A BitTorrent Module for the OMNeT++ Simulator

Konstantinos Katsaros
Mobile Multimedia Laboratory
Department of Informatics
Athens University of
Economics and Business
Athens, Greece
ntinos@aueb.gr

Vasileios P. Kemerlis
Network Security Laboratory
Computer Science
Department
Columbia University
New York, NY, USA
vpk@cs.columbia.edu

Charilaos Stais
Mobile Multimedia Laboratory
Department of Informatics
Athens University of
Economics and Business
Athens, Greece
stais@aueb.gr

George Xylomenos
Mobile Multimedia Laboratory
Department of Informatics
Athens University of
Economics and Business
Athens, Greece
xgeorge@aueb.gr

ABSTRACT

In the past few years numerous peer to peer file sharing, or more generally content distribution, systems have been designed, implemented, and evaluated via simulations, real world measurements, and mathematical analysis. Yet, only a few of them have stood the test of time and gained wide user acceptance. BitTorrent is not just one such system; it holds the lion's share among them. The reasons behind its success have been studied to a great extent with interesting results. Nevertheless, even though peer to peer content distribution remains one of the most active research areas, little progress has been made towards the study of the BitTorrent protocol, and its possible variations, in a fully controllable but realistic simulation environment. In this paper we describe and analyze a full featured and extensible implementation of BitTorrent for the OMNeT++ simulation environment. Moreover, since we aim to establish a realistic simulation platform, we show our enhancements to a conversion tool for popular Internet topology generation and a churn generator based on the analysis of real BitTorrent traces.

Keywords

BitTorrent, P2P, OMNeT++, OverSim, network simulation

1. INTRODUCTION

A major characteristic of today's Internet is that a large fraction of its traffic is due to content distribution applications. This has prompted researchers to consider redesigning the Internet so as to best support the distribution of content between publishers and subscribers of information, rather than communication between endpoints [18]. The primary means for content delivery is currently represented by *Peer-to-Peer* (P2P) applications [13] where multiple entities, or *peers*, collaborate in order to efficiently exchange content between them. The technique of *swarming*, that is, the concurrent downloading of content from multiple peers while simultaneously uploading to multiple other peers (first presented in BitTorrent [9]), has greatly contributed to this development.

BitTorrent is a P2P content distribution system, comprised of a set of network protocols between the participating peers. It utilizes a simple bartering scheme to reduce parasitic behavior such as *free-riding*, where peers only download and never upload content. When a host wants to distribute a set of files through BitTorrent, it organizes all files as a sequence of bytes, logically splits the sequence into equal size *pieces* and calculates a hash value for each piece. Then, a server is located, which is willing to host the file exchange (not the file themselves); this is the *tracker*. The content metadata and supporting information such as hash values, piece size, file size, tracker address and so on, are recorded in a file with an arbitrary name that acts as a description and summary of the content. This *metafile* can then be distributed over the Web so that search engines can match user queries with metafile names.

When presented with a metafile, a client[1] connects to the indicated tracker and asks for a list of other hosts currently participating in that particular exchange; all these hosts comprise the *swarm*. Note that the tracker does not itself participate in the swarm. Subsequently, each client constructs and maintains a bitmap with the pieces that it has (a new client initially has nothing, while the original distributor has everything). Then, each client randomly contacts other clients, called *peers* (or *seeds*, if they hold the entire file), and exchanges bitmaps with them. Based on the bitmaps, and other available data, such as path delay or bandwidth, each client can freely decide the peers to exchange pieces, in what order, and with how to talk at any given time. In general, pieces are exchanged in a *tit-for-tat* fashion, but to kickstart new clients, peers occasionally give out pieces for free.

BitTorrent owes its huge success to its ability to distribute resource consumption among the participating entities, thus avoiding the bottlenecks of centralized distribution, as well as to its ability to avoid performance deterioration and service unavailability by

[1]In the rest of this paper we follow the BitTorrent protocol specification terminology [7] which employs the term *client* for the local instance of the BitTorrent modules and the term *remote peer* for the instances operating at remote sites.

enforcing cooperation. While BitTorrent has drawn strong interest from researchers, most studies have concentrated on the performance evaluation of the protocol and its potential variations via the study of real world trace data sets [11, 13]. This approach has significant advantages with respect to the reliability of the extracted results, but it is characterized by inflexibility: there is no control over the participating peer characteristics and major protocol variations cannot be studied without first implementing and then deploying them. Moreover, the trace collection process is cumbersome and the data gathered may be incomplete. For instance, collecting information for peers behind firewalls is difficult, while gathering information about the swarm's size and structure may be hindered by the tracker protocol itself [17]. Analytical studies are even more problematic due to the highly dynamic character of BitTorrent: peers dynamically enter and leave the swarm, establish and tear down connections, decide on the preferred pieces of a file and chose to exchange data with peers or not. Simulation appears to be a more promising alternative, as it allows fast prototyping, provides the ability to perform large scale experiments, and offers a common reference platform for experimentation. Nonetheless, current BitTorrent simulators either consider coarse-grained representations of the underlying network, thus reducing the realism of the simulation, or omit many important features of the BitTorrent protocols.

In this paper, we present a full featured and extensible implementation of the BitTorrent protocol for the OMNeT++ simulation environment. We chose this platform due to its simplicity, its high degree of modularity and the availability of several protocol implementations ranging from a complete TCP/IP protocol stack (provided by the INET framework) to a large set of overlay protocols (encapsulated in OverSim [1]). In order to increase the degree of realism in our simulation environment, we also present our enhancements on a *topology conversion tool* that allows our platform to use Internet like topologies generated by the popular *Georgia Tech Internet Topology Model* (GT-ITM) [22]. In the same vein, we present a *churn generator model* that inserts BitTorrent nodes in a network topology by following an arrival process derived from the analysis of actual BitTorrent traces [11].

The rest of the paper is organized as follows. In Section 2 we provide a detailed description of the BitTorrent protocols, focusing on the features that we implemented. In Section 3 we present the architecture of our simulator implementation with respect to module structure and code organization. Section 4 details the procedure for establishing realistic simulation scenarios, including the GT-ITM topology conversion tool and the churn generator module. In Section 5 we discuss the limitations of the other available simulators that prompted our work. Finally, we discuss our future work plans in Section 6 and we conclude in Section 7.

2. THE BITTORRENT PROTOCOLS

One of the difficulties faced during the implementation of the BitTorrent modules, was the lack of an official protocol specification. Despite the immense embrace of BitTorrent from the user and research communities, no formal protocol specification has been drawn up yet. The only authoritative document available, describes the entities involved in the protocols, the basic concepts, and the rudimentary transactions among them, but it lacks behavioral and implementation details [7]. In effect, we had to resort to the unofficial BitTorrent Protocol Specification [6], which nevertheless does not constitute a formal and unambiguous source of information. In fact, several attributes of the protocols appear to be under dispute. In the remainder of this section we provide a detailed description of the protocols implemented, clarifying at the same time our approach in all cases of dispute.

2.1 The Tracker Protocol

Typically, the distribution of a new file[2] with BitTorrent starts by publishing a `.torrent` metafile; this metafile is distributed to peers using an out-of-band channel, usually by posting the metafile on a web page. Among other information, a `.torrent` metafile contains the tracker address, file size, piece size, and the hashes for the file pieces. Trackers are responsible for helping peers discover each other so as to form a swarm. In most cases, each `.torrent` metafile is served by a single tracker, but recent extensions to the protocol (not implemented by us) allow multiple trackers for one file or even no trackers at all [8]; the *trackerless* approach employs *Distributed Hash Tables* (DHTs) for decentralized peer discovery.

Clients communicate with the tracker via a simple text-based protocol, layered on top of HTTP/HTTPS, using the tracker's URL stored inside the metafile. During the download phase, each client communicates with the tracker and publishes its progress (in terms of total bytes downloaded/uploaded), as well as its *contact details* (e.g., IP address, TCP port, identification info). These parameters are passed from the client to the tracker using the standard HTTP GET method[2]. Note that most of the information *announced* by the client is for statistical purposes; only the IP address and TCP port of a client are crucial for the tracker. After such message, called a *tracker request*, the tracker randomly selects a set of peers and returns their contact details in a *bencoded* dictionary [6]. Since each tracker request provides the contact details of a client to the tracker, the tracker can return such details back in its replies. In this manner, over time the peers discover increasing subsets of the swarm.

2.2 The Peer-wire Protocol

The peer-wire protocol provides the core BitTorrent functionality: interaction with remote peers. In the following we first present an overview of the implemented peer-wire protocol and then proceed with the details of its operation, focusing on the most important features available in our implementation.

2.2.1 Protocol overview

After contacting the tracker, a client attempts to establish TCP connections with the peers listed in the tracker response. Upon connection establishment, the two peers exchange HANDSHAKE messages in order to verify their peer identities and ensure that they are interested in the same torrent metafile. This handshake is then followed by an exchange of BITFIELD messages that contain the *bitfield* of each client, which is a bitmap denoting the availability of each piece at the client. Based on that information a client can determine whether it is interested in one or more pieces of the remote peer. Note that this exchange is optional when the client has no pieces, since it would result in the exchange of useless information, and is therefore avoided in our implementation.

By following the above procedure over multiple[3] peer connections, a client collects information regarding the availability of the pieces that it is still missing in the subset of the swarm that has explored using that information. Based on this information, it then decides which pieces to request from each peer. In general, if a peer does not hold any pieces that the client does not already hold, a NOT INTERESTED message is sent to that peer to indicate the

[2]Since BitTorrent organizes the set of files to be distributed as a linear sequence of bytes, akin to a single file, we use the terms file and files interchangeably throughout the rest of the paper.

[3]The number of connections to establish is an important protocol parameter discussed in Section 2.2.2

lack of interest for its data. At the beginning of a connection, peers are assumed not to be interested in each other's pieces by default.

Although at this stage a client knows the peers interested in, it cannot make any requests yet as data cannot be exchanged until the remote peer actively permits this by sending an UNCHOKE message. This means that each client is by default blocked, or, in BitTorrent parlance, *choked* by the corresponding remote peer. The decision to choke or unchoke a client is made based on several criteria embodied in the *choking algorithm* [6] of the protocol:

Reciprocation: Peers unchoke the clients providing the best upload rates.

TCP performance: TCP behaves better when the number of simultaneous uploads is capped.

Fibrillation avoidance: Frequent choking/unchoking causes data transfer interruptions that deteriorate protocol performance.

Optimistic unchoking: New peers are occasionally unchoked so as to discover potentially better connections. Moreover, peers are thus given the chance to acquire their first pieces.

When a client is unchoked by a peer, it starts sending REQUEST messages, each asking for a specific *block* of the selected piece. The peer sends back the requested data using PIECE messages. Upon completing the downloading of a piece, a client informs via HAVE messages the peers that it has established connections to. These peers update the bitfield for that client and may then, potentially, express their interest for that piece with an INTERESTED message.

2.2.2 *Connections*

A client learns about other peers by employing the Tracker protocol and parsing the peer list returned by the tracker. The client then joins the swarm by establishing connections with some peers. However, as noted in [6], each connection incurs an increase in signaling traffic, especially for bitfield maintenance via the exchange of HAVE messages. Thus, in our implementation we provide configurable lower and upper bounds for the number of established connections, using the MINNUMCONNECTIONS and MAXNUMCONNECTIONS configuration parameters (see Table 1).

2.2.3 *Piece downloading strategy*

The piece downloading strategy refers to the policy followed in the selection of the pieces that will be requested from a peer. It is an important aspect of BitTorrent as it heavily affects the diversity of the pieces available at each peer. A low degree of diversity would result in low interest for a peer's pieces, thus causing degraded application performance. We have implemented the two most prevalent piece downloading strategies, *Rarest First* and *Random First*. Based on the information gathered during the BITFIELD – HAVE message exchange, the *Rarest First* strategy selects those pieces that appear less frequently in a client's set of connected peers. This selection is randomized among several of the less common pieces, according to the RAREST LIST SIZE configuration parameter (see Table 1), in order to avoid multiple peers converging on the same piece. This way, peers download pieces that most other peers probably want, therefore facilitating data exchange. However, rare pieces are present only in a few peers, and it is possible that downloading them may be interrupted due to a choking decision. Clients with no pieces in their possession would therefore have to wait for an optimistic unchoking event from a peer holding the same rare piece in order to continue downloading. The *Random First* strategy avoids this problem by selecting a random piece which is more likely to be available from multiple peers, so that a choking decision would not have such an adverse effect.

2.2.4 *Queueing*

As mentioned above, REQUEST messages refer to specific blocks of a piece. This facilitates fine-grained data exchange by enabling the queuing of data requests. As common piece sizes vary from 256 KB to 1 MB [6] or even larger, per piece requests would result in a multitude of redundant packet retransmissions in the event of a choking decision during piece transfer. A window-based queuing mechanism is employed for these requests, otherwise propagation delays would dominate the total download time.

Since the exact nature of the queueing policy is under dispute, we implemented a generic queueing mechanism in which the user can specify the exact size of the queue. In this mechanism a client may send to a peer up to REQUEST QUEUE LENGTH (see Table 1) REQUEST messages for blocks. Once a PIECE message has been received, the client may send the next REQUEST message. Once a piece has been requested in its entirety, if the request queue is not full, the client chooses another desired piece from that peer's bitfield according to the piece selection strategy (see Section 2.2.3) and starts sending REQUEST messages for its blocks.

2.2.5 *Choking algorithm*

For the choking algorithm we followed the guidelines presented in Section 2.2, along with the ability to tune the choking algorithm as desired. The user is able to select appropriate values for the time between (optimistic) choking decisions, using the CHOKING INTERVAL and OPTUNCHOKING INTERVAL configuration parameters, and the maximum number of (optimistically) unchoked peers, using the DOWNLOADERS and OPTUNCHOKEDPEERS configuration parameters (see Table 1). In order to facilitate the deployment of content providers with advanced seeding capabilities (see Section 4.2) we have enabled the above parameters to be separately configured for such nodes via the SEEDERDOWNLOADERS and SEEDEROPTUNCHOKEDPEERS parameters. The configuration parameter NEWLYCONNECTEDOPTUNCHOKEDPROB is the probability that the most recently connected peer will be preferred over previously connected peers in an optimistic unchoke decision.

2.2.6 *Super Seeding*

The *super seed* feature is especially useful for content distribution as it helps the initial seeder of a file avoid excessive bandwidth consumption while fostering data exchange between participating peers. A super seeder does not inform its peers that it has all pieces available, masquerading as an ordinary client. Initially it pretends to possess no pieces and only later informs it about the availability of an individual piece with a HAVE message, as if it had just completed downloading it. The seeder either selects a piece it has never uploaded before or, if all pieces have already been uploaded at least once, a piece that has been uploaded only a few times. After the piece has been downloaded by the peer, the seeder will not inform it of another one until it sees this piece marked as available in the bitfield of another peer, implying that the first peer has in turn uploaded this piece.

Our module implements this feature at all clients, but only enables it at the initial seeder via the SUPER SEED MODE configuration parameter (see Table 1), since super seeding is not recommended for ordinary peers [6]. These peers provide their pieces as requested by their clients, acting as regular seeders after downloading all pieces. The duration of this seeding phase can be set via the TIMETOSEED configuration parameter (see Table 1).

2.2.7 *Endgame mode*

Parameter	Default Value
file size (MB)	700
piece size (KB)	256
block size (KB)	16
DHT port	-1
pstr	BitTorrent protocol
pstrlen	19
keep alive (sec)	120
have supression	true
choking interval (sec)	10
downloaders	4
optUnchokedPeers	1
optUnchoking interval (sec)	30
seederDownloaders	4
seederOptUnchokedPeers	1
rarest list size	5
minNumConnections	30
maxNumConnections	55
timeToSeed (sec)	0
request queue length	5
super seed mode	false
end game mode	true
maxNumEmptyTrackerResponses	5
newlyConnectedOptUnchokeProb	0.75
downloadRateSamplingDuration (sec)	20

Table 1: Peer-wire protocol parameters.

The *endgame* mode addresses the problem of slow transfers for the last data blocks of an exchange, since at that stage most pieces have been downloaded, therefore the degree of parallelization is low. In this mode the client sends REQUEST messages for each missing block to all peers that are not choking it, as opposed to a single peer. While this is not clarified in the specification [6], our implementation does not send these messages to all peers in its current peer set since a peer choking the client will simply discard the request. Another unclarified aspect of the endgame mode regards the entry condition. In our implementation, the client enters this mode when the number of missing blocks equals the number of requested blocks, meaning that all missing blocks have been requested. This feature can be turned on/off using the END GAME MODE configuration parameter (see Table 1).

3. IMPLEMENTATION

The architectural design approach we followed resembles the philosophy of the INET framework upon which we built our modules[4]. We opted for the following goals: *simplicity* for eliminating simulation complexity, *modularity* for supporting abstractions and add-ons, and *extensibility* for encouraging the model's evolution by community contributions. Most of the implementation specific characteristics (those features left open in the specification), were encapsulated in order to produce a simulation package that is concrete and extensible. Thus, simulations can run without touching the source code, by simply editing the corresponding configuration files of OMNeT++ (i.e., `.ini` files), while at the same time maintaining the ability to change many aspects of the model behavior. Furthermore, in order to facilitate the evolution of the provided modules, we have modularized the source code of most important peer-wire protocol features such as the choking algorithm and the selection of the peer(s) for optimistic unchoking (see Section 2.2.5), the piece downloading strategy (see Section 2.2.3), the entry condition for the endgame mode (see Section 2.2.7), and so forth. Hence, different algorithms may be easily implemented by simply redefining the respective methods.

[4]The source code of our BitTorrent implementation is available at http://mm.aueb.gr/˜katsaros/bittorrent.tar.gz

Our BitTorrent model consists of three modules, TRACKER, TRACKER CLIENT, and PEER-WIRE modules as shown in Figure 1. As their names suggest, the first module provides the functionality of the tracker as described in Section 2.1, the second module is responsible for communicating with the tracker on behalf of a client and the third module provides the functionality of the peer-wire protocol as described in Section 2.2. In order to facilitate the deployment of BitTorrent simulation scenarios, we created separate end host compound modules for each BitTorrent entity, namely BTHOST, BTHOSTSEEDER and TRACKER. All these compound modules were derived from INET's STANDARDHOST module; individual protocol modules can also operate as simple STANDARDHOST submodules. A detailed description of the simulation scenario deployment procedure is provided in Section 4.

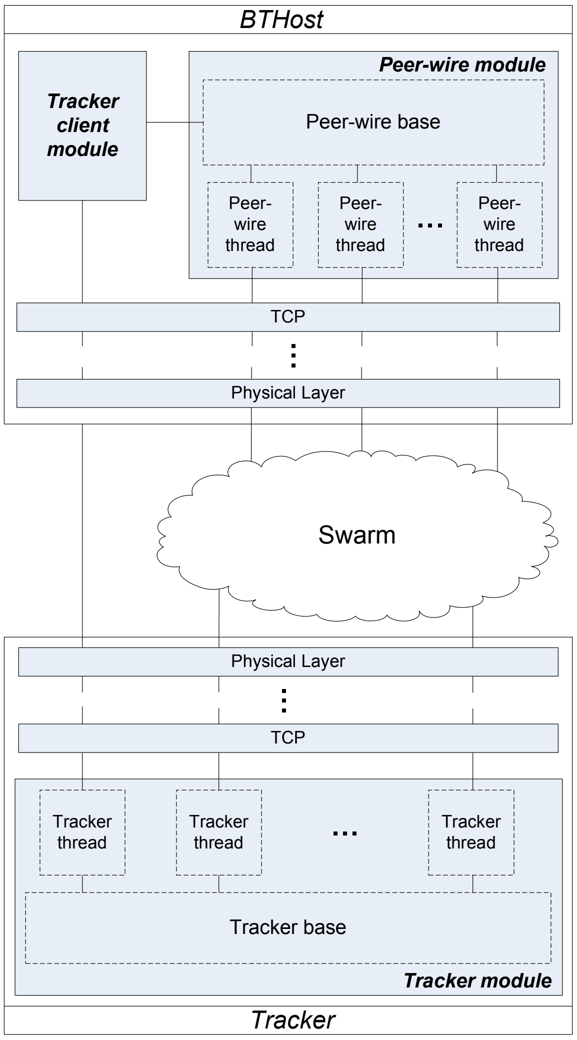

Figure 1: BitTorrent module architecture.

As parts of an INET framework application, all modules heavily rely on INET's TCP application models. The first design decision we faced was about the TCP server models we would employ for the tracker and the peer-wire protocol, given the two alternatives provided by the INET framework, namely the TCPSRVHOSTAPP and the TCPGENERICSRVAPP models. The former dynamically creates and launches a new *thread*[5] object to handle each incoming connection. The latter also accepts multiple connections but handles them in a centralized fashion. The first approach was regarded as closer to the symmetric character of a peer-to-peer protocol such as BitTorrent, while it also saved us from the burden of multi-peer centralized state maintenance required by the second approach.

3.1 The Tracker Protocol

The implementation of the tracker protocol consists of two principal modules: BTTRACKERBASE and BTTRACKERCLIENTBASE. The former is the server module, which is part of the Tracker, while the latter is the client module, which is part of the BitTorrent application. The communication between these modules is carried out through the BTTRACKERMSGANNOUNCE and BTTRACKERMSGRESPONSE messages, both derived from the CMESSAGE class. The former class implements the client announce messages and includes all necessary fields, with their corresponding semantics, while the latter class encodes the tracker's responses; both classes are fully compliant with the specification [6].

The Tracker functionality is implemented in the BTTRACKERBASE module, which is derived from the INET TCPSRVHOSTAPP, as a multi-threaded network application. Upon each successful connection to the Tracker, a thread is generated at run time in order to drive the session between the tracker and the peer. The BTTRACKERCLIENTHANDLERBASE module, stemmed from the INET TCPSERVERTHREADBASE, is used to encapsulate and modularize the details of tracker-to-peer communication such as message exchanges, input validation, reply construction and so on, while BTTRACKERBASE handles the underlying low-level operations such as timer handling, and message dispatching. Similarly, the client part is implemented in BTTRACKERCLIENTBASE, derived from the INET TCPGENERICCLIAPPBASE, including the functionality required to retrieve tracker responses and feed the BitTorrent application with the received information (i.e., the contact information of other peers). For both the tracker server and client we implemented the entire set of parameters described in [6]. The key configuration parameters of both modules, with their corresponding default values, are illustrated below in Table 2 (server) and Table 3 (client).

Parameter	Default Value
alwaysSendTrackerId	false
compactSupport	true
maxPeersInReply	50
announceInterval (sec)	30
cleanupInterval (sec)	60

Table 2: Tracker Server parameters.

Parameter	Default Value
connectGiveUp	3
reconnectInterval (sec)	2.0
sessionTimeout (sec)	30.0
infoHash	nil
compact	false
noPeerId	false
numWant	20
key	nil

Table 3: Tracker Client parameters.

3.2 The Peer-wire Protocol

The implementation of the peer-wire protocol consists of two principal modules: BTPEERWIREBASE and BTPEERWIRECLIENTHANDLERBASE. The main coordination point for the BitTorrent client is the BTPEERWIREBASE module, derived from the INET TCPSRVHOSTAPP. This module provides the following functionalities:

1. Retrieving tracker protocol information, that is, communicating with the tracker client module to retrieve the peer set information provided by the tracker.
2. Handling the connection establishment policy.
3. Implementing the piece selection strategies. This functionality involves maintaining state on:
   - Data availability in the client and throughout the part of the swarm that we are connected to.
   - A client's current data exchanges, that is, which blocks have been requested.
4. Informing peers about the availability of new pieces, both in the normal and in super seeding mode.
5. Applying the choking algorithm.
6. Coordinating the endgame mode.

The BTPEERWIRECLIENTHANDLERBASE class, derived from the INET TCPSERVERTHREADBASE, is responsible for handling communications with a single peer. This means that all peer-wire protocol message exchanges between peers (see Section 2.2.1) are handled by instances of this class. The coordination of individual thread (e.g., assuring that a certain block is not requested from more than one peers, except while in the endgame mode), is performed by the BTPEERWIREBASE either with direct method calls or with message exchanges.

Apart from the above classes, our implementation is also based on two utility classes, namely BITFIELD and BTUTILS. The former is used to represent a peer's bitmap, including block information[6], and provides all necessary handling functions, such as initialization, updates and queries. The latter is used for handling protocol state information as described above.

3.2.1 Connection establishment

In the TCPSRVHOSTAPP model, a TCPSRVHOSTAPP object creates a new socket for each *passive* connection which is handled by a new TCPSERVERTHREADBASE instance. In order for our implementation to reflect the completely symmetric character of peer connections, we decided to extend this model by also incorporating the functionality of the client side. That is, our BTPEERWIREBASE class allows each new *active* connection to be handled by a separate BTPEERWIRECLIENTHANDLERBASE thread

[5]The term *thread* is used to denote the individual connection handler rather than an actual Operating System entity.

[6]This information is not included in the BITFIELD messages.

object. Hence, when a client decides to establish a TCP connection with a peer it creates a new socket, establishes the connection and creates a new BTPeerWireClientHandlerBase instance to be set as the callback object of the socket. In effect, symmetry is achieved since two identical thread objects handle the two ends of the connection. Code structure is therefore simplified since the peer-wire protocol functionality is incorporated in a single event-based class.

Furthermore, as discussed in Section 2.2.2, our implementation was designed to limit the number of established connections per peer. To achieve this, connections are not initiated by a client unless the current number of established connections lies below the maximum number of allowed TCP connections. However, this is not enough, since a peer may initiate a connection with a client that has already reached the maximum number of allowed TCP connections. In this case, the connection should be refused by the client. This is accomplished by closing/opening our TCP serverSocket, subject to the connection establishment policy.

3.3 Statistics

The collection of application and protocol statistics is facilitated by BTStatistics, a simple module responsible for collecting and aggregating statistics. The set of currently available statistics includes the *download duration* for those peers that have managed to download the desired file in its entirety and the *number of downloaded blocks* for those peers that have failed (i.e., those peers that cannot find a peer to provide their missing blocks/pieces). A peer is considered to have failed if it receives an empty tracker response for maxNumEmptyTrackerResponses times before downloading completes (see Table 1). Furthermore, the *number of distinct data providers* and the *number of blocks downloaded from seeder*, are also included in the statistics collected for each peer. The former can be used to examine whether the downloading process is spread across the swarm, while the latter can assist in revealing the tendency of users to download directly from the initial seeder (i.e., the content provider) subject to several scenario specific parameters such as the arrival time of peers (see Section 4.2), which is also recorded, the available bandwidth at the content provider, the seeding policy of other peers, etc.

For all the above metrics, both individual measurements (*vector* statistics) and aggregated values (*scalar* statistics are recorded, accommodating coarse and fine grained analysis of the collected data. We plan to enrich this set with statistics on the download rate achieved by the peers during the download process, the amount of signaling traffic, and so on.

4. CREATING SIMULATION SCENARIOS

As already discussed above, our BitTorrent implementation was developed as a stand-alone INET framework application, therefore, in order to run a BitTorrent simulation, a network topology must be provided and the appropriate modules (Tracker, Tracker Client and Peer-wire) must be loaded as submodules of the compound modules representing the peers and the tracker. Although this procedure is sufficient for testing the modules, it is cumbersome to use when constructing realistic scenarios:

- Constructing large-scale realistic network topologies in OMNeT++/INET is a complex procedure that requires careful handling of node module placement and interconnection.

- Randomly introducing clients into the simulation, both topologically and chronologically, through network description files fails to capture the dynamics of such applications.

These considerations indicate that there is a need for globally controlled network-wide *dynamic module loading* in the construction of the simulation scenario. Hence, we turned to the OverSim overlay simulation framework [1], which provides several of the features required to establish realistic dynamic simulation scenarios. It must be stressed however that our BitTorrent implementation is not OverSim dependent: it can simply optionally employ several of the features provided by OverSim in order to establish realistic simulation scenarios. In the following sections we present the OverSim features that we exploited along with our enhancements.

4.1 Topologies

One of our concerns in creating a realistic simulation platform for BitTorrent is related to the underlying network topologies. OverSim provides various underlying network structures, both simple (SimpleUnderlay) and complicated (IPv4Underlay). However, both models present significant limitations in representing a realistic network substrate. The lack of protocol functionality and step-by-step routing in the SimpleUnderlay model turned our attention to the more realistic IPv4Underlay model. In [14] we addressed two important limitations of this model. First, the model only provides the distinction between backbone and access routers neglecting more complex structures imposed by the existence of multiple autonomous administrative domains. Second, the model provides no support for routing policy weights. Both problems were addressed by an extension to the BRITE topology generator [16] export tool [20] that enables full support of the very popular *Georgia Tech Internet Topology Model* (GT-ITM) [22] topologies within the IPv4Underlay model, including the employment of a weighted shortest path algorithm.

Despite these important enhancements, the resulting GT-ITM based IPv4Underlay model retains two more, significant limitations. The first is revealed when considering the locality properties (with respect to the consumption of ISP-specific resources) of data exchanges in P2P content distribution applications, such as BitTorrent. It has been shown that BitTorrent's network-agnostic peer-wire protocol has an adverse impact on capacity related ISP costs by allowing downloads from peers residing in external domains even though the desired data are already present locally [13]. This has triggered several research efforts (e.g., [21],[5]) which would benefit from a simulator providing the flexibility to study ISP level aspects of the protocol performance. Hence, while OverSim's IPv4Underlay model provides no access to such information, we have further enhanced our topology conversion tool to also preserve the unique Autonomous System numbers [12] produced by BRITE and to export them to a separate configuration file so that each router, as well as each attached end host, can be assigned the corresponding AS number. Direct access to this information is provided to the simulation programmer, facilitating the investigation of the aforementioned locality properties and protocol inefficiencies, as well as the implementation of location-aware schemes [5]. Our future work plans in that area include the extraction of more information regarding this aspect of the BitTorrent protocol (see Section 6).

Second, OverSim does not make any distinction between the uplink and downlink characteristics of access links (e.g bandwidth). However, this distinction is important in providing a realistic networking environment, since current typical access technologies, such as ADSL, do present this asymmetry. This issue is further signified by the fact that bandwidth heterogeneity results in systematic unfairness of the peer-wire protocol download rate for is based on the tit-for-tat mechanism [3]. Hence, we have further enhanced OverSim's IPv4Underlay model to support a range of

Uplink (Mbps)	**Downlink** (Mbps)	**Fraction**
1	4	0.20
1	8	0.40
2	16	0.25
2	24	0.15

Table 4: Bandwidth distribution of access links.

channel [7] characteristics for the two directions of each access link. Specifically, the simulation programmer is able to specify different channel options for the uplink and downlink of each access link type, along with the fraction of the total access links across the entire network that each channel type is assigned to. For example, in the measurements presented below, we have set the values presented in Table 4. The fraction values indicate that 40% of the participating peers can download data with a maximum rate of 8 Mbps while they can upload with a maximum rate of 1 Mbps.

4.2 Host Deployment

Having established a realistic network topology, the next step is to deploy the BitTorrent entities on it, that is, the tracker, the initial seeder and the peers. For the tracker, to avoid coupling the network topology description file with the application, we extended the OverSim IPV4UNDERLAYCONFIGURATOR module to dynamically introduce an end host in the network serving as the BitTorrent tracker.

Regarding the initial seeder, we implemented a separate deployment scheme, since in many realistic scenarios the initial seeder has different characteristics from ordinary peers. For instance, the content may be a new Linux distribution (e.g., an ISO image file), hosted by a dedicated server with a high capacity access link, with ordinary peers participating in the swarm through ADSL links. Protocol parameters may also be altered to achieve differentiated behavior between the initial seeder and the peers. For example, the initial seeder may optimistically unchoke multiple peers to speedup the distribution of the offered file. This was again achieved by extending IPV4UNDERLAYCONFIGURATOR and ACCESSNET modules' functionality and employing a separate host description file for the initial seeder, the BTHOSTSEEDER. Note that our extensions check whether the scenario is BitTorrent related before proceeding to deploy an initial seeder or a tracker, thus preserving the base functionality of the affected modules.

Unlike the tracker and the initial seeder, peers need to be randomly introduced into the network both in a topological and in a chronological sense (i.e., they need to be placed at random nodes in random time points). The churn models provided by the OverSim platform, together with the underlying IPV4UNDERLAY configuration mechanism, constitute a flexible mechanism for dynamically deploying peers in the network. However, the churn models available in OverSim were not designed to reflect the arrival processes of real applications. Instead, they provide a generic mechanism for the arrival process, and several distributions describing the duration of a peer's presence in the network. Since in BitTorrent this duration depends on protocol operation rather than on a predetermined distribution, we focused on the arrival process. Using the OverSim churn generator mechanism, we implemented the BITTORRENTCHURN model that reproduces the arrival process of BitTorrent clients presented in [11]. In this study, based on the analysis of BitTorrent user traces, it was observed that the peer arrival rate for a torrent follows an exponential decreasing rule with time t:

$$\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}},$$

where λ_0 is the initial arrival rate when the torrent starts and τ denotes the file popularity. Based on this distribution it can be shown that $N_{all} = \lambda_0 \tau$, where N_{all} is the total population size. Hence, by retrieving values for N_{all} and λ_0 from the configuration files, our model can generate random arrival times for each BitTorrent peer.

[7]At this point we adopt the OMNeT++ definition of a *channel* which includes the data rate, error rate, and propagation delay characteristics of a link.

5. RELATED WORK

Simulating the operation of BitTorrent is a difficult task due to the inherent protocol complexity and the lack of concrete specifications. The multitude of possible policies, such as those for piece selection and choking decisions, as well as parameter values, such as the choking interval and the number of unchoked peers, creates a highly dynamic environment. Hence, in order to isolate and focus on important protocol aspects, works such as [3, 4, 15], ignore the influence of the underlying protocols and focus on the *application logic*. However, this design decision obviously incurs a non negligible degree of inaccuracy. TCP dynamics, propagation delays and the potential queuing of packets in routers are important factors that can affect, for example, the perceived download rate in a client and therefore alter its choking decisions. Being written as an INET Framework application, our implementation avoids this situation by providing almost all features specified in [6] and by operating on top of full-fledged simulation platform.

To the best of our knowledge, there is only one packet-level BitTorrent simulation module available [10], implemented for ns-2 [19]. While this implementation shares our goal of providing a realistic simulation environment, our implementation provides several additional features, such as the entire Tracker protocol as well as the endgame mode of the peer-wire protocol. Furthermore, our implementation allows fine grained tuning of the protocols by providing several configuration parameters not available in [10], such as OPTUNCHOKING INTERVAL and NEWLYCONNECTEDOPTUNCHOKEDPROB (see Table 1).

Finally, we note that our implementation is not the first attempt to incorporate BitTorrent in the OMNeT++ simulation platform. A swarming-based simulation module is presented in [15], but it only provides a bare-bones subset of the BitTorrent protocol features, since it totally lacks the essential tit-for-tat mechanism. This module was also developed using trivial network topologies with dedicated non-TCP connections among all pairs of peers, unlike our module which fully exploits realistic, large-scale topologies build on top of the INET framework.

6. FUTURE WORK

Our plans for future work primarily focus on enriching the current set of extracted statistics. To this end, we first aim at keeping record of the achieved download rate of each peer during its participation in the swarm. Our intention is to make this information, as well as the currently provided statistics, further categorizable with respect to the access link capabilities of the peers so as to allow the fine grained analysis of the derived measurements. In the same vein, and by taking advantage of the enhanced topology model employed (see section 4.1), we plan to provide support for the extraction of topology-aware results i.e. results regarding specific areas of the network such as specific access networks. This will involve both per peer statistics (e.g. download time) and per area statistics (e.g. volume of data exchanged with neighboring access networks).

Furthermore, it is in our plans to enable the separate extraction of statistics for the operation of the initial seeders.

Regarding the creation of simulation scenarios (see Section 4), we intend to provide support for multiple initial seeders and to further enhance the tuning of the peer-wire protocol on these nodes by providing control over several parameters such as the choking interval and the number of optimistically unchoked peers. Such features, in combination with the seeder-specific statistics, are expected to enable the fine-grained investigation of the protocol performance from the perspective of content providers. Moreover, by taking advantage of the rich suite of overlay protocols provided by OverSim [1], we currently investigate the design space for the implementation of the DHT-based trackerless BitTorrent extension.

Finally, apart from extending the functionality of the presented modules, our goal is to continue the presented evaluation with an extended set of experiments regarding both the resource requirements of the presented simulation environment and the performance of BitTorrent 's peer-wire protocol. Our first steps towards this direction include the investigation of the impact of the file size on both the memory and the computation overhead, and the investigation of the block size impact on the peer-wire protocol performance.

7. CONCLUSIONS

In this paper we have presented an implementation of the BitTorrent set of protocols for the OMNeT++ simulation environment. Our main target was to produce a realistic simulation environment that will enable the detailed evaluation of the protocol in fully controllable conditions. Towards this direction we have faithfully implemented the only available protocol specification, trying to make our module resemble an actual BitTorrent application implementation. Furthermore, we created a set of tools, which enable the construction of realistic simulation scenarios that can capture the properties of the Internet like network topologies and the real world deployment dynamics of BitTorrent participants. Our goal is to enable simulation scenarios to be created where clients access the network over both symmetric and asymmetric links with various characteristics.

Acknowledgements

The work reported in this paper was partially supported by the ICT PSIRP project under contract ICT-2007-216173.

8. REFERENCES

[1] I. Baumgart, B. Heep, and S. Krause. OverSim: A flexible overlay network simulation framework. In *Proc. of the IEEE Global Internet Symposium*, pages 79–84, 2007.

[2] T. Berners-Lee, L. Masinter, and M. M. (eds). Uniform Resource Locators (URL). Internet Request For Comments, December 1994. RFC 1738.

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

[4] A. Bharambe, C. Herley, and V. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *Proc. of the IEEE INFOCOM*, pages 1–12, 2006.

[5] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving traffic locality in BitTorrent via biased neighbor selection. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[6] BitTorrent development community. BitTorrent Protocol Specification v1.0. http://wiki.theory.org/BitTorrentSpecification.

[7] BitTorrent.org. BitTorrent Protocol Specification. http://www.bittorrent.org/beps/bep_0003.html.

[8] BitTorrent.org. Torrent File Extensions. http://www.bittorrent.org/beps/bep_0005.html.

[9] B. Cohen. Incentives build robustness in BitTorrent. In *Proc. of the Workshop on the Economics of Peer-to-Peer Systems*, pages 116–121, June 2003.

[10] K. Eger, T. Hoßfeld, A. Binzenhöfer, and G. Kunzmann. Efficient simulation of large-scale P2P networks: Packet-level vs. flow-level simulations. In *Proc. of the Workshop on the Use of P2P, GRID and Agents for the Development of Content Networks*, pages 9–16, 2007.

[11] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. A performance study of BitTorrent-like peer-to-peer systems. *IEEE Journal on Selected Areas in Communications*, 25(1):155–169, 2007.

[12] IANA. Autonomous system numbers. http://www.iana.org/assignments/as-numbers.

[13] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet service providers fear peer-assisted content distribution? In *Proc. of the Internet Measurement Conference*, pages 63–76, 2005.

[14] K. Katsaros, N. Bartsotas, and G. Xylomenos. Router assisted overlay multicast. In *Proc. of the Euro-NF Conference on Next Generation Internet Networks (NGI)*, 2009 (to appear).

[15] P. Korathota. P2P swarming protocol simulation. http://me55enger.net/swarm/.

[16] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: an approach to universal topology generation. In *Proc. of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 346–353, 2001.

[17] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proc. of the International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 205–216, 2005.

[18] PSIRP Project Team. PSIRP project home page. http://www.psirp.org.

[19] UCB/LBNL/VINT. Network Simulator - ns (version 2). http://www.isi.edu/nsnam/.

[20] A. Varga. OMNeT++ export for BRITE 2.1. http://www.omnetpp.org/filemgmt/singlefile.php?lid=5.

[21] H. Xie, R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4P: provider portal for applications. In *Proc. of the ACM SIGCOMM*, pages 351–362, New York, NY, USA, 2008.

[22] E. Zegura, K. Calvert, , and S. Bhattacharjee. How to model an internetwork. In *Proc. of the IEEE INFOCOM*, volume 2, pages 594–602, 1996.