Available online at www.sciencedirect.com

Journal of Applied Research and Technology

Journal of Applied Research and Technology 15 (2017) 513–523

www.jart.ccadet.unam.mx

Original

BiT-SiEm: A packet-level simulation and emulation platform for BitTorrent

Alberto Aguilar-Gonzalez *, Camilo Lozoya, Carlos Ventura-Molina, Rodolfo Castelló, Armando Román-Flores

Tecnologico de Monterrey, Campus Chihuahua, Mexico

Received 20 October 2015; accepted 1 May 2017
Available online 11 December 2017

Abstract

Simulation and emulation have been used by researchers to develop new network protocols and to test, modify and optimize existing ones; these experiments must imitate the heterogeneous nature of the Internet in a controllable environment suitable to answer what-if questions. BitTorrent is not the exception. In this paper, we present a simulation platform that allows the interaction of simulated and emulated peers; the BitTorrent protocol is implemented in the simulator with a high degree of detail in which any genuine BitTorrent implementation would seamlessly fit into the platform. To evaluate our platform, several experiments were completed with thousands of peers; then, we compared the results of these tests with two mathematical models to corroborate its correctness.

Keywords: BitTorrent simulation; BitTorrent emulation; Packet-level simulation

1. Introduction

Peer-to-peer (P2P) systems have emerged as a simple yet effective solution for two main weaknesses of traditional client/server content sharing applications: scalability and reliability. For starters, typical systems tend to suffer as the number of hosts increases, which degrades quality of service (QoS) and in some cases generate denial of service (DoS). In client/server schemes providing some level of scalability implies high costs in terms of infrastructure, management and maintenance (Aguilar, Ege, Makki, & Bustos, 2007). Also, fault tolerance in these scenarios is not easy to implement since servers become a single point of failure, and consequently, costs to have replicated services are higher as the number of hosts increases. Most P2P systems have the capability to solve these problems.

In P2P systems scalability is enforced by balancing requirements of resources between hosts in the network, indeed, P2P protocols tend to function better as more peers join the system (Cohen, 2003; Lin, Fan, Lui, & Chiu, 2007). There systems are designed to share load in terms of bandwidth and computing power in a way that when a host is added not only its demand for a service is aggregated, but also, it provides processing power and network resources that can be used directly or indirectly by any of the hosts; this is why we also refer to them as *peers* since there is no clear distinction if it always acts as a client or has a server role. Load balancing also improves reliability since P2P systems do not depend on a single host; members of these networks can be treated as servers or clients according to protocol-specific needs.

Although there are several P2P content distribution systems (Eger, Hoßfeld, Binzenhöfer, & Kunzmann, 2007) available on the Internet, BitTorrent has become the most used file sharing system and it accounts for a representative portion of Internet traffic (Izal, Urvoy-Keller, Biersack, & Felber, 2004). Currently, there are several studies on this P2P protocol, some of them focus on analytical models (Qiu & Srikant, 2004; Lin et al., 2007), others on the study of real traces (Izal et al., 2004; Pouwelse, Garbacki, Epema, & Sips, 2005) and finally, some of them focus on research based on analysis via simulation of the protocol (Bharambe, Herley, & Padmanabhan, 2005; Bindal et al., 2006; Eger et al., 2007; Katsaros, Kemerlis, Stais, & Xylomenos, 2009; Microsoft, 2007). Of course, this list of studies is far from being exhaustive but gives us an idea of the varied interests of the

* Corresponding author.
E-mail address: alberto.aguilar@itesm.mx (A. Aguilar-Gonzalez).
Peer Review under the responsibility of Universidad Nacional Autónoma de México.

https://doi.org/10.1016/j.jart.2017.05.009

research community in this matter. In this paper, the research focus is on the simulation and emulation of BitTorrent.

Simulation has been used by researchers to develop new protocols and to test, modify and optimize existing ones. To accomplish these purposes, the scientific community has developed a wide range of tools in order to facilitate this creative process. Since most of these applications and protocols were designed to be used on the Internet, simulators must mimic the heterogeneous nature of this network in a controllable environment suitable to answer *what-if* questions. Similarly, research tools must support configuration of large-scale experiments to imitate how a system would behave during the interaction of a vast number of hosts.

Furthermore, if we need realism during the development or improvement of network protocols it is necessary to have tools that allow interaction of *real* applications with the simulation platform. With these capabilities, researchers are able to test and debug a system not only in terms of some specification, but also considering how a running application would perform.

Given this, we are confident that experiments with any protocol must run in controllable environments to be evaluated with different parameters, but also have a level of immersion that only *real* applications can provide by means of emulation.

Of course, BitTorrent is not an exception, there are several BitTorrent simulators available (Naicken, Basu, Livingston, & Rodhetbhai, 2006), however, our survey indicates that none of them are able to exchange information with genuine implementations of the protocol and most of them have been tested on small to medium network scenarios.

In this paper we present a simulator with emulation capabilities built on top of PRIME (Liu, 2008), a parallel real-time simulator, which has these desired properties.

2. Background

In this section we discuss three topics of interest for the study presented throughout this paper. First, a description of the BitTorrent P2P protocol is given: an overview and basic operation, the protocols that are part of it are presented and, also an explanation of its main algorithms and mechanisms is provided to understand some of the complexity of simulating and emulating the protocol. Second, we provide a survey of some of the available BitTorrent research studies. And finally, we explain PRIME, a platform for real-time simulation and emulation of large networks, which is used as our primary tool to achieve a large-scale simulator and emulator for BitTorrent.

2.1. BitTorrent

BitTorrent (Cohen, 2003) is a P2P system designed to distribute content in an efficient and fair manner. Efficiency encompasses several factors, for instance, workload is distributed between peers and, according to several studies, as more hosts become part of the system efficiency tends to improve (Bharambe et al., 2005; Lin et al., 2007). Also, the algorithms that are part of it are designed in such manner that gives BitTorrent proficiency over other P2P systems; we will discuss more on these later. Fairness is accomplished using a tit-for-tat approach (Cohen, 2003), that is, a peer shares data with other hosts that are also willing to contribute at similar rates.

To start, it is necessary to clarify some concepts related to BitTorrent. Hosts are generally referred as *peers* and mentioned indistinctly. A peer may be either downloading data from other hosts or uploading information to other peers. We say that a peer becomes a *seed* once it has a complete copy of the file being made available. In BitTorrent, a file is divided into fragments called *pieces*, and then reassembled as these are received; the purpose of breaking a file is to distribute small blocks without the need of a sequential send/receive process; this feature accelerates the downloading process by obtaining pieces that are already available from known connected peers.

A torrent is a file that contains *meta data* about the files being shared including name, sizes and checksums of all the pieces in the torrent; these checksums are done using the One Way Hash Function SHA1 to avoid piece tampering; in addition, this torrent file contains the URL of the *tracker*, which is an special host in charge of keeping membership of the *swarm*. Finally, a *swarm* is the set of peers interested in a specific torrent.

2.1.1. Overall operation

In Figure 1, a flow diagram with the overall operation of this protocol is shown. A peer that needs to join a swarm must have the corresponding torrent file in order to contact the tracker. Then, this latter receives a request from a peer and sends back a random list of IP and ports of other hosts that are part of the swarm. With this list of hosts, a peer establishes TCP connections with a sufficient number of them; these connected peers are known as *neighbors*. In parallel, two other processes take place: choking (Cohen, 2003) and connections monitoring; this link

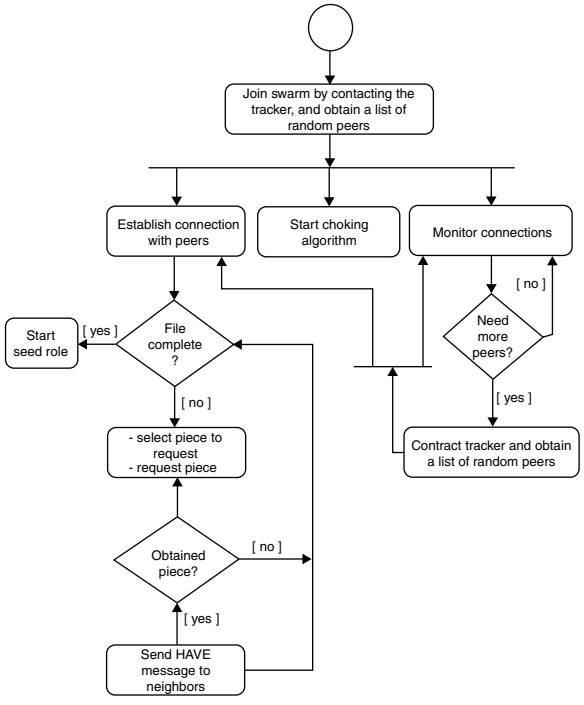

Fig. 1. BitTorrent overall operation.

supervision is necessary to have a proper number of neighbors. Next, a peer evaluates constantly if it has a complete copy of the file, that is, all pieces have been received from other peers. If a peer is still missing pieces, an algorithm selects one and requests it to one of its neighbors. Once a piece is received, a HAVE message is sent to all neighbors to inform which piece is now available from that peer. After a peer obtains all pieces, its role is changed to *seed*.

2.1.2. Tracker

A tracker maintains a list of hosts that are interested in some torrent. To establish communication with a tracker, peers send an HTTP request with several parameters such as: port used for incoming connections, a peer id, and size of the requested list. Since a tracker congregates the information from all peers in a swarm, it is able to respond with a random list of peers that will be used to create the overlay network.

2.1.3. Peer-wire protocol

Once peers are aware of the presence of other hosts in a swarm, and after TCP connections have been established, hosts exchange control and data messages (Cohen, 2003). These messages are:

- Choke. Used to inform a peer that it has been choked which means temporally blocked.
- Unchoke. Sent when a peer is unchoked.
- Interested. A peer sends this message when interested in keeping the connection.
- Not interested. No longer interested in maintaining a connection.
- Have. Message sent to neighbors when a piece has been received.
- Request. Request a specific piece.
- Piece. Send a piece to another peer, this message includes piece data, known as payload.
- Bitfield. This message is sent during the handshake between two peers, and is simply a bit array to inform which pieces are currently in possession.

We studied in detail the purpose and format of each of these messages in order to fully implement logic and behavior of BitTorrent, as we will explain in Section 3.2.

2.2. Studies on BitTorrent

BitTorrent has been the target of several research papers; in summary we can divide studies in three areas. Some work uses analytical models (Lin et al., 2007) to represent its behavior, while other work has focused on the evaluation of several aspects of the protocol using real traces from a tracker (Izal et al., 2004; Pouwelse et al., 2005). Finally, as a third research focus, the scientific community has been studying this system using simulators as a primary tool to evaluate its behavior under different circumstances, some examples are (Bharambe et al., 2005; Bindal et al., 2006; Chen, Chu, & Liu, 2010; Eger et al., 2007; Katsaros et al., 2009; Microsoft, 2007). In this last branch of research, we detected three areas of opportunity. First of all, most of these simulators are designed to run hundreds of peers, but not large-scale experiments. As a second observation, most of these tools use flow-level simulation (Microsoft, 2007) where TCP behavior is not considered, but according to some studies (Eger et al., 2007) data rates can vary up to 30% in P2P networks while using packet level instead of solely exercising flow level simulations. As a third comment, current work lacks interactions of *simulated* entities with *emulated* hosts, which might lead to improper implementation of a system since validation via real applications is not available. Nevertheless, we imply that these simulators are not correct, but we are confident that inclusion of real applications gives more precision to experiments and adds a validation layer not available on simulation tools.

2.3. PRIME

PRIME (Liu, 2008), which stands for Parallel Real-time Immersive network Modeling Environment, is a parallel real-time network simulator written in C++ capable of running large-scale network models (Erazo, Li, & Liu, 2009; Liu, Li, & He, 2009). It is based on scalable simulation framework (SSF) (Liu, 2008), which has been proven capable of managing large network models.

It provides all the necessary elements to build on top of it network and application protocols with the following advantages:

(1) Flexibility and repeatability of experiments providing a controllable environment.
(2) Scalability, simulation of thousands of hosts using advanced parallel techniques.
(3) Emulation-enabled platform, which gives realism and adds validation to simulated software.
(4) Ability to test with varied realistic network models.

The distinctive attribute of PRIME is its ability for real-time simulation, which combines the advantages of both simulation and emulation: it can run different simulation models and interact with the physical world simultaneously; simulation and emulation are seamlessly integrated. Simulation provides topology, traffic and TCP behavior while emulation provides real application behavior for the BitTorrent protocol. Details of the complexity to enable real-time simulation are out of the scope of this paper, but more information can be found in Liu (2008) and Liu et al. (2009).

3. BiT-SiEm

To construct BiT-SiEm, which stands for *BitTorrent and Tracker Simulator-Emulator*, we extended PRIME with a complete implementation of BitTorrent and, under specific circumstances, augmented the peer-wire protocol. In Figure 2 the high-level structure of BiT-SiEm is shown. Notice that it is possible to have simulated peers communicating through virtual

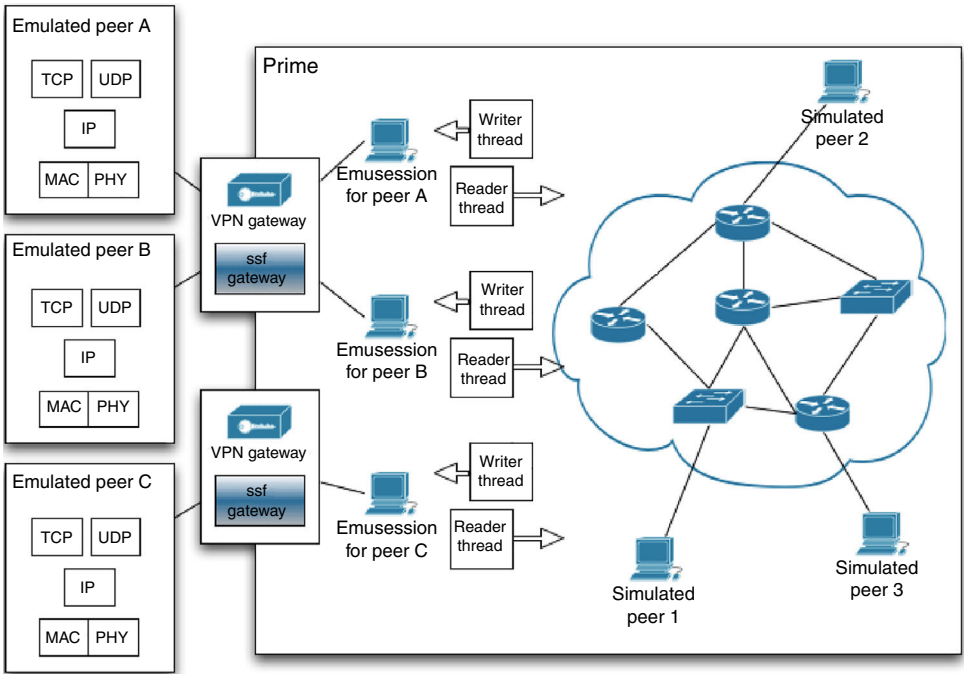

Fig. 2. BiT-SiEm structure.

network models (right side) and also real applications (left side) interacting with simulated entities.

3.1. Extending PRIME

Addition of new protocols in PRIME can be done extending a dummy protocol (Liu, 2008). Although PRIME is built on top of solid software architecture in order to be extensible, we created an additional abstraction layer to hide some level of complexity, to this end we implemented an intermediate module in order to abstract:

(1) TCP socket creation and handling.
(2) Threading.
(3) Concurrency mechanisms such as semaphores and the producer–consumer design pattern.

Our goal was to create an API which allows: (1) to focus on application logic, testing and debugging of BitTorrent, (2) to hide low-level PRIME classes and tools, and (3) to use concise and easy to learn interfaces. For instance, to connect simulated and emulated entities there is no need to use input and output streams, or to have threads waiting for data to be available. In this case, a host simply calls a method, which returns a connection identifier if a successful TCP session was opened. When data is available, a virtual callback function is executed with the connection identifier and a pointer to the data received (both as local parameters), thus, a host is able to identify from which connection data is ready to be processed.

Similarly, to avoid the use of threads we employ timers. As an example consider the choking algorithm (BitTorrent, 2014a) from BitTorrent which should run every 10 s. In our implementation we call a *set_timer* method, which returns a handle that can be stored in an integer variable. Then, when a timer expires a virtual callback function is called with the timer handle as local parameter.

These abstractions make our platform useful not only for our simulator, but also for other uses such as creation of application-level protocols. That is, use BiT-SiEm not only in academic and scientific research scenarios, but also in industry for software development and testing (Aguilar et al., 2007). Also, this platform can be used for stress and load testing of concurrent applications by putting heavy loads on the tested parts. All this with the implicit benefits of real-time simulation and emulation: controllability and immersion via real applications.

3.2. A complete BitTorrent implementation

In order to understand BitTorrent and develop our simulator, the following strategy was used. We consulted research papers and also the latest specifications, which can be found in (BitTorrent, 2014b). However, we ran into similar problems as the authors from (Katsaros et al., 2009): no formal protocol specification is available. Given this, and since our goal was to mimic protocol behavior in all aspects, we downloaded the client developed by Bram Cohen, creator of BitTorrent-available at (BitTorrent, 2014a) or via BitTorrent package in a Debian-like installation, which is implemented in Python; then, we did reverse engineering on about 7000 lines of code to fully perceive BitTorrent logic and its related algorithms. With this information, a new protocol in PRIME that replicates BitTorrent behavior was created.

3.3. Piece exchange

Since our simulator must be able to interact with real BitTorrent implementations, messages from the peer-wire protocol were imitated to serve their specific function. Furthermore, our simulator implementation exchanges messages *exactly* as Cohen's implementation (Cohen, 2003) does. Of special interest is the PIECE message, which includes a piece index and also the data. Thus, its size is considerable compared with other messages. Since we need to evaluate how the protocol behaves using packet-level simulation, piece data is filled with dummy content and then transmitted over the virtual network, or the real network when send to/from emulated peers.

Because a PIECE message includes the data itself, in a typical emulation this information flows from a virtual machine (VM) to an SSF gateway. Then, it is forwarded to the corresponding *EmuSession* (Liu, 2008) within a PRIME instance. Finally, a *Reader Thread* (Liu, 2008) creates a virtual packet that is injected into the simulation environment. This complete process is shown in Figure 2. Notice that emulation of a large number of emulated peers requires some type of virtualization (Barham et al., 2003). Therefore, there is one drawback to sending entire pieces: during an experiment, several VMs that are running one python BitTorrent instance might share a physical host. Since the network interface is also shared among emulated peers, this might lead to bandwidth constraints and bottlenecks.

One method to avoid these bottlenecks is applying some time dilation factor (Grau, Maier, Herrmann, & Rothermel, 2008) into the simulation to obtain more bandwidth from the network devices. However, in such case, simulation-running time grows in the same pace the dilation factor does, which could make some experiments hard to perform. For instance, consider the case where several virtual machines in different computers need to emulate nodes from a full-mesh enterprise network (Aguilar et al., 2007), in this case a small value for a dilation factor is not enough to keep up with 100 Mbps or bigger bandwidths. Therefore, when time dilation is not possible for practical reasons, we extend the Peer-Wire Protocol with two control messages:

(1) *S-PIECE-START <piece index> <piece size>* which is sent by the peer that is uploading. Both parameters for this message are self-explanatory: the first one is a number indicating an ID for the piece, while the latter is an integer containing the piece size in kilobytes.
(2) *S-PIECE-FINISH <piece index>* that is sent back by the receiver when a piece download simulation has finished.

An example of how these two messages fit in the platform is shown in Figure 3: a PIECE message is replaced with a S-PIECE-START and a S-PIECE-FINISH. Then, small messages are sent, which contrasts with a PIECE message that includes piece content. S-PIECE-FINISH is an ACK (acknowledge) that is sent back when the piece has been processed in the simulation environment, this message is used to inform the emulated peer that a simulated PIECE message has been transmitted successfully within simulation, that is, the stream for a piece is complete. Notice that the packet is simulated by BiT-SiEm in the same way a PIECE packet would be replicated.

3.4. Swarm membership

During the debugging process of BiT-SiEm, we started with experiments using simulated peers exclusively. In this case, we augmented the functionality of one peer to act as a tracker. However, in our proposed platform, simulated and emulated entities must interact seamlessly, which brings an additional challenge: keep track of membership for both peer types. To solve this issue we did the following: (1) augment the simulated peers not only with the Peer-wire protocol, but also with the required messages to contact a *real* tracker, and (2) configure an emulated tracker, that is, a tracker runs on a VM and receives requests from simulated *and* emulated hosts. In this way, real and simulated BitTorrent peers are able to contact the tracker and this latter is in a position to keep track of swarm membership.

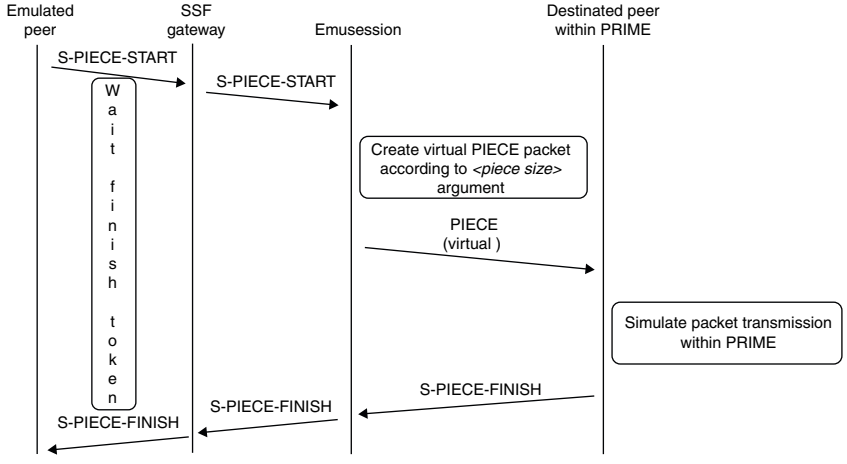

Fig. 3. Additional messages for the peer-wire protocol.

3.5. Interaction between emulated instances and simulated peers

Emulated peers communicate with simulated hosts using a kind of proxy that we call *Translator Agent* (TA), which acts as a mediator between emulated peers and simulated instances to translate messages and make communication feasible. A TA is needed in the following two cases.

For the first case, we have *e* emulated peers running on an equal number of virtual machines and *i* simulated hosts running on a physical machine. The TA should be able to handle communication for *e* TCP sessions at most. An example is show in Figure 4. We consider *e* a maximum value, but it can be the case of a TA will handle a smaller number of sessions when not all emulated peers are exchanging information with some simulated peer from the PRIME instance. When an emulated peer sends a PIECE message to any simulated host, the TA needs to remove the data part (piece content) and forward it to the intended destination peer. On the other hand, when an internal peer sends a PIECE message to an external host, the TA must create the missing data to send a complete PIECE message, except in the case explained in Section 3.3.

The second case that needs to be considered is when peers from two PRIME instances need to exchange information, which can be seen in Figure 5. In this case, only one TCP session needs to be established since information flows from TA to TA, that is, between two physical machines. The job of a TA is to convert C++ messages (method calls) into data that can be sent across the network and vice versa.

4. Experiments

To assess BiT-SiEm, a test bed used to run several experiments was installed and configured. Also, a network model was used to recreate a real environment for simulated and emulated peers. Finally, the results were compared with two mathematical models.

4.1. Test bed

The experiments comprised simulated and emulated peers. To this end, 30 personal computers running Debian Etch were configured. Emulated peers ran in OpenVZ containers acting as VMs. OpenVZ (Zhao, Zhang, & Cui, 2010) has demonstrated being capable of providing superior performance and scalability compared to other virtualization approaches, such as Xen (Barham et al., 2003), VMWare Workstation (VMWare, 2009), and User-Mode Linux (Dike, 2000). In our testing, 50 VMs ran, without problems or degraded performance, in a single physical host; we used this capacity as upper limit. Each VM ran a genuine BitTorrent application written in Python (BitTorrent, 2014a). To gather results and data files from the experiments, a network file system (NFS) was installed and configured in the physical machines; this is another advantage of OpenVZ over other virtualization techniques: the file system of the guest host can be accessed from the VMs. To handle creation, start, configuration and shutdown of OpenVZ containers several UNIX shell scripts were created.

Also, a time dilation factor (TDF) (Grau et al., 2008) of 25 was used to guarantee the required bandwidth for the experiments. Since OpenVZ does not support time dilation we used a simple yet effective approach to implement it in the BitTorrent source code. These additions can be seen in Figure 7. A *TimeDilation.py* file was created and imported in all modules which required the *time()* function, then, calls were substituted with the *TDFTime()* implementation.

To evaluate the behavior of the time dilation implementation for the BitTorrent application, several experiments were run sharing a 140 MB file in a simulated 100 Mbps switched environment. This information is summarized in Table 1.

The first column indicates the experiment number. For the second column, PM stands for physical machine, VM for virtual machine, while TDF stands for time dilation factor in the third column. The bandwidth can be *limited* or *real*. For instance, in experiments 4 thru 9 the bandwidth between the VMs was

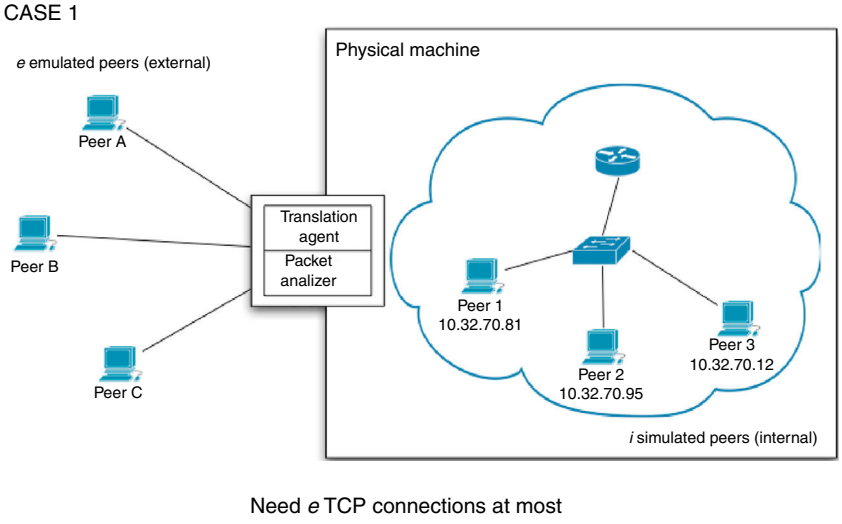

Fig. 4. Case 1 for simulation/emulation interaction.

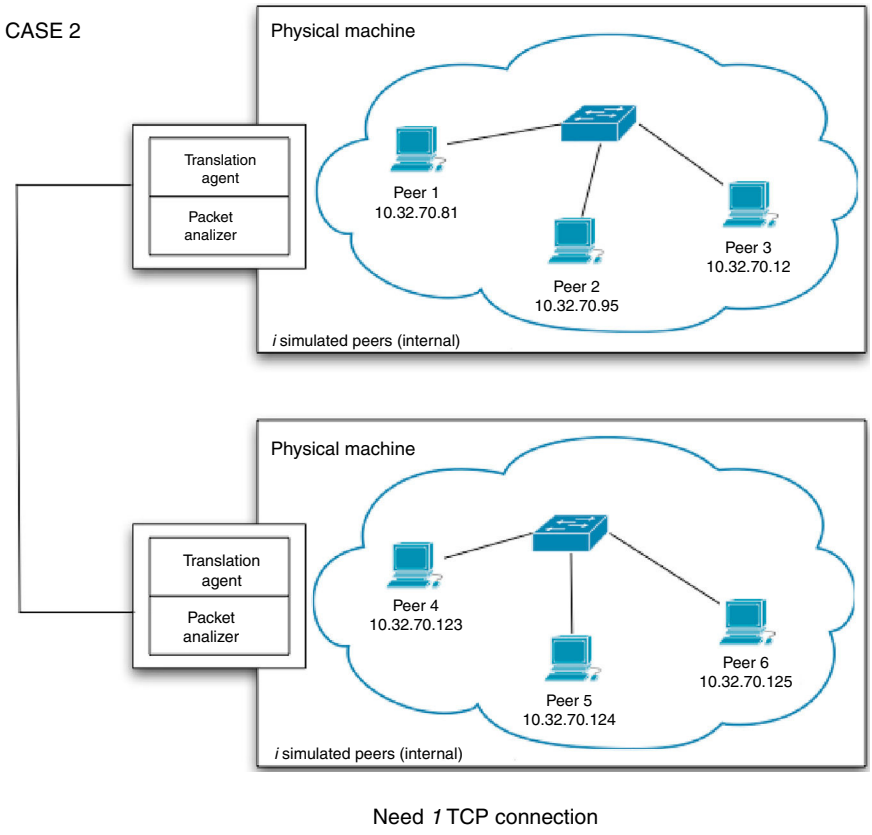

Fig. 5. Case 2 for simulation/emulation interaction.

Table 1
Experiments to test time dilation impact.

#	Peers	TDF	Bandwidth (Mbps)		Time (s)	
			Limited	Real	Clock	Real
1	5 PMs	–	–	100	25	25
2	5 VMs	–	NO	>100	12	12
3	5 VMs	–	100	100	24	24
4	50 VMs	50	2	100	1624	32.4
5	100 VMs	50	2	100	1694	33.98
6	150 VMs	50	2	100	1738	34.76
7	200 VMs	50	2	100	1870	37.4
8	500 VMs	50	2	100	2113	42.26
9	1000 VMs	50	2	100	2102	42.04

limited to 2 Mbps and a TDF of 50 was set, in this way the *real* bandwidth is 100 Mbps for each VM (Grau et al., 2008).

Experiment number 2 is a special case: 5 VMs running in a single host with no bandwidth restriction. This is possible because OpenVZ VMs run using a shared file system among containers. Then, the bandwidth would be restricted by the hard drive disk speed and, as result, real time is less than that of experiments 3 and 1. For the time columns, it can be either *clock* or *real*; when there is no TDF, real time is equal to clock time, otherwise, *real* equals *clock* divided by *TDF* (Grau et al., 2008).

Notice that, with the exception of experiment number 2, the real time increases slowly as the number of VMs grows, and stabilizes for 500 and 1000 VMs. This situation disclosures why P2P systems tend to function better as more peers are added (Bharambe et al., 2005; Cohen, 2003; Lin et al., 2007).

After this assessment process, our final test bed was able to run 3000 peers: 1000 were emulated while 2000 remained simulated. This configuration is shown in Figure 6. Emulated peers ran using 50 OpenVZ containers per physical host, while simulated peers ran in PRIME using a cluster of 10 computers with a kernel patch to enable time dilation (Grau et al., 2008).

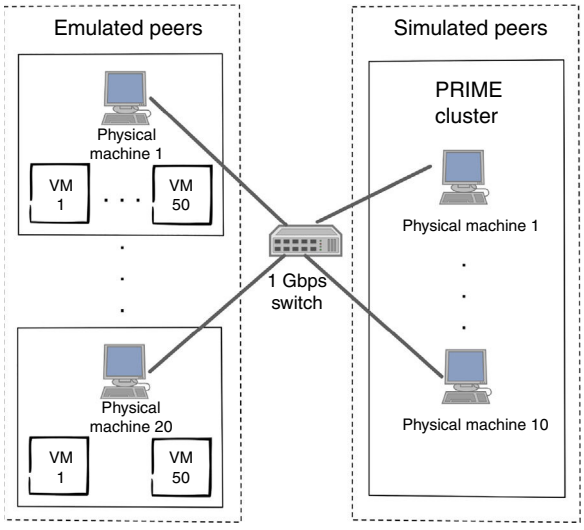

Fig. 6. Simulation/emulation testbed.

```python
from time import time

def TDFtime():
    if getTDF() == 1:
        return time()
    else:
        return time() / getTDF()

def getTDF():
    return 25
```

Fig. 7. TimeDilation.py.

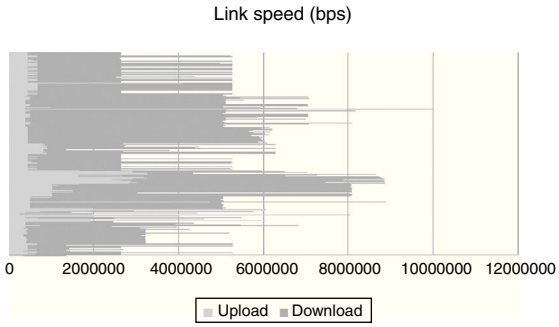

Fig. 8. Sample of different bandwidths.

4.2. Network model for experiments

To create a network model that replicates a real scenario, the data obtained in (Dischinger, Haeberlen, Gummadi, & Saroiu, 2007) was used which, to our knowledge, was the first large-scale measurement study of major cable and DSL providers in North America and Europe. Two types of broadband access networks were considered: digital subscriber line (DSL) and cable. Both link types typically have asymmetric bandwidth, that is, their downstream bandwidth is higher than their upstream bandwidth. These heterogeneous download and upload bandwidths are depicted in Figure 8, x-axis represent the link speed, while y-axis denote hosts.

In Table 2, a summary of the different DSL hosts from major commercial Internet service providers (ISPs) is given to show the heterogeneous nature of the peers used for simulation and emulation in Bit-SiEm. Furthermore, the model includes hosts from different regions and therefore network delays are varied, which gives realism to the model. This can be observed in Figure 9.

Similarly, several hosts connected to Cable ISPs were used in the network model to include other conditions that are common in the industry, for instance, over-subscription in the access network which causes different level of service experienced by customers. Table 3 summarizes the information for Cable ISPs.

With this information, a network scenario comprised of 516 routers and 3000 peers was constructed. Observe that each of these hosts had specific characteristics such as download bandwidth, network delay and upload bandwidth to recreate a real heterogeneous network. The proposed platform in this document considers network models with this asymmetric property allowing the definition of hosts with varied topologies.

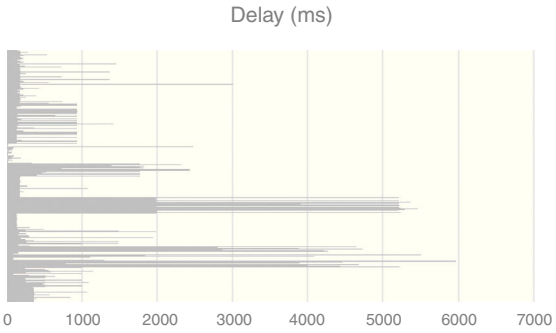

Fig. 9. Sample of different network delays.

4.3. Experiments results

One of the benefits of emulation is that real applications can be tested in a controlled environment. In this case, a genuine BitTorrent application written in Python ran in hundreds of VMs. For the simulated peers, we applied reverse engineering to implement all the related algorithms within PRIME. Furthermore, the peer-wire protocol was implemented in such detail that emulated and simulated peers were able to interact and share a 150 MB file among the 3000 hosts in the swarm: 1000 emulated, 2000 simulated. During the experiments, we gathered data from the hosts. Then, we evaluated how the simulation results compared with two mathematical models: (1) number of seeds as function of time and (2) quality of neighbors list; this needed to be done to validate our work with previous research and assure the legitimacy of the proposed platform. Finally, to show the platform's flexibility, some experiments with other BitTorrent variants were executed.

5. Number of seeds as function of time

To evaluate if our platform was consistent with the behavior of an authentic BitTorrent swarm, we compared the simulation results with a fluid model developed in (Qiu & Srikant, 2004). The notation for this mathematical model is the following:

$x(t)$	number of downloaders at time t
$y(t)$	number of seeds at time t
λ	Peers arrival rate using a Poisson process

Table 2
DSL hosts used for the experiments.

ISP	Ameritech	BellSouth	BT	PacBell	Qwest	SWBell
Company	AT&T	AT&T	BT group	AT&T	Qest	AT&T
Region	S + SW USA	SE USA	UK	S + SW USA	W USA	S + SW USA
Host measured	113	155	173	158	97	397
Offered BW (bps)	768K, 1.5M, 3M, 6M	768K, 1.5M, 3M, 6M	2–8M	768K, 1.5M, 3M, 6M	256K, 1.5M, 7M	768K, 1.5M, 3M, 6M

Table 3
Cable hosts used in the experiments.

ISP	Charter	Chello	Comcast	Road runner	Rogers
Company	Charter Comm.	UPC	Comcast	Time Warner	Rogers
Region	USA	Netherlands	USA	USA	Canada
Host measured	114	120	118	301	148
Offered BW (bps)	3M, 5M, 10M	384K, 1.5M, 3M, 6M, 8M	6M, 8M	5M, 8M	128K, 1M, 5M, 6M

μ Uploading bandwidth of a given peer
c Downloading bandwidth of a given peer
θ Rate at which downloaders abort download
Υ Rate at which seeds leave the system
η Effectiveness of file sharing

With this notation, a deterministic fluid model for the growth of the number of seeds is given by

$$\frac{dy}{dt} = \min\{cx(t), \mu(nx(t) + y(t))\} - \gamma y(t) \quad (1)$$

While effectiveness of file sharing is specified by $\eta = 1 - P\{\text{downloader i has no piece that the connected peers need}\}$. Ten replicas were run using parameters showed in Table 4. We computed confidence intervals (CI) to assure consistency between replica results. In Figure 10 results for these experiments are shown; we plotted $y(t)$, that is, the number of seeds as a function of time. For each time t we computed η dividing the number of peers that received at least one piece by the number of total connections, that is, the rate at which a downloader i has at least one piece that connected peers need. Notice that the fluid model from (Qiu & Srikant, 2004) and our data are comparable, differences are minimal.

Table 4
Parameters used for the fluid model.

Parameter	Value
Hosts	3000
Pieces	600
λ	Flash crowd
c	Data from (Dischinger et al., 2007)
μ	Data from (Dischinger et al., 2007)
θ	0
η	Computed for each t
Initial seeds	1
Υ	0

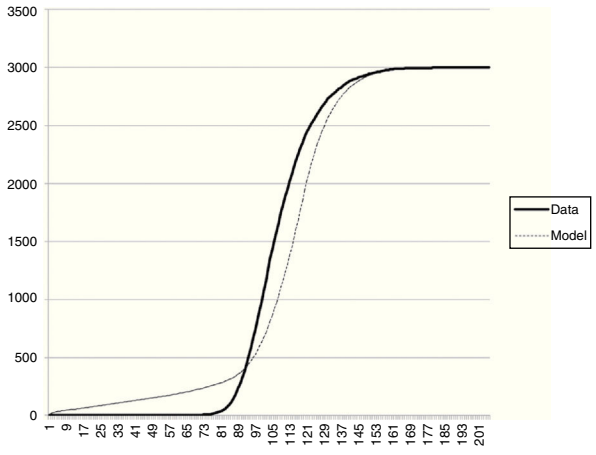

Fig. 10. Number of seeds, as a function of time.

6. Quality of the neighbors list

In Aguilar et al. (2007), authors measured the quality of the neighbors list. A high quality list has a high success rate, where a success is considered if a peer in the list provides at least one piece. According to this research, successes can be modeled using a Beta-Binomial distribution, and given by:

$$P_k = \binom{N}{x} \xi^k (1-\xi)^{N-k}, \quad k = 0, 1, \ldots, N \quad (2)$$

where ξ is a random variable, and from (Rodriguez-Dagnino & Bustos-Gardea, 1998), Beta-Binomial is given by

$$P[X = k] = \frac{(N/x)\Gamma(\alpha+\beta)\Gamma(\alpha+k)\Gamma(N+\beta-k)}{\Gamma(\alpha)\Gamma(\beta)\Gamma(N+\alpha+\beta)} \quad (3)$$

Also, we considered the estimation of the parameters using the moment method, where α and β are defined as follows:

$$\hat{\alpha} = \frac{-\bar{X}[S^2 - \bar{X}(N - \bar{X})]}{NS^2 - \bar{X}(N - \bar{X})} \quad (4)$$

$$\hat{\beta} = \frac{\alpha(N - \bar{X})}{\bar{X}} \quad (5)$$

where $\bar{X}$ and S^2 are the mean and variance respectively. With this information, we determined that the chi-square test passed at a 5% significance level and 50 degrees of freedom when compared against simulation results for our platform. The probability density functions can be seen in Figure 11, results for Binomial, Beta Binomial and our experiments are shown.

7. An experiment with BitTorrent variants

One of the central advantages of our proposed platform is that peers can run in virtual machines to emulate real BitTorrent clients. This translates to the possibility of running any BitTorrent variant and creates experiments to test and evaluate different features of specific versions.

To assess the versatility of the proposed platform, three BitTorrent versions were examined. First, the original specification by Cohen (2003). Also, two BitTorrent versions built to distribute content in large-scale enterprise networks: ABT-K which stands for Augmented Kind BitTorrent (Aguilar et al., 2007), and ABT-M, a multicast-enabled variant of the software

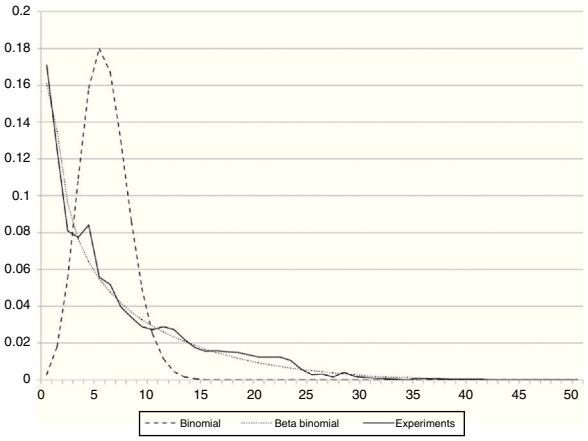

Fig. 11. PDF for the quality of the neighbors list.

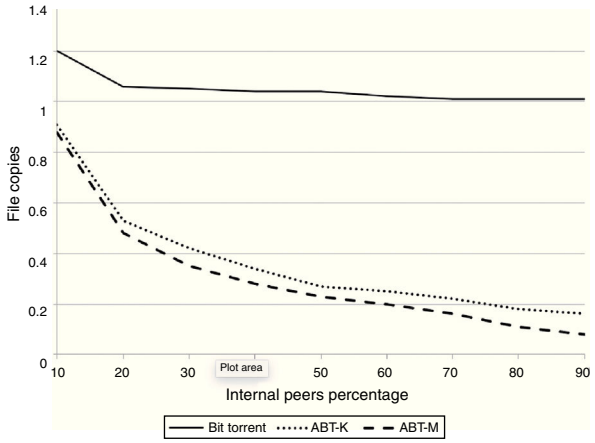

Fig. 12. WAN link utilization.

(Bustos, Aguilar, Makki, & Ege, 2008). The idea behind both augmented versions is that, with some modifications to the protocol, download times can be improved in enterprise networks if compared to the original BitTorrent specification. The details of these versions are out of the scope of this document, but we are confident that the implementation of different variants shows the flexibility of the platform.

The first version was downloaded directly from the BitTorrent web site (BitTorrent, 2014b). For ABT-K and ABT-M, the downloaded version was taken and modified to include the additions that authors proposed in their research.

For the experiments, three thousand emulated peers ran using 50 OpenVZ (Zhao et al., 2010) containers per physical host, the tracker was installed in a separated machine. Next, ten experiments with the modified software clients were performed. Confidence intervals were calculated to ensure consistency between the different experiment results.

Since ABT-K and ABT-M are optimized for enterprise networks, when network address translation (NAT) is widely used, we regulated the number of peers inside a LAN, referred as internal peers, from 10% to 90% (300 up to 2700 peers) in tenth increments to simulate several scenarios; this information is shown in Figure 12 on the *x*-axis. Also, two seeds outside the enterprise network with public IP addresses are initially available.

To collect results and data files from the experiments, a network file system (NFS) was installed and configured in the physical machines; an advantage of OpenVZ over other virtualization techniques: the file system of the guest host can be accessed from the VMs.

To measure efficiency in terms of WAN bandwidth usage, in Figure 12 the average number of file copies per internal host is shown. This value is obtained dividing the total pieces sent and received thru the WAN link by the product of the number of internal hosts and the number of file pieces; for instance, a centralized approach would have a value of 1 because each peer downloads one copy from the Internet. For BitTorrent, notice that this value is at least 1 and up to 1.20, that is, the downloaded copy plus sharing price (Cohen, 2003); this also means that the file was downloaded from the seeds, else, it would have been closer to 2.0 since BitTorrent uses a tit-for-tat approach and internal peers would be constantly choked (Bustos et al., 2008).

Also, notice that bandwidth utilization for ABT-K and ABT-M are much less than BT, although ABT-M performs better. Remarkably, for 90% of internal peers this value is 0.08 for ABT-M, meaning that only 8% of the required bandwidth is used when compared to any centralized approach, or even to the use of a content distribution network (Katsaros et al., 2009).

8. Conclusions

In this article, a platform with simulation and emulation capabilities for the BitTorrent protocol was presented. Since it is based on PRIME, packet-level simulation is feasible. To our knowledge, there is no other large-scale tool capable of implementing the protocol in such detail that the interaction between simulated *and* emulated peers is possible; simulated peers seamlessly exchange messages with real BitTorrent instances running in Linux containers. Furthermore, a test bed with 3000 peers running on top of a *real* network model was configured. With this setup, we compared the experimental results with mathematical models to guarantee the validity of our proposed framework. All this makes the proposed platform useful not only for simulation, but also for other uses such as creation of application-level protocols; that is, use BiT-SiEm not only in academic and scientific research scenarios, but also in industry for software development and testing. Besides, two additional BitTorrent variants were emulated, which corroborates the versatility of the platform.

Future work includes the creation of a complete formal specification of the BitTorrent protocol to support research for this P2P content distribution approach.

Conflict of interest

The authors have no conflicts of interest to declare.

Acknowledgements

To Marcel Dischinger for providing the information from his work (Dischinger et al., 2007) to model our network. To Jason Liu for the tutorials on PRIME.

References

Aguilar, A., Ege, R. K., Makki, K., & Bustos, R. (2007). Enabling peer cooperation in private local area networks using BitTorrent. In *12th IEEE symposium on computers and communications* (pp. 615–622).

Bharambe, A. R., Herley, C., & Padmanabhan, V. N. (2005). *Analyzing and improving BitTorrent performance. Technical report MSR-TR-2005-03*. Microsoft Research.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al. (2003). Xen and the art of virtualization. *Operating Systems Review ACM*, *37*(5), 164–177.

Bindal, R., Cao, P., Chan, W., Medved, J., Suwala, G., Bates, T., et al. (2006). Improving traffic locality in BitTorrent via biased neighbor selection. In *26th IEEE proceedings – international conference on distributed computing systems, 2006. ICDCS 2006* (p. 66). IEEE.

(2014). *BitTorrent*.. Retrieved from http://www.BitTorrent.com/

(2014b). *BitTorrent specification*. Retrieved from http://wiki.theory.org/BitTorrentSpecification

Bustos, R., Aguilar, A., Makki, K., & Ege, R. K. (2008). Multicast-P2P content distribution in large-scale enterprise networks. In *IEEE symposium on computers and communications. ISCC 2008* (pp. 487–494). IEEE.

Chen, X., Chu, X., & Liu, J. (2010). A lightweight emulator for BitTorrent-like file sharing systems. In *2010 IEEE international conference on communications (ICC)* (pp. 1–5). IEEE.

Cohen, B. (2003). *Incentives build robustness in BitTorrent*. pp. 68–72. *Workshop on economics of peer-to-peer systems* (Vol. 6). Retrieved from http://www.BitTorrent.org/BitTorrentecon.pdf

Dike, J. (2000). A user-mode port of the Linux kernel. In *Proceedings of the 4th annual Linux showcase & conference*.

Dischinger, M., Haeberlen, A., Gummadi, K. P., & Saroiu, S. (2007). Characterizing residential broadband networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, IMC'07* (pp. 43–56).

Erazo, M. A., Li, Y., & Liu, J. (2009). SVEET! A scalable virtualized evaluation environment for TCP. In *Proceedings of the 5th international conference on testbeds and research infrastructures for the development of networks and communities (TridentCom'09)* (pp. 1–10).

Eger, K., Hoßfeld, T., Binzenhöfer, A., & Kunzmann, G. (2007). Efficient simulation of large-scale P2P networks: Packet-level vs. flow-level simulations. In *Proceedings of the second workshop on use of P2P, GRID and agents for the development of content networks* (pp. 9–16).

Grau, A., Maier, S., Herrmann, K., & Rothermel, K. (2008). Time jails: A hybrid approach to scalable network emulation. In *22nd workshop on principles of advanced and distributed simulation. PADS'08* (pp. 7–14). Rome, Italy: IEEE.

Izal, M., Urvoy-Keller, G., Biersack, E., Felber, P., Al Hamra, A., & Garces-Erice, L. (2004). Dissecting BitTorrent: Five months in a torrent's lifetime. In *Passive and active network measurement*. pp. 1–11.

Katsaros, K., Kemerlis, V. P., Stais, C., & Xylomenos, G. (2009). *A BitTorrent module for the OMNeT++ simulator*.

Lin, M., Fan, B., Lui, J. C., & Chiu, D. M. (2007). Stochastic analysis of file-swarming systems. *Performance Evaluation*, *64*(9–12), 856–875.

Liu, J. (2008). A primer for real-time simulation of large-scale networks. In *41st annual simulation symposium 2008. ANSS 2008* (pp. 85–94). IEEE.

Liu, J., Li, Y., & He, Y. (2009). A large-scale real-time network simulation study using prime. In *Proceedings of the 2009 winter simulation conference*.

Microsoft Corporation. (2007). *BitTorrent simulator (Version 1.0)*.. Retrieved from http://research.microsoft.com/en-us/downloads/20d68689-9a8d-44c0-80cd-66dfa4b0504b/

Naicken, S., Basu, A., Livingston, B., & Rodhetbhai, S. (2006). A survey of peer-to-peer network simulators. In *Proceedings of the seventh annual postgraduate symposium*.

Pouwelse, J., Garbacki, P., Epema, D., & Sips, H. (2005). The BitTorrent P2P file-sharing system: Measurements and analysis. In *Proceedings of the 4th international conference on peer-to-peer systems* (pp. 127–143).

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

Rodriguez-Dagnino, R., & Bustos-Gardea, R. (1998). Beta-binomial video traffic modeling for the knockout ATM multicasting switch. In *Proceedings of SPIE, Vol. 3530* (pp. 357–368). The International Society for Optical Engineering.

VMWare. (2009). *Workstation*. Retrieved from http://www.vmware.com/products/desktop/workstation.html

Zhao, Y., Zhang, G., & Cui, J. (2010). Optimizing performance of packet capture in virtual containers of OpenVZ. In *Proceedings of the 2010 spring simulation multiconference*. San Diego, CA, USA: Society for Computer Simulation International.