active connections in both directions, but ChunkSim allows for any combination of upload and download connections. (Setting a zero for upload connections is possible, but the same effect can be obtained with the "freeriders" parameter below.) The upload bandwidth is shared evenly between all the concurrent connections, naturally taking into account the download bandwidths of the peers. For example, if a peer has 1000 kbit/s of upload bandwidth and it is serving 4 peers, 3 with 1000 kbit/s download bandwidth and 1 with 100 kbit/s download bandwidth, then the "slow" peer will get its 100 kbit/s and the remaining 900 kbit/s is evenly divided between the 3 "fast" peers.

The user behavior is modeled with two parameters: the number of freeriders in the group and the disconnect probability. Freeriders are peers who are only interested in downloading files, but not interested in contributing anything back to the system. Freeriding in general has been observed to be prevalent in peer-to-peer networks, but no studies on its real effects have been made. In ChunkSim, freeriding peers refuse to upload any chunks to others; however, other peers still keep on serving them. As mentioned above, BitTorrent uses a tit-for-tat policy which in practice forces peers to upload if they wish to download. In BitTorrent, a freerider would be a peer who disconnects immediately after having completed its download. ChunkSim models this with the disconnect probability parameter, which gives each peer a probability that they will leave the network after having completed their download.

Our choice of including several possibilities for modeling freeriders stems from the lack of studies in this area. Although the *existence* of freeriding in real-world systems is demonstrated [1, 25], there are no studies which evaluate the *effect* of freeriders on the system performance.

We believe the above parameters are sufficient to model the peers in a peer-to-peer content distribution system. ChunkSim is easily extensible, so that new parameters can be taken into account with ease.

Chunks

The other main entity in the system are the relationships between the peers, which are realized through exchanging chunks between peers. The chunks themselves are parts of files, which we model as follows. ChunkSim allows us to define several files for distribution, but simulates each file as its separate content distribution network, as in a real-world system. A peer participating in several networks, might see its participation in one network being hindered because it has already filled its download capacity from another network. Files are divided into different file types according to file size and inside each type, we can create multiple unique files.

In ChunkSim, we can configure the size of a chunk, which is then valid for all files and all chunks. For each file in the system, we can set the file size, which automatically determines the number of chunks in the file. (For simplicity and without loss of generality, the process is constrained in a way which always results in a file size being an integer multiple of the chunk size.)

We can set the number of initial copies of each file and determine on which peers the initial copies are placed. The placement happens at a granularity of peer groups, such that we can select at what proportions should the file be distributed among the peer groups, and a probability distribution for placing the copies on the individual peers inside the groups. The initial copies are always placed as complete files.

Requests

The requests from the peers to the files are controlled by two parameters: mean request period and the file popularities. The mean request period simply is a parameter for a Poisson distribution which defines the mean time between two requests. File popularities can be selected as a two-step process. If there are more than one file being distributed, we can select the request probability per file type. Inside each file type, we have a configurable request probability distribution. Currently we have implemented uniform and Zipf-like distributions [7].

3.2. Architecture

Figure 1 shows the control flow in the simulation. Since ChunkSim is a discrete-event simulator, the control flow is executed only when an event is triggered at a peer. The bulk of Figure 1 depicts the actions inside one peer. The File Request Generator triggers a new request for a file from a peer. Typically, this means the arrival of a new peer into the distribution network. ChunkSim first determines randomly which of the peers will initiate the download.

The download gets inserted into the download queue, by creating a separate download event for each of the chunks in the file. When the download queue is processed, we first check if the peer can download more chunks of whether all download connections are already used. If new chunks can be downloaded, then we select a peer, and start the download of the chunk from that peer. This happens by inserting an event to trigger the upload at that peer (shown on the right-hand side of Figure 1). If the uploading peer has no more upload capacity because it is already serving other peers, the download request will be queued at that peer. Such queued requests are served always in FIFO order. Note that if the request would be queued, a peer could have the option to cancel this download request and select a new peer for the chunk. This strategy has already been implemented in the simulator and more fine-grained waiting strategies can easily be implemented.

On both downloading and uploading peers, the respective queues get processed according to their current loads. When a chunk has been uploaded (downloaded, depending on the peer), this will trigger new actions. On the uploading peer, we check if we can upload more chunks to other peers, and on the downloading peer, we check if we have completed the file download. When a peer has finished a download, it stays in the network as an additional seed with the given disconnect probability.

3.3. Settings

We now describe the basic settings which have already been implemented. As we discuss below, the functionality can easily be extended with new strategies.

The configuration of ChunkSim is defined in an XML-file which contains the settings. This configuration file can be edited with a graphical interface which is part of the simulator.[1] ChunkSim can be run either in the graphical mode or in a command line mode. The graphical mode provides some additional output, but requires the user to manually start each simulation. The additional output can also be re-created after the simulation from the log files.

The whole simulation length is governed by a global setting which defines how many seconds the simulation should last (in simulation time). In most cases, this would typically be set to a large value, to allow the file distribution to complete. If this maximum time is reached, then the simulation will continue to run until all currently active downloads are finished, but no new downloads will be started. If all downloads have finished before the end of the simulation, the simulation will end.

Another global setting is whether a peer could attempt to download the same file more than once. Since the file request generator will randomly pick a file and a peer, it could be that the selected peer already has downloaded the file. In peer-to-peer networks, peers normally would only download a given file once, but on the Web, users download the "same" files multiple times, i.e., users access the same URL which may or may not have different content (e.g., news page). Typical in most cases would be to disallow repeated downloads since we are concentrating on peer-to-peer networks, but we have left the possibility open for allowing them.

We have currently implemented two chunk selection strategies: random and rarest-first. In the random strategy, a peer picks one chunk, distributed uniformly over all not-yet-downloaded chunks. With the rarest-first strategy, we model BitTorrent's behavior by selecting the chunk with the least number of copies in the network. If there are several chunks with the same lowest number of copies, we pick one of them uniformly at random.

For selecting the peer, we provide two possible strategies: random or best upload. Random picks one of the peers with the chunk uniformly at random. Best upload picks the peer with the highest upstream connection, selecting randomly in case there are several such peers. This again models BitTorrent's behavior. We also provide two additional improvements to the peer selection strategy. Once the peer has been chosen, it could be that its upload capacity is fully used by other uploads and the new request would be queued. We provide a setting for either allowing the request to be queued or sending the request to another peer.

As mentioned above, we can create multiple files and each file will then create its own distribution network which run in parallel with other networks. For controlling the peer behavior, it is possible to determine which peers belong to which networks and how the requests are divided between the different file types. Inside each file type, we provide two popularity distributions, uniform and Zipf-like, for deciding exactly which file was requested. In a typical BitTorrent-simulation, we would have only one file type and one file instance in that type. However, as described above, our file model allows for a very flexible set up of the files in the network, thus making it easy to simulate many different cases.

3.4. Collected Data

ChunkSim collects many different kinds of data, before, during, and after the simulation. Before the simulation we

[1] It is possible to edit the XML-file by hand, but the graphical interface also performs consistency checks and is therefore recommended.

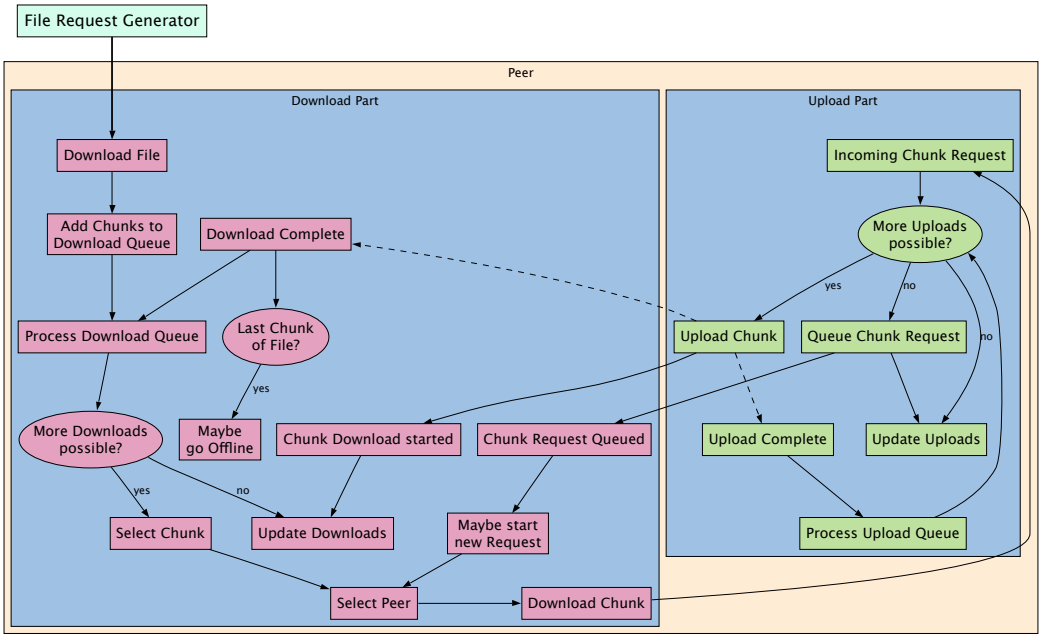

Figure 1. Main simulation loop and called functions

save the initial configuration, which consists of the following items: a list of all peers and their attributes, list of all file types, and a complete list of all the files in the system (recall that there could be several files of a given type); and a list of peers and the initial copies of files placed on them

During the simulation we log each chunk transfer by noting the time when it started and completed, and the identities of the sending and receiving peer. Likewise, for each complete download of a file, we note the start and end times, the peer who downloaded the file, and the file identifier. We also record additional events such as a peer going offline and "errors" by which we mean a situation where a peer wanted to download a chunk but the chunk was not available on any currently online peer. In the current version, peers cannot return online, so these errors imply that the file transfer cannot be completed. Our future plans involve extending the peer behavior model to allow peers to return online after going offline, which would make these errors possible to resolve.

After the simulation is finished, ChunkSim writes a list of all peers with the files they currently possess. By comparing this list with the list given in the beginning, we can determine which peers have downloaded which files (this can also be obtained from the log files written during the simulation).

If the simulator is started from the graphical interface, it will display a window with progress information and statistics, as shown in Figure 2. The information shows the progress of the simulation, both as a percentage as well as an estimated time to complete the simulation. It also displays a graphical view of how many chunk transfers are going on in the system at a given time, how many file downloads are active, and how many downloads are incomplete (because some chunks are unavailable since the last peers with those chunks have left the system). All of the information displayed on screen can be reconstructed from the logfiles. The interface also offers the possibility to set the update frequency of the display and a button for pausing and resuming the simulation. If ChunkSim is run from the command line, a summary line with a brief progress report is printed on the console.

3.5. Performance

ChunkSim is written in Java and uses some external libraries to provide additional functionality. Currently we use Jakarta Commons Math library for generating distributions and random variables [16]; Jakarta Commons Configuration for handling the XML-based configuration file [16]; and JFreeChart for previewing [17]. ChunkSim runs on Java Runtime Environment 5 and works on Windows, Linux, Solaris, and Mac OS platforms.

The main limitation for the size of the simulations which can be done is the total number of chunks in the system. This is because for every chunk, the simulator keeps an array of all the peers which possess that chunk. We experimented with using a hashtable to store the peers which hold a given chunk, but found out that the memory usage was much greater, thus our decision to keep an array.

Typically the number of chunks is significantly (orders of magnitude) larger than the number of peers or files, hence the memory consumption is dominated by the number of

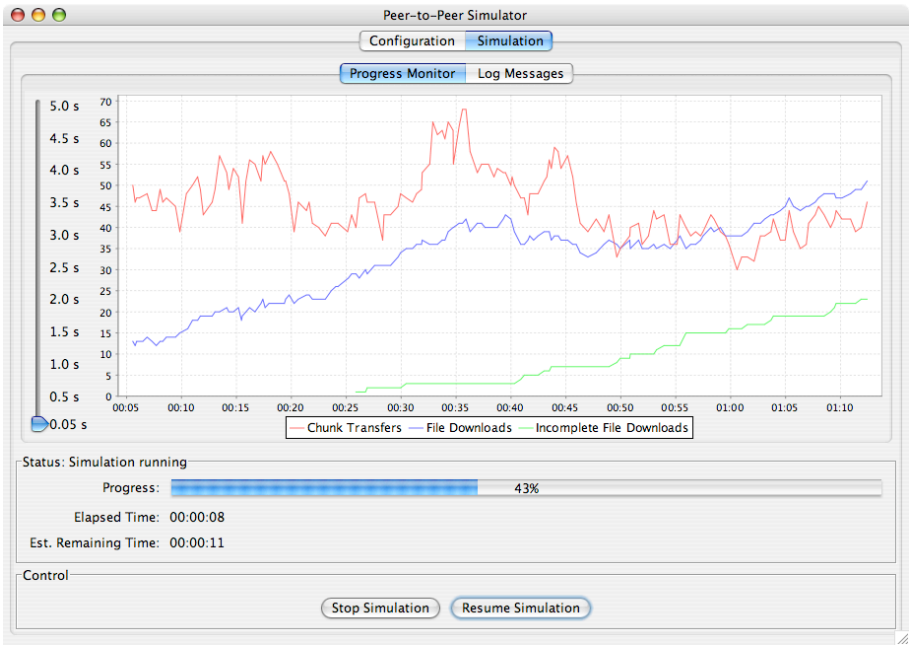

Figure 2. Example of graphical progress report

chunks.[2] We have experimented with different sizes of simulations and have observed that a simulation with $9 \cdot 10^6$ chunks needs a minimum of 500 MB of RAM to generate and about 1 GB to run. Assuming a default chunk size of 256 KB, we conservatively state that we can simulate a set of files with a combined size of about 1 TB (in one or more files) per each GB of RAM available on the machine running ChunkSim. We believe this to be sufficient for virtually all simulations of peer-to-peer content distribution networks. We also tested a large network with 10000 peers and a 1 GB file (4000 chunks) and were able to run it in under 1 GB of memory.

3.6. Extensions

ChunkSim has been implemented in a very modular fashion to allow for easy addition or replacement of functionality. Given the need to simulate many different kinds of systems, we believe that easy extensibility is a vital requirement for a simulator of peer-to-peer content distribution networks.

Easily modifiable components include the file request generator as well as chunk and peer selection strategies. The file request generator is the part of the simulator which determines when new requests arrive (in some cases equal to arrival of new peers). The current behavior of using a Poisson distribution with a configurable mean to determine the time and then selecting a file and peer which requests it at random can easily be overridden by providing a different implementation of the *FileRequestGenerator* class.

To add new chunk or peer selection strategies is easy. All that is required is an implementation of the selection strategy and adding it to a configuration file. After compilation the new strategy is then available to be used in the simulation. Adding more sophisticated chunk or peer selection strategies is part of our future work. In particular, we will consider selection strategies which operate on limited information. Currently every peer knows every other peer in the system, but this could easily be modified to allow peers to know only a subset of the other peers in the system.

Two other areas of extension are incentives and a more detailed modeling of the network. In terms of incentives, the current simulator does not offer any incentive models, but allows peers to be freeriders. Incentive models can be implemented as part of the peer selection strategy, where peers could refuse to upload to certain peers. Similar functionality is already implemented for the case when a peer's upload capacity is full and the downloading peer does not want to have its request queued. Adding an incentive model (e.g., tit-for-tat) would be a minor modification of the above code and could be added like any peer selection strategy.

Our network model is sufficient for modeling the typical Internet-based content distribution systems where each chunk transfer consists of a relatively large amount of data, thus only the long-term average bandwidth between peers is relevant. In other scenarios, it could be desirable to add a more

[2] Naturally, if we are simulating an extremely large network with a small file, the number of peers would also become a factor. However, the data structures are roughly of the same size, so the totals mentioned would apply to the sum of all entities.

fine-grained modeling of the underlying network. This can be achieved by changing the code which schedules the "download completed" event which is generated at the start of the upload of a chunk. Currently this code simply takes the available bandwidth and the chunk size and computes the end time.[3] Again, changing this computation to something more sophisticated which takes into account more parameters is a relatively simple task.

In summary, our simulation framework can easily be extended to handle new situations and strategies.

4. RELATED WORK

In this section, we present other simulators which have been developed for simulating peer-to-peer systems and discuss their suitability for simulating peer-to-peer content distribution systems. The simulators we will discuss are GPS [29], PeerSim [22], PlanetSim [23], and p2psim [21]. Note that there are also other peer-to-peer simulators which have been developed, but these four appear to be the most widely used ones when considering the simulators which have been used in the literature (apart from problem-specific simulators).

GPS [29] is maybe the closest simulator to ChunkSim in terms of features and goals. GPS is a message-level simulator for investigating peer-to-peer systems. The authors use their simulator to investigate a BitTorrent-system. However, being a message-level simulator, GPS needs to model the underlying IP network explicitly and take into account details such as sockets and setting up TCP connections. As we have argued in this paper, we believe that such level of detail is not only hard to model concurrently, but in fact it is also *counterproductive* to research on peer-to-peer content distribution. The main reason for our claim is that typically the interactions of interest in a content distribution system concern the selection of chunks and peers, and the frequency and importance of those selections are governed by the bandwidth between peers. Given that the chunks are typically on the order of a few hundred KB, the message-level dynamics of the network become irrelevant to the overall performance. Hence, we believe that abstracting the network-level details into a simple upload/download bandwidth capacity is sufficient for most cases. As mentioned above, ChunkSim can also be extended to handle a more detailed model of the underlying network, should it ever prove to be necessary.

PeerSim [22] is an event-based simulator for peer-to-peer systems. Its main focus is on epidemic protocols and it provides support for both unstructured and structured networks. The authors claim that PeerSim can scale to networks with million of nodes. PeerSim is a pure overlay network simulator in that it completely ignores the low level network details, such as TCP/IP. The PeerSim website provides implementations of several protocols for PeerSim and demonstrates that the system is extensible. Since the focus of PeerSim is on epidemic protocols, it is not immediately suitable for content distribution evaluation.

PlanetSim [23] is a discrete event simulator written in Java and implements both structured and unstructured overlays. As with PeerSim, the focus is more on the overlay maintenance and communication, not on the application-level communication. There exists a possibility to model the network by providing a suitable modeling component (similar to our proposed extension of ChunkSim). PlanetSim is in active development and the website provides several components for studying different aspects of peer-to-peer systems.

One of the first widely available peer-to-peer simulators was p2psim [21]. Its main focus is on modeling structured peer-to-peer networks. Currently its development appears to have stopped. From the available information, it appears as if p2psim is limited to simulating only structured overlays. This makes it inappropriate for peer-to-peer content distribution, since the low-level overlay mechanics are not relevant to the issues in peer-to-peer content distribution.

In summary, although several simulators have been developed for peer-to-peer systems, none of the existing solutions satisfactorily addresses the needs of peer-to-peer content distribution systems. Existing simulators typically focus on either overlay or message-level issues, which are not relevant for content distribution systems.

In contrast, ChunkSim has been exclusively developed for studying peer-to-peer content distribution systems. It is easily extensible to allow the integration of new selection strategies, and can even be extended to model the network in more detail.

5. CONCLUSION

In this paper, we have presented ChunkSim, a simulator designed for simulating peer-to-peer content distribution systems. Peer-to-peer content distribution is a rapidly growing area and the current lack of good simulation tools severely limits the possibilities for research. ChunkSim is an extensible simulation framework which allows us to study all kinds of peer-to-peer content distribution scenarios. Our tests have shown that it scales to a large number of peers and large files.

REFERENCES

[1] E. Adar and B. A. Huberman. Free riding on Gnutella. Technical report, Internet Ecologies Area, Xerox Palo Alto Research Center, September 2000.

[2] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, July 2000.

[3] Should the available bandwidth change because of other downloads or uploads, the estimate is automatically updated.

[3] A. Al Hamra and P. A. Felber. Design choices for content distribution in P2P networks. *Computer Communications Review*, 35(5):29–40, October 2005.

[4] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *Proceedings of IEEE Infocom*, Barcelona, Spain, April 2006.

[5] E. W. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Proceedings of Fifth International Workshop on Quality of Future Internet Services*, Barcelona, Spain, September 2004.

[6] BitTorrent web site. <http://www.bittorrent.com>.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE Infocom*, New York, NY, March 21–25, 1999.

[8] S. Das and J. Kangasharju. Evaluation of network impact of content distribution mechanisms. In *Proceedings of International Conference on Scalable Information Systems*, Hong Kong, May 2006.

[9] eDonkey web site. <http://www.edonkey2000.com>.

[10] P. A. Felber and E. W. Biersack. Self-scaling networks for content distribution. In *Proceedings of Self-**, Bertinoro, Italy, May 2004.

[11] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 17(6):6–16, 2003.

[12] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proceedings of IEEE Infocom*, Miami, FL, March 2005.

[13] G. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovic. Planet scale software updates. In *Proceedings of ACM SIGCOMM*, Pisa, Italy, August 2006.

[14] Gnutella web site. <http://www.gnutella.com/>.

[15] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. In *Proceedings of Passive and Active Measurements*, April 2004.

[16] Jakarta commons project web site. <http://jakarta.apache.org/commons/>.

[17] Jfreechart web site. <http://www.jfree.org/jfreechart/>.

[18] J. Kangasharju and J. Kangasharju. An optimal basis for efficient peer-to-peer content distribution. In *Proceedings of 15th International Conference on Computer Communications and Networks*, Arlington, VA, October 2006.

[19] R. Kumar and K. W. Ross. Peer-assisted file distribution: The minimum distribution time. In *Proceedings of HotWeb*, Boston, MA, November 2006.

[20] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *Proceedings of Internet Measurement Conference*, Rio de Janeiro, Brazil, October 2006.

[21] p2psim website. <http://pdos.csail.mit.edu/p2psim/>.

[22] Peersim: A peer-to-peer simulator, web site. <http://peersim.sourceforge.net>.

[23] Planetsim: An overlay network simulation framework. <http://planet.urv.es/planetsim/>.

[24] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proceedings of ACM SIGCOMM*, Portland, OR, September 2004.

[25] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing networks. In *Proceedings of Multimedia Computing and Networking*, San Jose, CA, January 2002.

[26] M. Schiely, L. Renfer, and P. Felber. Self-organization in cooperative content distribution networks. In *Proceedings of IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, July 2005.

[27] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, April 2004.

[28] World of warcraft web site. <http://www.worldofwarcraft.com>.

[29] W. Yang and N. Abu-Ghazaleh. GPS: A general peer-to-peer simulator and its use for modeling bittorrent. In *Proceedings of MASCOTS*, 2005.

[30] X. Yang and G. de Veciana. Service capacity of peer-to-peer networks. In *Proceedings of IEEE Infocom*, Hong Kong, March 2004.