Versatile Configuration and Deployment of Realistic Peer-to-Peer Scenarios

George Milescu, Răzvan Deaconescu, Nicolae Țăpuș
Automatic Control and Computers Faculty
University POLITEHNICA of Bucharest
Emails: {george.milescu,razvan.deaconescu,nicolae.tapus}@cs.pub.ro

***Abstract*—With the advance of Peer-to-Peer solutions, research and commercial players have shown interest in enhancing local client and overall swarm performance in order to improve content distribution and user satisfaction. Protocol measurements and careful client and swarm behavior analysis are required to provide valuable information on improving performance. In this paper, we present a Peer-to-Peer testing infrastructure that enables easy deployment of complete and controlled BitTorrent swarms. The infrastructure allows a variety of realistic scenarios to be run with the ability to configure characteristics such as client type, bandwidth management, churn rate and number of connections.**

***Index Terms*—Peer-to-Peer, BitTorrent, infrastructure, automation**

I. Introduction

The last 20 years have seen the birth and expansion of the Internet from a small network of academic and government institutions to a global network spanning borders, cultures and homes. With the ever increasing network bandwidth, file and data transfer is the Internet service that is responsible for the largest chunk in the Internet backbone. HTTP and Peer-to-Peer systems are nowadays the main bandwidth consumers in the Internet, with video content as the most common type of traffic going through the Internet links [1].

Peer-to-Peer systems have emerged as the most suitable solution to capitalize on the huge unexploited network bandwidth available on the Internet. Since the inception of Napster in the late '90s, Peer-to-Peer systems have evolved to a variety of solutions and applications that continuously stir the interest of institutions (be them academic or commercial) across the world.

The most eloquent example of Peer-to-Peer systems' success story is the BitTorrent protocol, currently responsible for the larges chunk in Internet Peer-to-Peer traffic [1]. With simple, yet highly effective features such as optimistic unchoking, tit-for-tat and rarest-piece first, the BitTorrent protocol is one of the best suited solutions for large data distribution. Recent research focus has been in integrating features such as social networking, reputation management, video streaming as core features or overlays on top of the protocol.

In this paper we present a Peer-to-Peer software testing infrastructure that provides flexibility, control and automation. The infrastructure allows deployment of realistic P2P scenarios, gives full control to the experimenter and automates the interaction with Peer-to-Peer clients. Our solution provides the means to define an array of input variables for scenarios: number of peers, leechers, seeders, bandwidth limitation, client type, number of connections, intervals of activity (churn rate). Client output information is automatically retrieved as log files and rendered through statistical processing.

The testing infrastructure uses shell scripts and configuration files to setup and manage BitTorrent client swarms. The use of shell scripts allows easy integration with BitTorrent clients, takes advantage of the SSH (Secure Shell) protocol for remote system control and provides interaction with tools for parsing and processing output information.

We have successfully deployed and used the testing infrastructure both on a physical environment (consisting of 10 hardware nodes) and on a virtualized environment (consisting of 100 OpenVZ [4] containers) running on top of the physical environment. We have been able to run scenarios containing 100 hosts, each running a BitTorrent client instance. The use of OpenVZ allows lightweight virtualization and easy simulation of complete systems on top of a small number of hardware nodes.

II. Related Work

Current research focus regarding Peer-to-Peer systems and protocols uses carefully crafted experiments and network simulators.

A survey of the use of Peer-to-Peer simulations has been undertaken by Naicken et al. [13]. The authors surveyed papers and collected information regarding the use of simulators for Peer-to-Peer systems. Five criteria had been used to evaluate the simulators: simulation architecture, usability, scalability, statistics and underlying network simulation. A large number of custom simulators were detected to have been deployed, the main cause for that being assumed to be the lack of proper statistics output. The authors criticize the use of NS-2 as a simulator for Peer-to-Peer systems and provoke discussion to help build a consensus on the common platform for Peer-to-Peer research.

One of the best places to look for deploying network experiments, also heavily used by Peer-to-Peer researchers, is Planet Lab [6]. With more than 1000 nodes and 500 sites spread all over the world and healthy documentation, PlanetLab offers a suitable environment for Peer-to-Peer experiments. As user nodes are virtualized through the use

 ISBN: 978-1-61208-133-5

of Linux-Vserver, experimenters have complete control over their system and its resources. The user may deploy a given set of tests or use PlanetLab as an underlying layer for a testing infrastructure (such as the one presented in this article) and be able to deploy a realistic environment for various scenarios.

NS-2 [7] is one of the most popular network simulators. Thorough documentation, continuous development over the past two decades and a rich set of features have ensured NS-2 as a prime candidate for network experiments. However, as Naicken et al. [13] conclude, NS-2 is particularly useful for detailed modelling of the lower network layer, a characteristic that is of little interest to Peer-to-Peer researchers, though it has been often used in Peer-to-Peer experiments.

We consider PlanetLab [6] and NS-2 [7] to be located at separate poles when discussing about the purpose of Peer-to-Peer experiments. PlanetLab and virtualized environments allow deployment of realistic scenarios, and collected valuable realistic information, but lack scalability. On the other hand, NS-2 and network/P2P simulators allow simulation of large number of nodes (even to the degree of millions) while failing to provide accurate data about client behavior and detailed statistics. We consider that, given the nature of the BitTorrent protocol as a solution for content distribution, realistic (or even real) environments are appropriate for experiments regarding BitTorrent swarms.

Dinh et al. [11] have used a custom network simulator (dSim) for large scale distributed simulations of P2P systems. The authors have been able to simulate approximately 2 million nodes for Chord and 1 million nodes for Pastry. Similar work has been presented by Sioutas et al. [16]. Video streaming in Peer-to-Peer networks has been simulated as described by Bracciale et al. [9] using a custom simulator dubbed OPSS.

With respect to BitTorrent simulators and closer to the purpose of this article, Pouwelse et al. [14] have undertaken a large BitTorrent measurement study spanning over several months on real BitTorrent swarms (provided by the Suprnova tracker). Data was collected through HTML and BitTorrent, (ab)using scripts, from the central tracker and BitTorrent clients. A similar approach has been employed by Iosup et al. [12]. The authors have designed and implemented MultiProbe, a framework for large-scale P2P file sharing measurements on the BitTorrent protocol. MultiProbe has been deployed in real swarms/environments and collected status information from BitTorrent peers and subject it to analysis and dissemination.

Our testing infrastructure is deployed on a hardware experimental setup (similar to a local PlanetLab) presented in an earlier paper [10]. Instrumented BitTorrent clients, logging facilities and an OpenVZ lightweight virtualization solution are basic block on top of which the software testing infrastructure was developped and used.

III. Design and Architecture

A. Design goals

The use of network simulators for creating controlled environments has been an easy solution for achieving BitTorrent measurements. However, real BitTorrent clients behave differently from simulators and the network protocol stack has an important influence on the outcome of a scenario.

Considering the decreasing cost of hardware and the improvements in virtualization solutions, running network emulations with hundreds of nodes, each having a dedicated instance of an operating system, is an achievable objective. In this sense, Rao et al. [15] concluded that results gathered from BitTorrent experiments performed on clusters are realistic and reproductable.

The proposed infrastructure for controlling peer-to-peer clients aims at providing an extensible and adaptable tool for experiment setup, execution and analysis. It has four primary goals, allowing it to be used in a large variety of scenarios.

The first goal is to provide an **extensive tool for managing both clients and log files** during experiments. Running scenarios that include a large number of clients (up to a few hundred) requires a control mechanism for starting, monitoring and stopping clients in a short time-frame. Most of the scenarios result in a collection of log files, at least one log file per client or per machine. Collecting and analysing these log files, considering the large number of remote machines, has to be automated.

The second goal is to use a **common interface for accessing remote systems**. The nodes on which clients run must consist of various Linux or Unix distributions, and, most likely, the machines are not administrated by the user running the scenarios. Also, the nodes could be hardware or virtual machines. A common access interface to this heterogeneous node infrastructure is needed, and the interface must not require administrative privileges for accessing the remote nodes.

The third goal is to offer **support for bandwidth control**. Cluster computers are generally connected with 1Gbit/s or faster network connections. These types of connections are not common for end-users. In order to provide realism to the experiments, the infrastructure needs to offer a mechanism for controlling the amount of bandwidth each client can use. Having the bandwidth control integrated in the infrastructure offers the advantages of fine-tuning the scenarios and recreating a wide range of network environments.

The last goal is to **allow the user to introduce churn in the environment**. Starting and interconnecting P2P clients is only the first step towards reproducing a real-life scenario. Two of the elements that characterize real swarms are churn and population turnover. Both translate into clients joining and leaving the network at different time intervals. Controlling the periods when each client is connected to the network gives the user the freedom of creating a variety of scenarios, from a controlled flash-crowd to a swarm close to extinction.

As mentioned, the proposed infrastructure provides a tool for experiment setup, execution and analysis. It is the experimenter's task to design the experiment parameters and to validate the used models against simulated results or other real-life measurements.

 ISBN: 978-1-61208-133-5

B. Design elements

From a design point of view, the infrastructure uses four concepts: campaign, scenario, node and client.

A **campaign** consists of a series of experiments, each experiment being independent of others and having associated a specific type of data processing. The difference between a campaign and an experiment resides in the fact that results from an experiment may be plotted on a single graph, while results from a campaign need a deeper analysis. Multiple experiments may be included in a campaign. If an experiment needs to be run multiple times (to retrieve significant results), it can be included multiple times in the same campaign.

A **scenario** coresponds to a single experiment. It is associated with a specific type of data processing and its results are generally presented on a single graph.

A **node** is one of the infrastructure machines. It can be a virtual or a hardware machine. The user running the experiments needs to have access to the nodes both for experiment deployment and execution and for bandwidth control.

A **client** is a single instance of a peer. The infrastructure is designed to run a single client on each node, in order to reproduce the real-life execution context for P2P clients.

Campaigns and scenarios each use configuration files that include a complete specification of the experiments. The campaign configuration file specifies the scenarios included in the campaign. The scenario configuration file includes all nodes that are part of the infrastructure used to execute the experiment; for each node, the configuration file defines access parameters, client types, churn and the bandwidth limitations.

C. Architecture overview

The local machine is used to control the infrastructure. It stores the infrastructure scripts, configuration files, and campaign output. It may also store code or executable files for P2P clients. The infrastructure scripts copy required files from the local machine to remote nodes, set up the enviroments and start the clients. After the experiment ends, the results (log files) are brought back from the remote node to the local machine.

The testing infrastructure uses a modular architecture. Some of the modules are *generic* (such as the module that parses the configuration files); other modules are *node or client specific* (for example the module that parses the log files obtained from a client). From a different point of view, part of the modules are executed on the *local machine*, others on the *remote host*.

The infrastructure architecture is depicted in Figure 1. The *run_campaign* component reads the campaign configuration file and executes each of the specified scenarios. After a scenario is executed, its results are processed, and the next scenario is run. At the end of the campaign, campaign results may be published as a web-page for preliminary analysis.

run_scenario, the central point of the intrastructure, is responsible for managing all activities related to the execution of an experiment. Its specific components will be detailed in the following section.

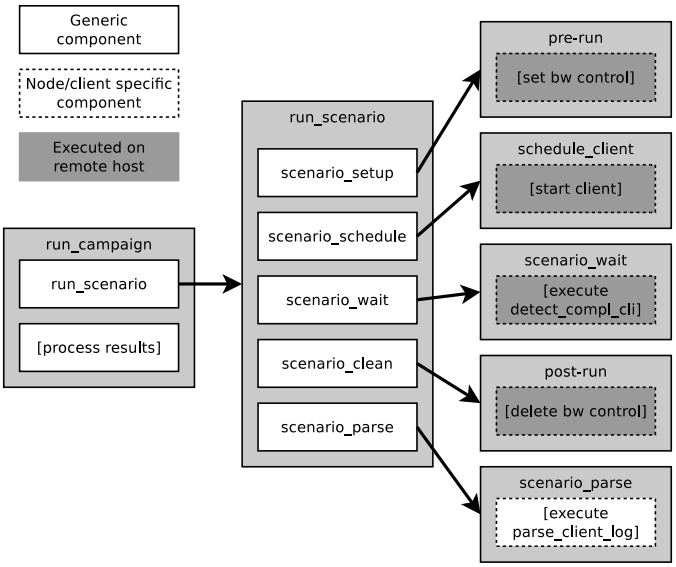

Figure 1. Infrastructure design overview. The components use "_" between the component names. The actions, that are not directly included in a component, are placed between [ ]

D. Architecture details

Figure 1 presents the architecture overview. The central point of the infrastructure is *run_scenario*, the component responsible for executing a scenario. This section details its components and explains the mechanisms it uses to deploy and execute scenarios.

After the scenario configuration file is parsed, each of the nodes will be prepared for the experiment by *scenario_setup*. This component is detailed in Figure 2. The first step is to synchronize the local infrastructure scripts with the remote host. The synchronization phase cleans up the remote host and ensures that consecutive scenarios do not influence each other.

A local node-specific configuration file, including parameters related to that node, is created for each of the nodes specified in the scenario configuration file. The node-specific configuration file is then copied to the remote host. This file is used for inter-component communication between the local-executed and the remote-executed components.

The *pre-run* component prepares remote host environments for the experiment. This component parses the node-specific configuration file and applies settings required for the scenario. The *pre-run* component also handles bandwidth limitations.

The *schedule_client* component schedules client executions on the remote host. Based on the node-specific configuration files stored on the remote host, *schedule_client* starts and stops the client to simulate the specified churn. The client lives until the *scenario_wait* component detects completion of the experiment, after which it may be stopped. The client will not be immediatelly stopped, as the infrastructure waits for all the clients to complete the experiment before stopping them.

After all clients complete the experiment, each node will be cleaned up by the *scenario_clean* component, as presented

 ISBN: 978-1-61208-133-5

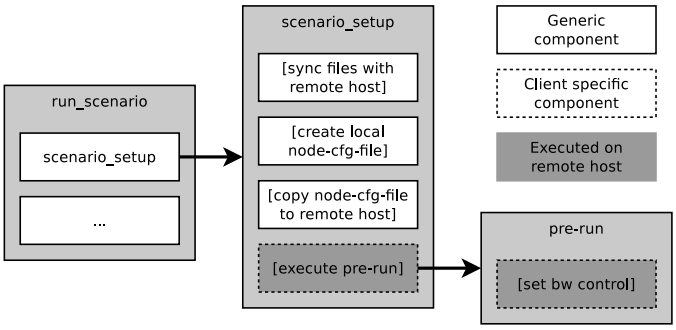

Figure 2. Detailed scenario_setup components. The components use "_" between the component names. The actions, that are not directly included in a component, are placed between [ ]

in Figure 3. This component stops the client and retrieves the remote log files. A *post-run* component is then executed reverting all settings applied by *pre-run* to ensure that consecutive scenarios do not influente each other. In the end, the remote node-specific configuration file is deleted and local infrastructure scripts are synchronized to the remote host to clean any temporary file.

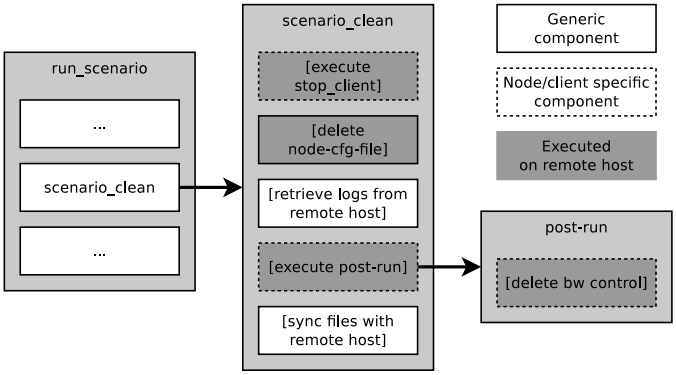

Figure 3. Detailed scenario_clean components. The components use "_" between the component names. The actions, that are not directly included in a component, are placed between [ ]

Information from clients is stored in log files. The last stage of the scenario execution, *scenario_parse*, translates the client-specific log format to an unified format used by the processing stage.

Log files are used to analyse the evolution of various client parameters during each scenario by storing periodic status information, such as download speed, number of connections or ratio. Specialized log files could also be created, if the clients are instrumented, and gather detailed periodic information (for example information consisting of instant per-peer download speed and upload speeds).

Specially designed R scripts are invoked in the post-processing phase. Using information stored in the unified log files format as input, the R scripts output graphical representation of the evolution of client parameters such as download speed.

IV. Infrastructure Implementation

A. Node and client specific components

Part of the components presented in Section III are node or client specific and will be detailed in this section.

The first client- and node-specific component is *pre-run*. One of its main tasks is to configure the bandwidth limitations on the remote host. Three solutions have been tested to enforce the limitations (the solutions will be detailed in IV-D):

- controlling bandwidth at the operating system level
- controlling bandwidth at the process level
- controlling bandwidth within the P2P client

pre-run is both client and node-specific. Some clients do no offer bandwidth control, while bandwidth used by some virtualization solutions can not be limited at the operating system level.

The interface between the infrastructure and the BitTorrent clients is composed of three client-specific components: *scenario_schedule*, *scenario_wait* and *scenario_clean*. The only infrastructure requirements for BitTorrent clients are to provide a CLI (Command-Line Interface) interface to run on top of a Linux system and to offer runtime-generated log messages..

scenario_schedule is responsable for starting the clients on the remote nodes. The *start client* script is client-specific. This script also prepares the running environment prior to starting the client.

After a client starts, the *scenario_wait* component monitors it to detect the experiment completion. The detection phase is dependant on both the goal of the experiment, and on the type of client used. A remote client is considered completed either by reaching a run-time state defined by the scenario or when the churn configuration implies a final stop action (see Section IV-C). The runtime-based completion detection requires the infrastructure to detect the completion of the experiment based on the messages the client logs while it runs. As each client has a different log format and specific experiments require special log messages, *scenario_wait* is adapted to user needs. Given the generic architecture, the infrastructure may be used for multiple types of experiments, targeting download performance, epidemic protocol measurements, user behavioral patterns, etc.

The *scenario_wait* component causes the command station to wait for all remote clients to complete the experiment. Subsequently, log files from remote clients are retrieved to the command station and parsed. The parsing process results in an unified generic format (consisting of table and matrix structured files) that is used as input for statistical analysis.

After all clients have completed the experiment, the *scenario_clean* component stops them and cleans up the remote host. The script used to stop the client is paired with the script used to start it, and is client-specific. The *post-run* component is used for the clean-up phase. Similar to *pre-run*, it has to revert the settings prior to stopping the experiment; this stage includes deleting the bandwidth limitations. *post-run* is node specific.

 ISBN: 978-1-61208-133-5

The last client-specific component is *scenario_parse*. Each client uses a particular log format that should be transparent to the results processing stage. As mentioned before, a translation is required, from the client-specific log format to an unified format used by the processing stage.

B. Employed technologies

The testing infrastructure implementation is based on shell (Bash) scripts. With support in any Linux operating system, and no requirements for additional software, shell scripts provide an ideal environment for easy deployment and exploitation. Shell scripting offers access to a flexible set of tools for parsing client output logs and automating tasks.

The common interface used to access remote systems is based on the SSH protocol. Although file transfers are available via SCP (Secure Copy), the rsync protocol was preferred for folder synchronization between different hosts, transferring only the information that was updated.

Statistical analysis in the testing infrastructure is achieved through the use of automated R language scripts. A powerful tool for processing large amounts of data, R can also do graphical post-processing.

With the exception of scripts used to run a campaign or a scenario or for post-processing, all other scripts are run on the remote systems. The scripts running a campaign or a scenario parse the configuration files on the command station and use SSH to command the scripts on the remote systems. The remote system scripts prepare the node for the experiment and manage the P2P clients (start, monitor, stop).

C. Churn simulation

One of the main goals of the proposed infrastructure is to allow the user to introduce churn in the environment by controlling the periods when each client is connected to the network. An array of intervals included in the scenario configuration file specifies the on-off behaviour for each of the clients. The *schedule_client* control script uses the UNIX signals SIGTOP and SIGCONT to suspend and resume the client processes at the specified moments of time. The churn model (specified the array of time intervals) has to be provided by the user.

D. Bandwidth limitation

As mentioned in IV-D, three solutions regarding bandwidth limitation have been tested.

The first solution is using the `tc` [2] (traffic control) Linux tool, allowing a variety of limitation algorithms implemented at the kernel level. Due to particularities of the OpenVZ implementation, `tc` cannot be currently used as a bandwidth limiter between containers.

In order to bypass this issue, client level limitation (also known as rate limiter) was also tested. hrktorrent and transmission clients offer implicit limitation functionality. This approach does have its downsides, as it is less flexible and is process-centric – one cannot limit the total amount of traffic sent by a client (e.g. a combination of P2P and HTTP traffic).

If the client offers no implicit rate limiter, bandwidth control may still be enabled through the use of the `trickle` [8] tool. `trickle` uses a form of library interposition to hook network related API (Application Programming Interface) calls and limit per-process traffic. It has two drawbacks: it is not actively maintained and issues arise when using the `poll` library call; in case of Linux, `epoll` support is absent.

V. Running Experiments

One of the main goals of the testing infrastructure is to relieve the experimenter of the burden of experiment management and monitoring, providing an extensive tool for managing both clients and log files. As much of the experiment as possible should be run in "background" with little input from the user.

By use of the proposed testing infrastructure, the activity of managing clients, sending commands and collecting information is completely automated, leaving the experimenter with only three tasks to accomplish, sequentially:

1) create the client-specific scripts
2) create the campaign configuration and the scenario configuration files
3) run the campaign startup script

After filling the required information in configuration files, the user running the experiment starts the campaign through the use of a control script that receives, as argument, the name of the campaign configuration file. The script parses the configuration file and creates and manages a swarm for each scenario accordingly. In order to limit the possibility of the user accidentally stopping the campaign control script, it is recommanded to detach the running terminal using tools such as `screen`, `nohup` or `dtach`.

After completion of campaign experiments, all output information and R processed graphic files are stored locally, in a campaign-specific folder. This folder contains a sub-set of folders, one for each scenario, that store log and graphics files.

In terms of scalability, we have successfully deployed and used the testing infrastructure on a 100-node virtualized infrastructure [10] running on top of the physical environment. Thus, we were been able to run scenarios containing 100 hosts, each running a BitTorrent client instance.

Figure 4 presents the outcome of such a scenario, comparing the evolution of peer download speed with respect to download percentage in a 90 peer swarm consisting of 50 seeders and 40 leechers. All peers were limited to `8Mbit/s` upload and download speed and shared a 700MB file. As the figure depicts, the clients reached the maximum allowed transfer rate.

VI. Conclusion and Further Work

This article presented a new approach to building an automated infrastructure that allows easy deployment of experimental scenarios involving Peer-to-Peer clients. Main design goals for the infrastructure were providing an extensive tool for managing both clients and log files, using a common interface for accessing remote systems, offering support for

 ISBN: 978-1-61208-133-5

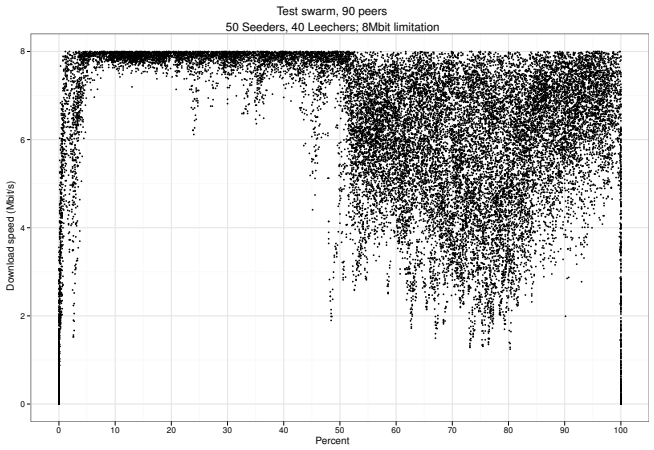

Figure 4. Scenario output: download speed evolution

bandwidth control and allowing the user to introduce churn in the environment. The infrastructure uses a hierarchical set of configuration files and run scripts and has been deployed for a variety of Peer-to-Peer experiments.

The main advantage of the proposed infrastructure when compared to other solutions is automation coupled with easy deployment. The use of a single commanding station, shell scripts, SSH and rsync allows the user to rapidly deploy a given scenario. The possible use for deployment of an OpenVZ virtualization allows consolidation – a small number of hardware nodes are used to create a complete virtualized framework capable of running sandboxed BitTorrent clients. With the use of Linux specific networking tools, the user may define bandwidth limitation and network topology characteristics in order to simulate realistic scenarios.

Given the flexibility of the client-interface and the provided churn and bandwidth-control features, any given Peer-to-Peer scenario can be deployed using the proposed infrastructure. The definition of the scenario and the validation of the Peer-to-Peer models used to design it are however the experimenter's task.

As of this writing the infrastructure has been up and running for one year. Tracker interaction scripts have been added to allow deployment of experiments consisting of multiple trackers. Various BitTorrent clients (hrktorrent, nextshare, swift) have been configured and deployed to provide valuable information regarding performance. Since the initial implementation new scripts have been added for client monitoring and data processing, proving the flexibility of the infrastructure.

Future plans include heavy usage of the infrastructure in various BitTorrent experiments. Bandwidth limitation is currently limited to client features; we aim to identify how this can be migrated to container level – how can one configure upload/download speed limitation for each container. A medium-time goal is "porting" the proposed infrastructure to run on top of Linux Containers (LXC [3]).

ACKNOWLEDGMENTS

This paper is suported from POSCCE project GEEA 226 - SMIS code 2471, which is co-founded through the European Found for Regional Development inside the Operational Sectoral Program "Economical competivity improvement" under contract 51/11.05.2009, and from the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/6/1.5/S/19.

This work is part of the EU FP7 P2P-Next project [5], aiming to deliver the next generation Peer-to-Peer content delivery platform.

The authors would like to thank Alex Herișanu for providing access to the NCIT cluster systems we have been using throughout our experiments.

REFERENCES

[1] ipoque Internet Studies. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009, accessed 2010.

[2] Linux Advanced Routing & Traffic Control HOWTO. http://lartc.org/, accessed 2011.

[3] Linux Containers - LXC. http://lxc.sourceforge.net/, accessed 2011.

[4] OpenVZ. http://wiki.openvz.org/, accessed 2011.

[5] P2P-Next. http://www.p2p-next.org/, accessed 2011.

[6] PlanetLab. http://www.planet-lab.org/, accessed 2011.

[7] The Network Simulator – ns-2. http://www.isi.edu/nsnam/ns/, accessed 2011.

[8] trickle. http://monkey.org/ marius/pages/?page=trickle, accessed 2011.

[9] L. Bracciale, F. L. Piccolo, S. Salsano, and D. Luzzi. Simulation of peer-to-peer streaming over large-scale networks using opss. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[10] R. Deaconescu, G. Milescu, B. Aurelian, R. Rughiniș, and N. Țăpuș. A Virtualized Infrastructure for Automated BitTorrent Performance Testing and Evaluation. *International Journal on Advances in Systems and Measurements*, 2(2&3):236–247, 2009.

[11] T. T. A. Dinh, G. Theodoropoulos, and R. Minson. Evaluating large scale distributed simulation of p2p networks. In *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 51–58, Washington, DC, USA, 2008. IEEE Computer Society.

[12] A. Iosup, P. Garbacki, J. A. Pouwelse, and D. H. Epema. Correlating Topology and Path Characteristics of Overlay Networks and the Internet. October 2005.

[13] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37(2):95–98, 2007.

[14] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The Bittorrent P2P File-Sharing System: Measurements and Analysis. *Peer-to-Peer Systems IV*, pages 205–216, 2005.

[15] A. Rao, A. Legout, and W. Dabbous. Can realistic bittorrent experiments be performed on clusters? In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1 –10, 2010.

[16] S. Sioutas, G. Papaloukopoulos, E. Sakkopoulos, K. Tsichlas, and Y. Manolopoulos. A novel distributed p2p simulator architecture: D-p2p-sim. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, pages 2069–2070, New York, NY, USA, 2009. ACM.

 ISBN: 978-1-61208-133-5