A BitTorrent Module for the OMNeT++ Simulator

Konstantinos Katsaros, Vasileios P. Kemerlis, Charilaos Stais and George Xylomenos
Mobile Multimedia Laboratory, Department of Informatics
Athens University of Economics and Business, Athens, Greece
{*ntinos, stais, xgeorge*}*@aueb.gr*
Network Security Laboratory, Computer Science Department
Columbia University, New York, NY
vpk@cs.columbia.edu

***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 a popular Internet topology generator and a churn generator based on the analysis of real BitTorrent traces. Finally we present the results from the evaluation of our prototype implementation regarding resource demands under different simulation scenarios.**

I. INTRODUCTION

One of the major characteristics 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 content delivery between *publishers* and *subscribers* of information, rather than communication between endpoints [1]. The primary means of content dissemination are currently represented by *Peer-to-Peer* (P2P) applications [2], where multiple entities (*peers*) collaborate in order to efficiently exchange content. The technique of *swarming*, which is the concurrent downloading of content from multiple peers while simultaneously uploading to multiple other peers (first presented in BitTorrent [3]), has greatly contributed to this development.

BitTorrent is a P2P content distribution system, comprised of a set of network protocols for realizing communication between the participating entities. It utilizes a simple bartering scheme for reducing parasitic behavior such as *free-riding*, where peers only download and never upload content. Each time a host wants to distribute a set of files through BitTorrent, it organizes all of them as a sequence of bytes, it logically splits the sequence into equal size *pieces* and calculates a hash value for each piece. Then, a server that is willing to host the file exchange (not the file itself) is located; this is the *tracker*. The content metadata and supporting information such as hash values, piece size, file size, tracker address and so forth, 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 data.

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 and exchanges bitmaps with them. Based on the bitmaps and other available data, such as path delay or bandwidth, each client can freely select the peers it will exchange pieces with. In general, pieces are exchanged in a *tit-for-tat* fashion, but for bootstrapping purposes, peers occasionally give pieces for free.

The success of BitTorrent stems from two key characteristics. First, to its ability to distribute resource consumption among the participating entities, thus avoiding the bottlenecks of centralized distribution. Second, to its aptitude to avoid performance deterioration and service unavailability by 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 [5], [2]. 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 might be hindered by the tracker protocol itself [6]. 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

[1]In this paper we follow the BitTorrent protocol specification terminology [4], 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.

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 [7] 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 [8] module). 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) [9]. In the same vein, we present a *churn generator* that activates BitTorrent nodes in a network topology by following an arrival process derived from the analysis of actual BitTorrent traces [5].

The rest of the paper is organized as follows. In Section II we provide a detailed description of the BitTorrent protocols, focusing on the features that we implemented. In Section III we present the architecture of our simulator with respect to module structure and code organization. Section IV details the procedure for establishing realistic simulation scenarios, including the GT-ITM topology conversion tool and the churn generator module. In Section V we provide some sample results from the simulator, including measurements of its processing and memory requirements. In Section VI we discuss the limitations of the other available simulators that prompted our work. Finally, we discuss our future work plans in Section VII and we conclude in Section VIII.

II. 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 both the user and research community, 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 [4]. In effect, we had to resort to the unofficial *BitTorrent Protocol Specification* [10], 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.

A. The Tracker Protocol

As we already mentioned, 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 it on a web page. Trackers are responsible for aiding peers to discover each other and form a swarm. In most cases, each metafile is served by a single tracker, but recent extensions to the protocol (not implemented by us) allow multiple trackers for each file or even no trackers at all [11]. This is the *trackerless* approach, which 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[12]. 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. After each such message, called a *tracker request*, the tracker randomly selects a set of peers and returns their contact details in a *bencoded* dictionary [10]. This is the *tracker response*. The tracker discovers these details via the tracker requests made by the clients. In this manner, over time the peers discover increasing subsets of the swarm.

B. The Peer-wire Protocol

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

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 each other's identity and ensure that they are interested in the same torrent. This handshake is then followed by an exchange of BITFIELD messages that contain the *bitfield* of each client (*i.e.*, the 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 peer connections[3], a client collects information regarding the availability of the pieces that it is still missing in the subset of the swarm explored thus far. Based on this, 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 have, a

[2]Since BitTorrent organizes the set of files to be distributed as a linear sequence of bytes, similarly 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 discussed in Section II-B.2.

NOT INTERESTED message is sent to that peer to indicate the lack of interest for its data. At the beginning of a connection, peers are assumed not to be interested in each other's pieces.

Although at this stage a client knows the peers it is interested in, it cannot make any requests yet as data are not exchanged until the remote peer actively authorizes this by sending an UNCHOKE message. This implies that each client is initially blocked, or, in BitTorrent lingo, *choked* by the remote peer. The decision to unchoke, or not, a client is made based on several criteria embodied in the *choking algorithm* [10]:

- *Reciprocation*: peers unchoke the clients that provide the best upload rates.
- *TCP performance*: TCP behaves better when the number of simultaneous uploads is capped.
- *fibrillation avoidance*: frequent (un-)choking causes data transfer interrupts that deteriorate protocol performance.
- *optimistic unchoking*: new peers are occasionally unchoked so as to discover potentially better connections. This is also how new peers acquire their first pieces.

When a client is unchoked by a peer, it starts sending REQUEST messages, each soliciting a specific *block* of the selected piece. The peer sends back the requested data using PIECE messages. Upon completing a piece download, the client informs via HAVE messages all the peers that it maintains connections with. These peers update the bitfield for that client and may then express their interest for that piece.

2) Connections: A client periodically learns about other peers by utilizing the Tracker protocol and parsing the peer list returned. The client joins the swarm by establishing connections with some of those peers. However, as noted in [10], each connection incurs an increase in signaling traffic, especially for bitfield maintenance via the exchange of HAVE messages. Thus, our implementation provides configurable lower and upper bounds for the number of established connections, using the `minNumConnections` and `maxNumConnections` configuration parameters (see Table I).

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 in 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 and HAVE message exchanges, the former 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 I), 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 from 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 latter 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.

4) Queueing: REQUEST messages refer to specific blocks of a piece. This facilitates fine-grained data exchange by enabling queuing of data requests. As common piece sizes vary from 256 KB to 1 MB [10] or even larger, per piece requests would result in a many redundant 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 I) request messages for blocks. Once a PIECE message has been received, the client may send the next REQUEST message. In case a piece has been requested in its entirety and 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 II-B.3) and starts sending REQUEST messages for its blocks.

5) Choking algorithm: For the choking algorithm we followed the guidelines presented in Section II-B, along with the ability to tune the choking algorithm as desired. The user may 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 I). To enable content providers to offer advanced seeding capabilities (see Section IV-B), the above parameters can be separately configured for such nodes via the `seederDownloaders` and `seederOptUnchokedPeers` parameters. The parameter `newlyConnectedOptUnchokedProb` is the probability that the most recently connected peer will be preferred over previously connected ones in an optimistic unchoke decision.

6) Super Seeding: The *super seed* feature is especially useful for content distribution as it helps the initial seeder to avoid excessive bandwidth consumption while fostering data exchange between participating peers. A super seeder does not inform its peers that is has all pieces available, masquerading as an ordinary client. Initially, it pretends to possess no pieces and only later informs them 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 a peer, the seeder will not inform it of other pieces until it sees this piece marked as available in the bitfield of other peers, implying that the first peer has in turn uploaded that piece.

Our module implements this feature in all clients, but only enables it at the initial seeder via the `super seed mode` configuration parameter (see Table I), since super seeding is not recommended for ordinary peers [10]. Instead, ordinary peers act as regular seeders after downloading all pieces. The

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 I
PEER-WIRE PROTOCOL PARAMETERS.

duration of this regular seeding phase can be set via the `timeToSeed` configuration parameter (see Table I).

7) Endgame mode: 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 [10], 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 I).

III. IMPLEMENTATION

The architectural design approach we followed resembles the philosophy of the INET framework upon which we built our modules. On the whole, we opted for the following: *simplicity* for eliminating simulation complexity, *modularity* for supporting abstractions and facilitating future add-ons, and *extensibility* for encouraging the model's evolution by community contributions. Most of the implementation specific characteristics (*i.e.*, the features left open in the specification), were encapsulated and abstracted in order to produce a simulation package that is both concrete and extensible. Thus, simulations can run without touching the source code, simply by editing the corresponding configuration files (*e.g.*, `.ini` and `.ned` files), while at the same time maintaining the ability to change many aspects of the model behavior. Furthermore, in order to promote the expansion of the provided modules, we have modularized the source code of important peer wire protocol features such as the choking algorithm and the selection of the peer(s) for optimistic unchoking (see Section II-B.5), the piece downloading strategy (see Section II-B.3), the entry condition for the endgame mode (see Section II-B.7), and so on. Hence, different algorithms and behaviors may be easily implemented simply by redefining the respective methods.

Our model consists of three modules, namely TRACKER, TRACKER CLIENT, and PEER-WIRE as shown in Figure 1. As their names suggest, the first module provides the functionality of the tracker as described in Section II-A, 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 II-B. 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 the INET 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 IV.

Since all modules depend on the INET framework, they rely on 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: the TCPSRVHOSTAPP and the TCPGENERICSRVAPP models. The former dynamically creates and launches a new *thread*[4] 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 P2P protocol such as BitTorrent, while it also saved us from the burden of maintaining centralized multi-peer state.

A. 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. Communication between these modules is carried out through the BTTRACKERMSGANNOUNCE and BTTRACKERMSGRESPONSE messages, both derived from the CMESSAGE class. The first class implements the client announce messages and includes all necessary fields, with their corresponding semantics, while the second class encodes the tracker's responses.

The tracker functionality is implemented in the BTTRACKERBASE module as a multi-threaded network application. Upon each successful connection to the tracker, a new thread is generated to drive the session between the tracker and the peer. The BTTRACKERCLIENTHANDLERBASE module, stemmed from the INET TCPSERVERTHREADBASE, is used

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

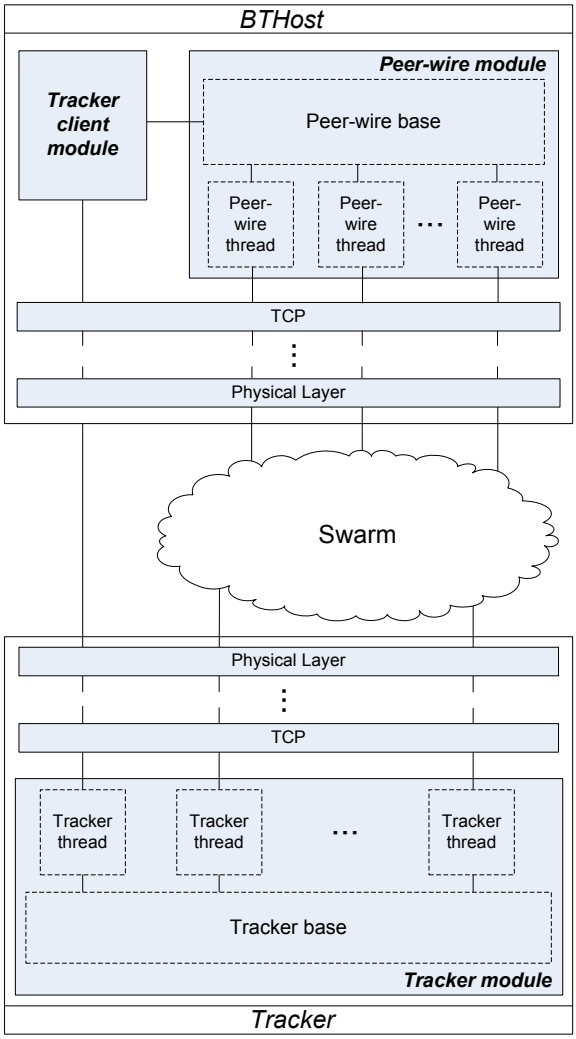

Fig. 1. BitTorrent module architecture.

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 including the functionality required to retrieve tracker responses and feed the BitTorrent application with received information (*i.e.,* the contact information of other peers). We implemented all the tracker protocol parameters described in [10]. The key configuration parameters of both modules and their default values are shown in Table II (server) and Table III (client).

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

TABLE II
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 III
TRACKER CLIENT PARAMETERS.

B. 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 that encompasses the following functionalities:

1) Tracker protocol information retrieval, 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 strategy. 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
4) Informing peers about the availability of new pieces, in both the normal and super seeding modes.
5) Applying the choking algorithm.
6) Coordinating the endgame mode.

The BTPEERWIRECLIENTHANDLERBASE class, derived from the INET TCPSERVERTHREADBASE, handles the communication with a single peer. This means that all peer-wire protocol message exchanges between peers (see Section II-B.1) are driven by instances of this class. The coordination of individual threads (*e.g.,* assuring that a certain block is not requested from more than one peer, except while in the endgame mode), is performed by BTPEERWIREBASE.

The TCPSRVHOSTAPP model 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 incorporating also the functionality of the client side. Because of that, our BTPEERWIREBASE class allows each new *active* connection to be handled by a separate BTPEERWIRECLIENTHANDLERBASE thread 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 the connection is handled by two identical thread objects; code structure is therefore simplified, since the peer-wire protocol functionality is incorporated in a single event-based class.

Apart from the classes presented above, our implementation employs two additional utility classes: BITFIELD and BTU-

TILS. The former represents a peer's bitmap, including block information, and provides all necessary handling functions, such as initialization, updates and queries. The latter is used for handling all the protocol state information described above.

C. 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.,* peers who cannot find a peer to provide their missing blocks/pieces). A peer is considered to have failed if it receives empty tracker responses for `maxNumEmptyTrackerResponses` times before downloading completes (see Table I). 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 quantifies the degree in which the downloading process is spread across the swarm, while the latter reveals whether users tend to download directly from the initial seeder.

For all the above metrics, both individual measurements (*i.e.,* vector statistics) and aggregated values (*i.e.,* scalar statistics) are recorded, accommodating both 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, *etc*.

IV. CREATING SIMULATION SCENARIOS

Our 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 (*e.g.,* TRACKER, TRACKER CLIENT and PEER-WIRE) must be loaded as submodules of the compound modules representing the peers and the tracker. While this procedure is sufficient for testing the modules, it is cumbersome to use when constructing realistic scenarios such as:

- Large-scale network topologies that require careful handling of node module placement and interconnection.
- Random introduction of clients into the simulation, both topologically and chronologically, as the network description files cannot capture such dynamics.

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 [8], which provides several of the features required to establish realistic and dynamic simulation scenarios. It must be stressed however that our BitTorrent implementation is not OverSim dependent: it can optionally employ several of the features provided by OverSim. In the following sections we present the OverSim features that we exploited along with our enhancements.

A. Topologies

One of the big concerns in creating realistic simulation platforms for BitTorrent is the underlying network topologies. OverSim provides various underlying network structures, both simple (*e.g.,* SIMPLEUNDERLAY) and composite (*e.g.,* IPV4UNDERLAY). However, both models present significant limitations in representing a realistic network substrate.

The SIMPLEUNDERLAY model was designed to provide a simple and scalable network substrate specially tailored for simulations focusing on the functionality of higher layer protocols, such as the set of overlay routing schemes provided by OverSim. In this model, packets are directly exchanged between end hosts completely neglecting the functionality of the underlying protocol stack. Packet delivery is performed by simply considering the characteristics of the communicating end hosts' access links and samples of end to end propagation delays derived from CAIDA's Skitter project [13].

This lack of protocol functionality and step-by-step routing in SIMPLEUNDERLAY turned our attention to the more realistic IPV4UNDERLAY model. In [14] we addressed two important limitations of this model. First, the model only provides a distinction between backbone and access routers neglecting the 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 extending the BRITE topology generator [15] export tool [16] that enables full support of the very popular *Georgia Tech Internet Topology Model* (GT-ITM) [9] 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 when the desired data are already present locally [2].

This has triggered several research efforts [17], [18] that 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 [19] 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 [18].

Second, OverSim does not make any distinction between the uplink and downlink characteristics of access links (*i.e.,* bandwidth). However, this distinction is important for providing realistic networking environments, since typical current access technologies, such as ADSL, do present this asymmetry. This issue becomes more important due to the fact that bandwidth heterogeneity results in a systematic unfairness of the peer-wire protocol, as the download rates achieved are based on the tit-for-tat mechanism. Hence, we further enhanced OverSim's

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

TABLE IV
BANDWIDTH DISTRIBUTION OF ACCESS LINKS.

Parameter	Value
Transit domains	7
Avg. routers per transit domain	4
Stub domains per transit router	7
Avg. routers per stub domain	7
Stub routers	1372
Transit routers	28
Total routers	1400

TABLE V
NETWORK TOPOLOGY PARAMETERS.

IPV4UNDERLAY model to support a range of *channel* [5] 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. In the measurements presented below, we have set these values as depicted in Table IV. For example, 40% of the participating peers can download data at a maximum rate of 8 Mbps and upload data with a maximum rate of 1 Mbps.

B. Host Deployment

Having established a realistic network topology, the next step is to deploy the corresponding entities on it. As far as the tracker is concerned, to avoid coupling the network topology description file with the application, we extended the OverSim IPV4UNDERLAYCONFIGURATOR module to dynamically introduce the BitTorrent tracker in the network.

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 example, the content might be a new Linux distribution (*i.e.,*, an ISO image file), hosted by a dedicated server with a high capacity access link, while ordinary peers participate in the swarm through ADSL links. Protocol parameters may also be altered to achieve a 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: 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 chronological sense (*i.e.,* they need to be placed at random nodes in random points of time). 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, which describe 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 modeling the arrival process. Using the OverSim churn generator mechanism, we implemented the BITTORRENTCHURN model that reproduces the arrival process of BitTorrent clients presented in [5]. 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. Peers leave the swarm after their download completes or fails; in the former case, they may optionally also act as seeds for a period of time.

V. RESULTS

In the following we present some preliminary measurements produced by our simulation modules. First, we investigate the resource demands of the presented implementation, including the underlying network models, and then we demonstrate the flexibility provided by the created simulation environment in investigating BitTorrent protocol parameters. The performance results reported in this paper were obtained with OMNeT++ v.3.3, INET v.20061020, and OverSim v.20080416 (patched with our modifications), on a Intel Dual Core E5200 2.5 GHz processor with 4 GB of RAM running Ubuntu Linux 8.04.

Based on our findings on the memory footprint of GT-ITM based topologies of several sizes [14], we created an IPV4UNDERLAY topology that strikes a balance between the memory consumption and complexity of the routing substrate, consisting of 1372 stub and 28 transit routers. The detailed characteristics of the employed topology are presented in Table V. We used the default link establishment probabilities, that is, a link between two transit routers was established with a probability equal to 0.6 and a link between two stub routers was established with a probability equal to 0.42. Each peer is attached via an access link to a randomly selected stub router upon arrival. In addition, we used a SIMPLEUNDERLAY topology consisting only of peers attached to access links, with the propagation delays between the peers being drawn from CAIDA's Skitter project [13].

[5]We adopt the OMNeT++ definition of a channel, which includes the data rate, error rate, and propagation delay characteristics of a link.

Parameter	Value
file size (MB)	200
swarm size	30, 60, 90, 120
piece size (KB)	256, 512, 1024, 2048, 4096
end game mode	false
seederDownloaders	25
timeToSeed (sec)	360
downloadRateSamplingDuration (sec)	9

TABLE VI
PEER-WIRE PROTOCOL PARAMETER VALUES.

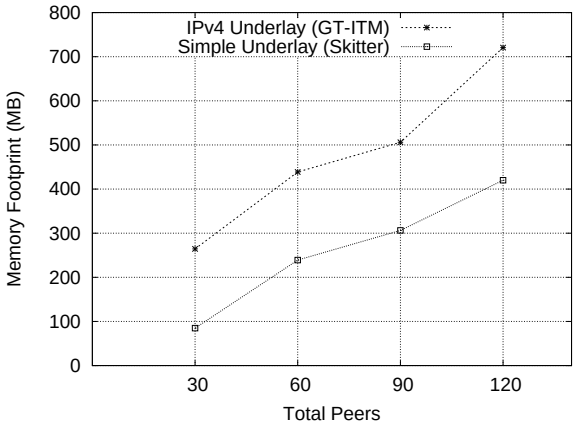

Fig. 2. Simulation memory requirements.

In both cases, peers enter the swarm according to the distribution described in Section IV-B with an initial arrival rate of 0.0166 peers/sec. We varied the access link bandwidths across the network in order to achieve the heterogeneity present in real environments. The uplink / downlink bandwidth values of the various access link types as well as their distribution are presented in Table IV. The tracker and the initial seeder (content provider) have 10 Gbps symmetric access links however. We chose to set such high values for the access link of the tracker in order to avoid creating a bottleneck, so as to focus on the performance of the peer-wire base protocol. The same access link attributes were used for the initial content provider in order to simulate a dedicated high bandwidth seeding node, as discussed in Section IV-B.

Table VI shows the parameters set to non-default values or varied in these sample experiments. Specifically, the size of the file to be distributed via BitTorrent is 200 MB, the swarm size is 30, 60, 90 or 120 peers, and the piece size varies (256, 512, 1024, 2048 or 4096 KB). The initial seeder is allowed to unchoke a large number of peers (*i.e.,* `seederDownloaders` was set to 25) and continues seeding until all peers in the swarm have downloaded the content. The rest of the peers only keep seeding the file for a small period after they have completed downloading the file (*i.e.,* `timeToSeed` was set to 360 seconds). In order to further stress the downloading procedure we have deactivated the endgame mode of the peer-wire protocol. Finally, the `downloadRateSamplingDuration` parameter was set to 9 seconds, reflecting the time interval during which a peer averages the collected samples of download rate.

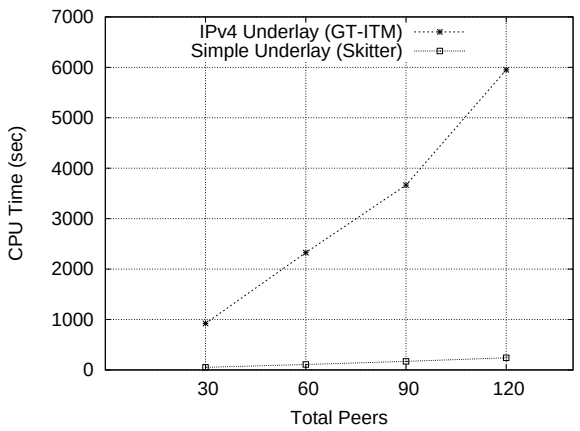

Fig. 3. Simulation processing time.

The simulator memory requirements are shown in Figure 2. The x-axis presents the total number of peers comprising the swarm, while the curve indicates the maximum size of the memory footprint recorded during the execution of the corresponding scenario. The difference between the IPV4UNDERLAY and SIMPLEUNDERLAY models clearly shows that a significant part of the former's memory footprint is due to the underlying routing substrate. This is an indication of the well known tradeoff between simulation scalability and realism. However, considering the fact that the IPV4UNDERLAY results refer to a full fledged simulation environment, the total memory footprint of 720 MB for the simulation of a swarm of 120 peers can be considered as an acceptable cost when realism is important.

The tradeoffs between the two models are more evident in Figure 3 which shows the processing (CPU) time for the same scenarios. In the IPV4UNDERLAY case the processing time reaches 98 minutes, due to the operation of the full protocol stack for 120 end-hosts and 1400 routers. On the other hand, in the SIMPLEUNDERLAY case, the processing time for the same number of end-hosts is only 4 minutes, indicating that our simulator can easily be used for studies that focus on the *application logic* of BitTorrent. However, in this case the impact of the underlying network operation on protocol performance is completely neglected (see Section IV-A).

Figure 4 illustrates the effect of the piece size on the average download time experienced by a swarm of 60 peers for both network models. The deviation of the individual values from the mean was quite high in both cases due to the heterogeneity of the access links, although this is not visible in the SIMPLEUNDERLAY due to the scale of the figure. While one could argue that similar results could be produced by increasing the propagation delays used in the SIMPLEUNDERLAY model so as to account for the queueing delays of the IPV4UNDERLAY model, a closer inspection of the results reveals that the two curves behave differently.

In the SIMPLEUNDERLAY case, as the piece size increases so does the average download time. This is attributed to the fact that as the piece size increases, the length of the bitfield decreases, thus providing fewer choices to the piece selection strategy (see Section II-B.3). Recall that in order to facilitate

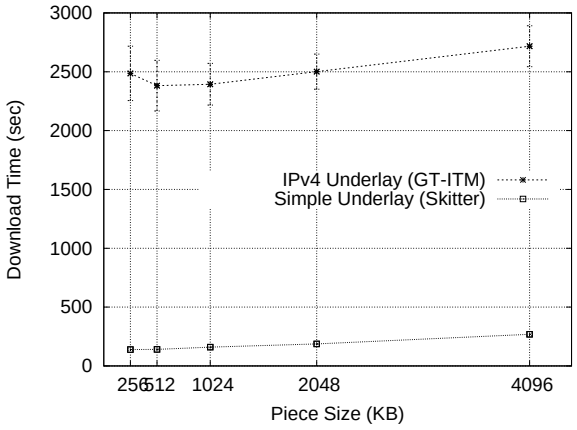

Fig. 4. Effect of piece size on download time.

the efficient implementation of the queueing policy (SectionII-B.4), a peer refrains from requesting a block belonging to a piece that it has already started downloading from another peer. Effectively, parallel downloads are limited by the number of missing pieces, not missing blocks. When the number of pieces comprising the file decreases due to the larger piece size, it becomes harder for a client to find enough pieces to request from different peers.

In the IPV4UNDERLAY case however, the download times also increase at smaller piece sizes. This is attributed to the fact that after each piece is downloaded, the client informs its connected peers via HAVE messages. By halving the size of pieces, their number doubles, and so do the HAVE messages, thus increasing the queueing delay at each client. In the simplistic SIMPLEUNDERLAY model queueing is ignored, effectively acting as if all these messages are transmitted in parallel with data blocks. As a result, while the SIMPLEUNDERLAY indicates that download time is minimized at the smallest piece size, the more realistic IPV4UNDERLAY reveals that download time is minimized at an intermediate piece size.

VI. 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 important protocol aspects, works such as [20], [21], [22], 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. Our implementation avoids this situation by providing almost all features specified in [10] 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 [23], implemented for ns-2 [24]. 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. Furthermore, our implementation allows fine grained tuning of the protocols by providing several configuration parameters not available in [23], such as `optUnchoking Interval`.

Finally, we note that our implementation is not the first attempt to incorporate BitTorrent in the OMNeT++ simulation platform. A swarming-based module is presented in [22], but it only provides a bare-bones subset of the BitTorrent protocol features, since it lacks the tit-for-tat mechanism. This module was also developed using trivial network topologies with dedicated non-TCP connections among all pairs of peers. Likewise, a BitTorrent implementation for OMNeT++ that does not utilize the INET framework is presented at [25]. The authors implemented most of the BitTorrent features (*e.g.,* rarest first piece selection strategy, chocking), but their model suffers from the same deficiencies as [20], [21], [22] do (*i.e.,* the underlying protocol stack is missing and the corresponding peer modules are linked to each other using abstract connections). More importantly, critical parameters such as link delay/bandwidth and swarm interarrival times are modeled using the probabilistic distributions provided by OMNeT++. In contrast, in our implementation we can use realistic and detailed network substrates, deploying the corresponding entities using a churn model derived from real-life traces. According to the authors [26], the next major version of their implementation will incorporate a realistic underlay. However, this version is not yet available and it is not clear whether it will be assembled on top of the INET framework, or on a custom-made equivalent.

VII. FUTURE WORK

Our plans for future extensions to the simulator primarily focus on enriching the set of extracted statistics. To this end, we aim to track 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. We also plan to take advantage of the enhanced topology model employed (see section IV-A) by providing support for the extraction of topology-aware metrics (*i.e.,* metrics for specific areas of the network such as selected 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, we plan to enable the separate extraction of statistics for the operation of the initial seeders.

Regarding the creation of simulation scenarios (see Section IV), 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 [8], we currently investigate the design space for the implementation of the DHT-based trackerless extension.

VIII. Conclusions

In this paper we have presented an implementation of the BitTorrent set of protocols for the OMNeT++ simulation environment [7]. 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 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. The results presented in this paper indicate that we can perform detailed medium-scale simulations of realistic Internet-like swarming scenarios in commodity hardware and, equally important, that is makes sense to do so.

References

[1] PSIRP, "Publish-Subscribe Internet Routing Paradigm," Jun 2009.

[2] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should Internet service providers fear peer-assisted content distribution?," in *Proc. of the Internet Measurement Conference (IMC)*, (Berkeley, CA, USA), pp. 63–76, Oct 2005.

[3] B. Cohen, "Incentives build robustness in BitTorrent," in *Proc. of the Workshop on the Economics of Peer-to-Peer Systems*, (Berkeley, CA, USA), pp. 116–121, Jun 2003.

[4] BitTorrent.org, "BitTorrent Protocol Specification," Jun 2009.

[5] 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*, vol. 25, no. 1, pp. 155–169, 2007.

[6] 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)*, (Ithaca, NY, USA), pp. 205–216, Feb 2005.

[7] K. Katsaros, V. Kemerlis, C. Stais, and G. Xylomenos, "BitTorrent module for OmNEt++," Jun 2009.

[8] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proc. of the IEEE Global Internet Symposium*, (Anchorage, AK, USA), pp. 79–84, Jan 2007.

[9] E. Zegura, K. Calvert, , and S. Bhattacharjee, "How to model an internetwork," in *Proc. of the IEEE INFOCOM*, vol. 2, (San Francisco, CA, USA), pp. 594–602, Mar 1996.

[10] TheoryOrg, "BitTorrent Protocol Specification v1.0," Jun 2009.

[11] BitTorrent.org, "DHT protocol," Jun 2009.

[12] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)." RFC 1738, Dec 1994.

[13] CAIDA, "CAIDA skitter Topology Traces," Jun 2009.

[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)*, (Aveiro, Portugal), Jul 2009.

[15] 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)*, (Cincinnati, OH, USA), pp. 346–353, Aug 2001.

[16] A. Varga, "OMNeT++ export for BRITE 2.1," Jun 2009.

[17] H. Xie, R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, "P4P: provider portal for applications," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 351–362, 2008.

[18] 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)*, (Lisbon, Portugal), pp. 66–66, Jul 2006.

[19] IANA, "Autonomous system (AS) numbers," Jun 2009.

[20] A. Bharambe, C. Herley, and V. Padmanabhan, "Analyzing and improving BitTorrent performance," Tech. Rep. MSR-TR-2005-03, Microsoft Research, Redmond, WA, USA, Feb 2005.

[21] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a BitTorrent network's performance mechanisms," in *Proc. of the IEEE INFOCOM*, (Barcelona, Spain), pp. 1–12, Apr 2006.

[22] P. Korathota, "Investigation of swarming content delivery systems," Master's thesis, Sydney University of Technology, November 2003.

[23] 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 (UPGRADE)*, (Monterey, CA, USA), pp. 9–16, Jun 2007.

[24] UCB/LBNL/VINT, "The Network Simulator - ns - 2," Jun 2009.

[25] K. De Vogeleer, D. Erman, and A. Popescu, "Simulating bittorrent," in *Proc. of the International Workshop on the Evaluation of Quality of Service through Simulation in the Future Internet (QoSim)*, (Marseille, France), Mar 2008.

[26] "BT-SIM a BitTorrent Simulator," Jun 2009.