A Novel Testbed for P2P Networks

Pekka H. J. Perälä[1], Jori P. Paananen[2],
Milton Mukhopadhyay[3], and Jukka-Pekka Laulajainen[1]

VTT Technical Research Centre of Finland,
[1]Kaitoväylä 1, FI-90571 Oulu,
[2]Vuorimiehentie 3, FI-02150 Espoo, Finland,
email: firstname.lastname@vtt.fi
[3]Pioneer Digital Design Centre Ltd, Pioneer House
Hollybush Hill, Stoke Poges, Slough, SL2 4QP, United Kingdom
email : milton@pddresearch.com

Abstract. Peer-to-Peer (P2P) overlay networks have recently gained significant attention in the research community. P2P applications, especially the ones using BitTorrent protocol, often require communication among a large number of peers and thus are particularly challenging to test in a controlled manner, because of the volatility of overlay-network structure with peers going on and off. This paper addresses the issue by introducing a novel testbed that enables intuitive network QoS profile configuration, automated peer deployment and test case execution with keyword driven test automation, as well as wireless network testing with real networks. We evaluate the fitness of the testbed by deploying a P2P video delivery application in the network and running trials while monitoring the application behaviour throughout them. Our results demonstrate the capabilities of the testbed in three test cases with different peer access network configurations. The results verify the correct functioning of the testbed and are the first step on our analysis of the P2P video delivery application. This paper provides valuable information for P2P application developers and testers and enables them to setup up similar environments for advanced testing and research on their applications and protocols.

Keywords: peer-to-peer networks, network emulation, network testing, testbeds, wireless networks

1. Introduction

Peer-to-Peer (P2P) paradigm has gained a significant amount of interest in research community due to the vast possibilities of the technology. With P2P technology it is possible to build scalable applications with high availability of content. BitTorrent [1] is one of the most popular file sharing protocols today and thus is on the focus of this work.

BitTorrent uses a *.torrent* metadata file to describe a file to be transferred. The metadata contains the hash of the pieces of a file, and a list of trackers that keep a

registry for the file. A central tracker keeps track of the peers that have the file described in the metadata. A peer connects to the tracker and receives the list of peers that have the file. After getting the list, the peer connects to other peers via *peer wire protocol* to ask for pieces of the file. The downloading peer contacts the tracker again only if the current peers are not satisfactory. After downloading the whole file, the peer turns into a *seeder*, i.e. a peer with a complete copy of the file. The peers involved in data exchange around a specific file form a *swarm* [1, 2].

Due to the distributed nature of BitTorrent, it has proven difficult to reliably perform reproducible experimentation during the development process. Most often the researchers and developers have to rely on simulations, which have only limited correspondence to the real world. On the other hand live trials with instrumented clients are often laborious to set up and the results are usually not reproducible, although they correspond to real behaviour of users. Therefore our work targets the middle ground of these two by providing a fully controllable environment for early testing of new features of P2P applications.

The major contribution of this work is the novel testbed with a keyword driven automation framework that enables testing distributed applications, especially P2P applications, with ease. It allows centralized control and configuration of the network Quality of Service (QoS) parameters with GUI, centralized control, and automatic test case execution and reporting. The GUI provides interactive diagrams for visualizing the complex network topology and queuing discipline structures. A centralized keyword driven test framework allows automation of installation, execution and reporting of the test cases. We found that the keyword driven automation framework is eminently suitable for testing P2P systems as it satisfies the need to be extremely flexible with its component based modular approach, at the same time it reduces the complexity of writing test cases by allowing test cases to be created using abstract keywords and simple logic. In our trials, we have used Nextshare P2P-application, developed by FP7 project P2P-Next consortium [3] as a System Under Test (SUT). Nextshare is a novel video broadcasting system that is based on BitTorrent and is still under heavy development. This makes it an ideal test subject in our testbed development work. To the best of our knowledge, a testbed with such advanced features has not yet been presented in the literature. Especially, configurable network QoS coupled with wireless access networks is a novel contribution in controlled P2P testing.

The measurements conducted with the testbed demonstrate with three test cases the possibilities the testbed creates for P2P research. The test cases are differentiated based on access network configuration of the peers and otherwise similar scenario is executed in all three cases. This gives us opportunity to observe how tuning a single property affects the whole system. The keyword driven test execution framework takes care of the scenario configuration based on given description. We use the reporting and measurement infrastructure to collect reports on the peers and in this specific case we use one wireless node and monitor network on its perspective.

The rest of the paper is organised as follows. Section 2 discusses the special issues that P2P applications pose to test environments, Section 3 presents the developed testbed in detail. Section 4 gives the results from our experiments. Section 5 presents the related work on the area. Finally, Section 6 concludes the paper and outlines the items for future work.

2. Challenges in Testing P2P Applications

P2P applications are challenging to test and experiment with due to their distributed and volatile nature. Main difficulties in producing a reliable test environment include the following: realistic scenario configuration, measurement and reporting, and network configuration.

P2P-applications, more specifically the ones that use BitTorrent protocol are particularly challenging in the testing point of view due to their distributed nature, hence they often require distributed monitoring system as well. The measurement results gathered from multiple sources must be synchronized in order to enable comparative analysis. Furthermore, if one requires the test results to be reproducible, the nodes of the system must be controlled with permissions to e.g. start and stop the execution of the application under test.

Another issue that needs to be solved is the correspondence of the test environment with the real world. In real P2P networks, such as BitTorrent, the system consists of peers that join and leave the swarm arbitrarily. Again, this issue can be addressed by centralized control that enables launching predetermined patterns for the nodes to join and leave the swarm. Furthermore, depending on the content the users, and thus also the network, behave differently. For example, a very popular piece of content may attract almost instantly an audience of millions of users, which may exceed the whole capacity of initial seeder. However, after more than one peer has a whole copy of the content, high availability is ensured. On the other end of the rainbow is marginal content, which may or may not be available due to lack of interest from the general public. These two examples show the available spectrum of possible test scenarios for emulated P2P testbeds.

Packet switched networks are dynamic by nature, making it difficult to produce simulated or emulated environment that accurately capture the whole of the dynamics. Therefore, simplifications and generalizations are needed to reduce the complexity of the system. It is an important design decision how a particular P2P system is built up and how it changes during a particular trial. We take the aforementioned challenges into account in the design of our system and present a detailed description in the following section.

3. Our Approach

Basically three different approaches to evaluate P2P-networks exist: analytical, simulation-based, and experimental. Experimental approach can further be divided in live network experiments concerning real users and in trials in controlled environment (see the related work section and references therein). In this work we focus on the controlled experimenting, because it provides a few advantages over the other approaches. First, real network equipment may be used. This is an advantage over analytical and simulation approaches, where the network behaviour can only be approximated. Second, the results gathered are reproducible. For example, the trials with real users are hard, if not impossible to reproduce and the results may be difficult to interpret correctly. Third, the network connection characteristics of peers can be

controlled. This enables us to conduct network related research on P2P-networks, i.e. how different network configurations affect the overall performance.

3.1 Testbed Architecture

The general architecture of the testbed, as it is implemented using facilities of two VTT laboratories, is shown in Fig.1. The system uses a centralized control point (admin computer in the figure), where test cases are defined and executed. The control point is used also to configure the network profile i.e. the network emulators in the router machines in the network. Finally, the Converging Networks Laboratory [4] provides multiple wireless access networks that enable research with real equipment and in this respect simplifications are not needed.

3.2 Testbed Components

The laboratory test platform consists of nodes running the applications to be tested and of traffic manipulating routers between the nodes. To restrict the number of real machines needed in the test, and to make their control easier, some nodes are run in virtual machines, while others are in real wireless devices. A centralized monitoring and control of the testbed is provided in order to enable reproducible measurements.

The traffic manipulators are routers or bridges equipped with network emulation functionality, which change the characteristics of the IP packet traffic passing through them. It is possible to emulate e.g. network connection equivalent to a typical home computer with wired connection (a broadband access via DSL or cable modem) in an Ethernet node. Several kinds of network congestion or malfunctioning cases can be created.

The testbed consists of three different types of nodes: Ethernet nodes, virtual nodes, and wireless nodes. The Ethernet nodes are machines with a wired Ethernet connection and virtual nodes are virtual machines running on the Ethernet node machines. Following the example from the Onelab project [5], the Xen paravirtualisation system was used also in the testbed presented in this paper. The CNL laboratory provides following wireless interfaces: WLAN a/b/g/n, WiMAX d/e versions, UMTS/HSDPA cell. These are connected to the rest of the test network via 1 Gbps Ethernet. Emulation of wireless devices and networks is not needed in this case due to existing real infrastructure. These three types of nodes are used simultaneously to create test cases that emulate the heterogeneousness of the real network environment.

Central administration allows remote control of the traffic manipulating routers and nodes and is done from a central administration machine. It runs a web server, which provides a GUI for the traffic manipulator control. A tester can control the testbed with a web browser from any machine that has a network access to the web server.

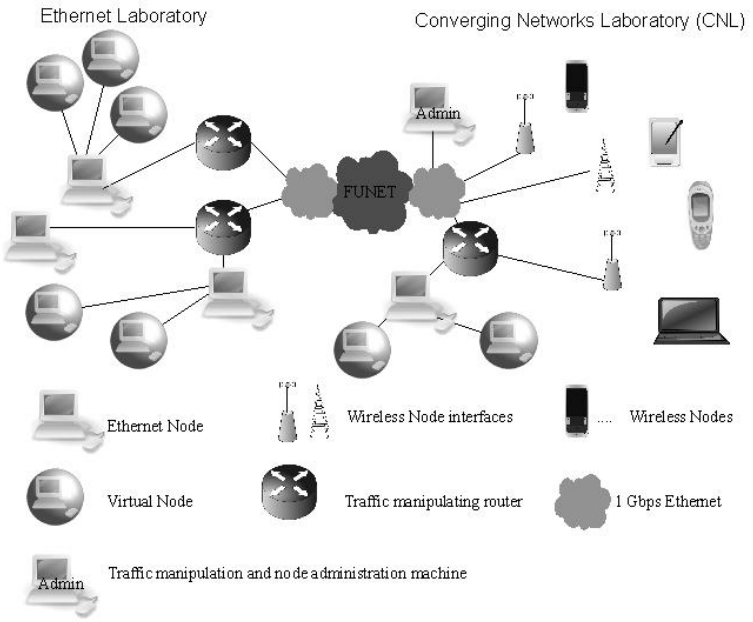

Fig. 1. The network topology of the testbed with the essential nodes.

3.3 Traffic Manipulation

From version 2.6.8 on, the standard Linux kernel includes a network emulation module, Netem [6], our choice for packet manipulation. Netem is one of the queuing disciplines available in Linux (A queuing discipline is a packet queue with an algorithm that decides when to send which packet [7]). Combining Netem with other disciplines, packet rate (i.e. bandwidth), delay, loss, corruption, duplication and re-ordering can be controlled [8]. In principle, these should be enough to emulate any kind of network level QoS (see e.g. [9], pp 3-4).

Possible open source alternatives for Netem would have been NIST Net [10] and Dummynet [11]. NIST Net runs in Linux but is no longer developed (Netem is partly based on NIST Net). Dummynet is part of FreeBSD and OS X firewall ipfw. It is older than Netem and has been perhaps the most popular network emulator. It is used e.g. in P2PLab (see [12] and Ch. 5). Its advantage over Netem (and NIST Net) has been, that it can manipulate both incoming and outgoing packets. By default Netem handles only egress queues. However, handling ingress queues can be achieved in Netem too by using the Intermediate Functional Block pseudo-device (IFB) or Ethernet bridging. On older Linux kernel versions, the timer accuracy of Netem was also inferior to Dummynet. In recent kernel versions, this has been corrected with High Resolution Timers subsystem. Also, the recent versions of Netem offer more features than Dummynet (e.g. packet duplication and corruption) [13].

With iproute2 [14] traffic control (tc) tool, the traffic can be classified with filters into several streams, each manipulated differently. Several types of filters exist. They

may be based on routing rules, firewall marks or any value or combination of values in a packet (like source and destination IP addresses and the upper level protocol types). The two main usages of tc are to configure the queuing disciplines (qdiscs) and to configure packet classification into qdiscs [7].

A qdisc can be either classless or classful. Classful qdisc has a configurable internal subdivision, which allows packet classification, i.e. filtering into multiple streams. A qdisc tree can be built on a classful root qdisc. Since Netem is classless, it requires a separate root, when used with classification.

We use the Hierarchy Token Bucket (HTB, [15]) as a root qdisc. Besides classification, it is needed in packet rate control. It is a root qdisc choice also in [14], the other qdiscs being either classless or too complicated.

A drawback of tc is the relative complexity of its syntax of hierarchical structures, compared to e.g. Dummynet command-line interface. Our system overcomes this with a more intuitive graphical interface.

3.4 A Graphical Front End for Traffic Manipulation Control

To ease the control of traffic manipulation, a centralized graphical front end is provided for examination of the topology of the test network and for manipulation of the emulation models in the traffic manipulating routers. The GUI runs on a web server in the central administration machine.

The control and information data are exchanged via SSH-sessions between the administration machine and the manipulators and nodes (it is ensured by filtering, that the SSH-sessions are not manipulated).

3.4.1 Usage

The start page of the GUI shows a diagram of test network topology – the nodes, routers and different network types (Fig 2.). Choosing a router on the diagram displays a dialogue to launch the information and the queuing discipline model pages of the router.

The router information page provides functionality to fetch the link, routing and queuing discipline (qdiscs, classes, filters) information from the router via SSH. The queuing discipline model page enables examination of qdisc hierarchies created for the network interfaces of a router. A qdisc model is presented as a tree diagram (Fig.3.). Qdiscs, their classes and associated filters are shown as the tree leaves (the leaf parameters are shown as tooltips). The user can add or delete leaves or change their parameters. When the model is ready, the user can send it - i.e. the generated tc-commands - to the router.

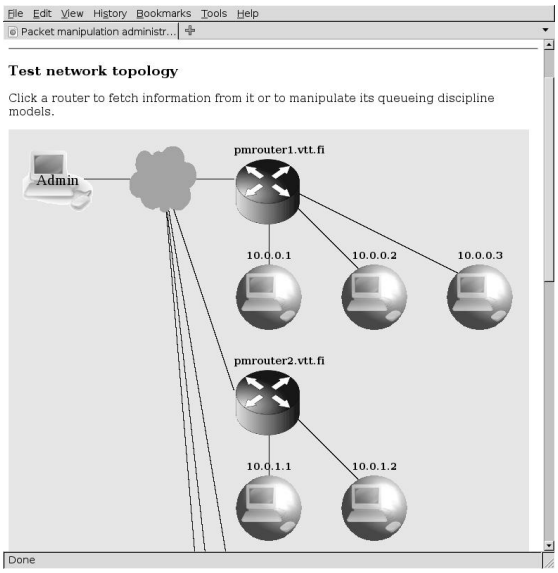

Fig. 2. The test network topology diagram on the start page of the GUI.

3.4.2 Implementation

The HTML-content of the GUI is generated dynamically by Python scripts running on the Apache web server module mod_python [16]. Python is also used for maintaining and manipulating the model data and communicating with the traffic manipulators through SSH-connections.

Ajax ("Asynchronous JavaScript+CSS+DOM+XMLHttpRequest", [17]) enables dynamic update of content on an HTML-page. The GUI Ajax functionality of exchanging XMLHttpRequest objects is based on Prototype JavaScript framework [18].

The network topology and qdisc model diagrams are drawn with JavaScript Diagram Builder library [19]. The data of one qdisc model resides in an instance of a Python class and it contains the parameters and inheritance info of qdiscs, classes and filters. The model diagram can be unambiguously drawn from the class contents, as illustrated in Fig 3. The data is updated according to user creation, change or delete of leaves of the tree diagram. Inheritance changes, which would lead to errors when submitting model to a router, are prevented by the class and return an error message to the user. The class generates the tc-commands to be previewed or to be sent to the router. The model data chosen to be saved by the user is permanently stored to a file in Python pickle format, usable in subsequent test sessions.

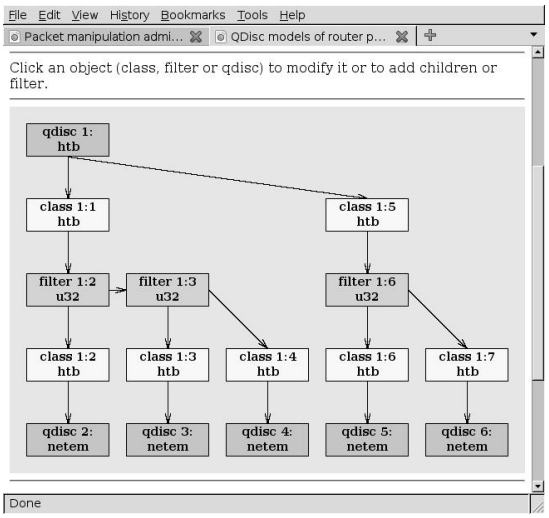

Fig. 3. The queuing discipline model diagram.

3.5 Keyword Driven Test Execution Framework

The Test Execution Framework has been developed to support both functional and non-functional system testing of P2P software. In such an environment, functional testing consists of API and protocol compliance testing, while non-functional testing includes performance and security testing.

3.5.1 Conceptual architecture

Fig. 4 presents the conceptual architecture of the test execution infrastructure. In the figure we have ignored physical nodes that do not directly take part in test execution process. It would be informative to compare this diagram with the actual physical topology as presented in Fig. 1.

Components that are important to the behaviour of the framework are: *Test Master*, *Test Agent, Test Peer* and a *Reporter*. The Test master is the single point of control for all the peers that are being managed by it. Test cases are managed and executed in test master. It also takes care of collecting logs and reports from agents to be stored in the file-system or database. To be able to manage peers from a remote manager, test master installs a thin agent (*Test Agent*) on each peer node. The test agent on each target peer mediates between the test master and the core peer-to-peer component on the peer node. An agent maps instruction received from the test master to APIs exposed by targets. *Test Peer*s are test script driven peers that are part of the testing ecosystem. These peer-nodes play different roles to interact with target peers depending on test scenarios. A *reporter* component is part of a P2P application that can be configured to report back logs or stats to the test master or any other monitoring node.

The test framework helps automate testing by automating individual steps in the testing process including installation, execution and report generation. Automated

installation allows test master to check and install necessary test agents and other required libraries and packages on different peer nodes in an automated fashion. With the help of test framework, test cases can be organized in test suites, which can be automatically executed in a controlled manner. Also the framework helps test designers and test case writers to share test data across multiple test suites. Logs and reports generated on peer nodes are collected from each node and presented to test master automatically.

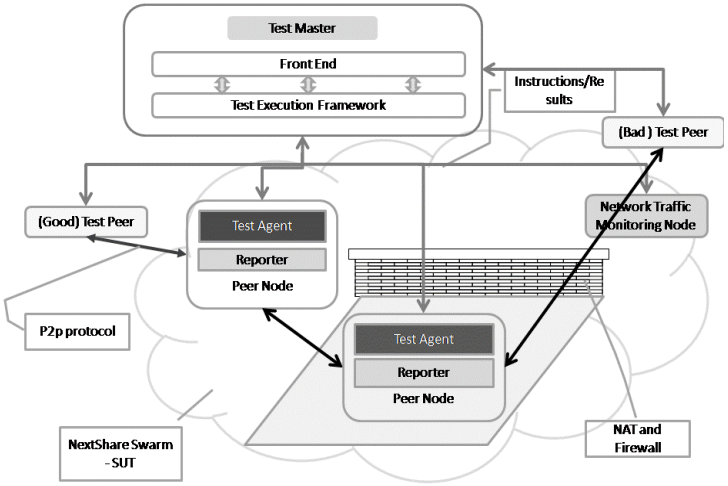

Fig. 4. Conceptual architecture of network with respect to test execution framework.

3.5.2 Keyword Driven Test Automation

With a keyword driven framework, test automation is made more useful and robust by creating a test language around the system to be tested. The keyword driven framework consists of two parts, a core engine and the libraries implementing the test language. The language is comprised of a vocabulary of keywords that can be used with a number of arguments [20]. The test language is independent and therefore creates an abstraction layer between the test description and low-level vocabulary. Also, test vocabulary can have a hierarchical organization where low-level keywords can be combined to create complex high level keywords. Fig 5 shows a simplified example of such a hierarchy. The core engine interprets and executes the test cases with the help of the keyword libraries.

The keyword driven approach has several advantages over the more traditional methods. For example, writing test cases is simplified from a two step process of defining test cases and translating them to a programming language, to a one step process of writing test cases in test language pseudocode. This method decouples test case writing process from test automation [21]. Once the vocabulary is defined, it is possible to start developing testcases in parallel to test automation process. As the SUT behaviour is encapsulated within the test language, the core engine could be

reused with other systems. With a well defined domain specific test language, test case developers are not required to have programming skills, while test automation engineers implementing the test language are not required to be system level subject matter experts. This system enhances reusability of test artifacts. The same set of test cases, data, reporting mechanism, comparison data and error handling can be used with different SUTs of the same type ensuring high reusability of the components as well as modularity [22]. Having a well defined vocabulary helps avoid inconsistency and enhances communication among different groups participating in system or product development.

The well defined division of ownership helps to improve the maintainability of automated test suites across configurations, versions and products. It is the responsibility of automation engineers to modify the language implementation when interfaces of SUT are changed while it is the responsibility of test case designers to add new vocabulary to the language and create corresponding test cases as new requirements are added to SUT requirements specification.

As P2P systems are being researched actively, multiple different implementations with varied interfaces are in development and require testing to compare functionalities and performance. With other types of frameworks a test case developer is obliged to understand each complex system and develop test cases for each system separately. In this case only the test language has to be ported by automation engineers.

For example some of the keywords we used in P2P media transport testing are: *start_player*, *stop_player*, *start_injector*, *stop_injector*, *start_swarm*, *stop_swarm*, *send_data_file_to_injector*, *collect_download_from_peer* etc. These are implemented with libraries that communicate with remote agents on peer nodes but the complexity of remote communication is completely hidden away from the testcase developers. So for instance to implement *start_swarm* test data is sent to the injector first, the injector is then instructed to start sharing content; metadata is collected from the injector which is then given to each peer, before it is instructed to join the swarm.

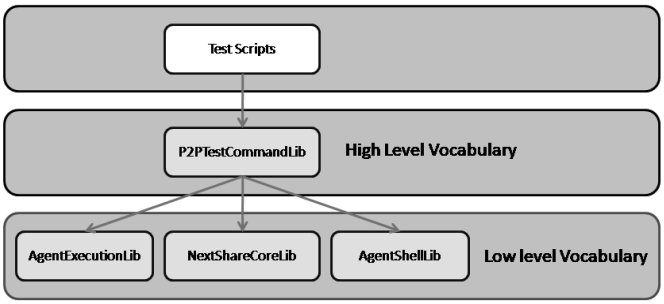

Fig. 5. Hierarchy of abstractions.

4. Experiments with the Testbed

We evaluated the usefulness of our testbed by deploying a P2P video delivery application in the test network and running test cases. The application used was Nextshare, which is a video streaming application being developed in P2P-Next project [3]. The application is based on transferring the video data with essentially unchanged BitTorrent.

4.1 Trial Configuration

Our test configuration included 19 peers. One of the peers was using a wireless (IEEE 802.11g) connection to the test network and the rest of the peers were running on wired Ethernet or virtual nodes. The main BitTorrent parameters are as follows: maximum number of concurrent uploads 7, maximum number of connections not limited, maximum upload rate not limited. The traffic manipulation capabilities of our testbed were used to simulate realistic access network configurations for the wired nodes. According to [23], the average speed of an internet connection in Europe is 6.56 Mbps in the downlink and 1.43 Mbps in the uplink. We assume that these values are optimistic estimates, since the people with fast connection are more likely to use the speed tests than the ones with slow connections. Thus, we selected to use three different asymmetric access network configurations in our tests as presented in Table 1.

Table 1 Access network configurations

	Pessimistic	**Conservative**	**Optimistic**
Downlink speed (kbps)	1000	2000	6000
Uplink speed (kbps)	500	1000	1500

For all access network configuration scenarios, we run three test cases with different access network configurations. The test scenario, which is used in all test cases, is presented in Table 2. In the scenario, wireless peer is launched next after the seeder peer that has the original data source (64 MB file). The time for preparing the seeder (T_s) may vary a bit between the test runs, but is around 30 seconds in all cases.

Table 2 Test scenario description.

Time (s)	**Event**
0	Seeder is launched
T_s	Seeder is ready and running
T_s + 10	Wireless peer is launched
T_s + 20	Wired peer #2 is launched
…	…
T_s + 180	Wired peer #18 is launched

4.2 The Experimental Results

While our testbed is able to perform detailed logging of all peers used in the test, we considered only the wireless peer in our evaluation. The statistics logged at the wireless peer were downlink data rate, uplink data rate, and the number of TCP connections to other Nextshare peers.

The results for the different access network configurations are presented in Fig. 6 for the pessimistic setup, in Fig. 7 for the conservative setup and in Fig. 8 for the optimistic setup. The effect of different access network configuration can be easily seen from the results. In the pessimistic case, the wireless peer does not get the file fully downloaded during the 1000 seconds measurement period. With the conservative settings, the download is finished after 775 seconds and with the optimistic settings after 655 seconds, as summarized in Table 3.

Table 3. Summary of numerical results

Testcase 1	**Pessimistic**	**Conservative**	**Optimistic**
Downlink speed (max/average kbps)	748/402	1440/800	1890/765
Uplink speed (max/average kbps)	7720/3080	12900/5930	17900/5740
Download complete time (s)	> 1000	775	655

The download speed is in all cases very limited because of the asymmetric limitations set to all the access network links of wired peers. In all cases the maximum download speed is still higher than the upload limit of the wired peers. This is possible because the download can happen from several peers simultaneously. The upload speed of the wireless node is significantly higher than download speed since that is not limited artificially and the peer is able to serve the other peers better than the wired peers. In addition, it can be noted that in the optimistic case, the capacity of the wireless link is the limiting factor for the uplink. In conservative and pessimistic cases, the wireless link is not fully used, but the bottleneck is in the wired peers' downlink limitation. The bottleneck could have been moved to the wireless link by increasing the maximum number of concurrent uploads of each peer (currently 7). This limitation also affects the rate of service that peers will receive: the one that arrives first can benefit from the upload slots of the ones that arrive later. On the other hand, the one arriving last will find that the upload slots of peers are already occupied, and it receives significantly lower data rates.

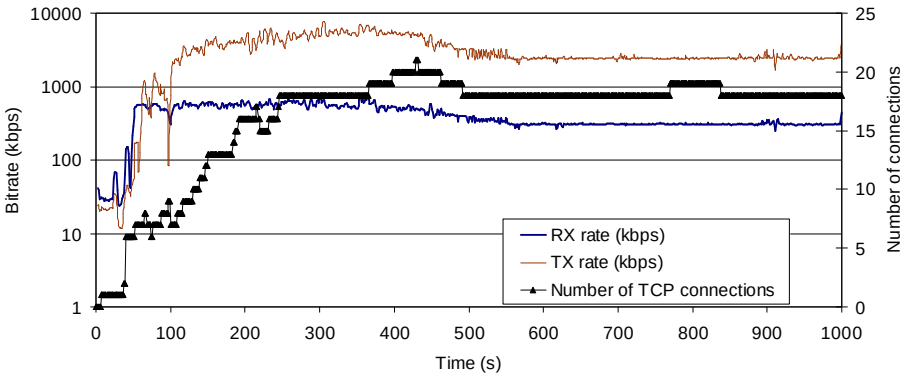

Fig. 6. Pessimistic setting.

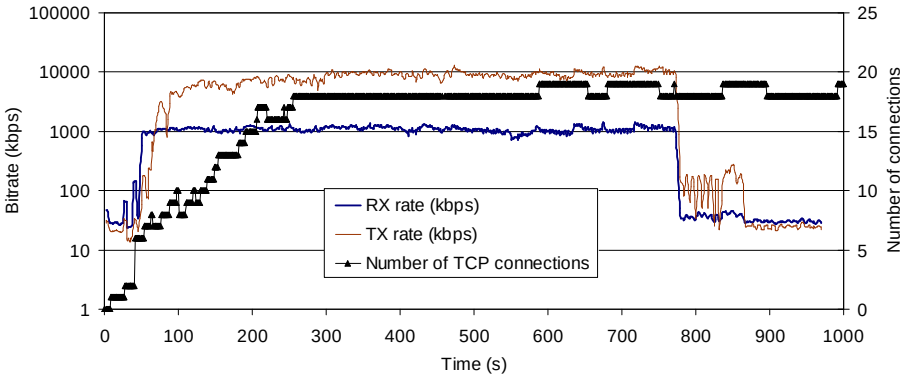

Fig. 7. Conservative setting.

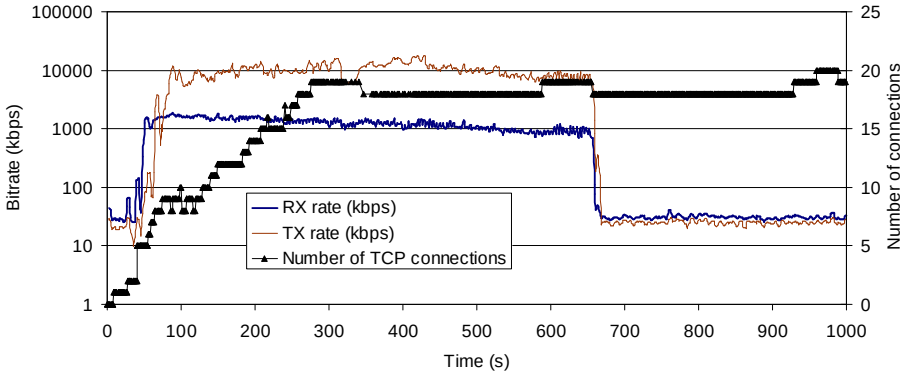

Fig. 8. Optimistic setting.

5. Related Work

Basically three different approaches to evaluate P2P-networks exist: analytical, simulation-based, and experimental. Experimental approach can further be divided into live network trials with real users or experiments conducted in a controlled environment.

The analytical approach is utilized in [24], where traces are collected from real networks and then used to build an analytical model in order to study inter-torrent collaboration. Often the analytical model is developed to enable simulations, as is with the fluid model in [25]. More recent simulations have been presented for example in [26], and in [27] an extensive study about the state of P2P simulations is provided.

The experimental approach in live networks is taken e.g. in [2], where a BitTorrent network is studied over several months. Many interesting findings are presented, but the scope of the deployments concerning real users differs significantly from the controlled environment, where more rapid testing is possible. Marciniak et al. [28], experiments in a real Internet environment provided by PlanetLab [29]. Number of nodes is around 40 and does not change during the test session (the measurements in this case did not require it). The most important advantage of this approach is, naturally, the real-world environment, which is too complex to wholly reproduce by emulation. On the other end of the scale, P2PLab [12] uses network emulation and heavy virtualization of the nodes allowing tests with thousands of nodes. The authors of P2PLab consider a PlanetLab-like environment as difficult to control and modify. Also the test results may be difficult to reproduce in it. The TorrentLab [26] includes both network simulation and tools for conducting live experiments. It does not, however, permit the configuration of the network's routers or to emulate network level phenomena. An automated BitTorrent testing framework is presented also in [30], where the download performances of different BitTorrent clients are compared. The framework lacks, however, the possibility to use network emulation, wireless access networks, and a keyword driven test framework. LKDT [22] is a generic keyword driven distributed framework on linux that addresses limitations of other types of frameworks but does not satisfy peculiarities of a P2P system testing framework with multiple types of remotely located components. Another large-scale testbed is called OneLab [31] that is built on the foundation of Planet lab, but connects to more cutting edge access networks. Thus it is suitable for Future internet research, such as P2P overlay networks, but to the best of our knowledge the P2P networks have not been studied in it so far.

The presented framework can be seen as a phase in development flow that is after simulations, but before the experiments involving real users. The benefits of this approach are fast feedback on new developed features (e.g. protocol enhancements), reproducibility of the results, possibility to adjust network level features, and automated testing.

The aforementioned experimental works leave a gap: they either lack full control over the nodes, wireless access networks, network emulation, or keyword driven test execution framework. Our work takes advantage of all these features and is thus, to the best of our knowledge, the most advanced fully controllable P2P testbed so far.

6. Conclusion

This paper presented a novel testbed for P2P application testing with full control over the network and the peer nodes. The testbed enabled wireless network testing, keyword driven test execution, easy access network configuration of the peers, and thus it is key enabler for testing complex scenarios that involve variable network conditions and rapid changes in the topology of the P2P network.

In order to demonstrate the capabilities of the testbed, we executed three test cases with different access network configurations. From the results it can be seen that network conditions of peers have direct effect to the client. We performed also other scenarios and it was found out that the first peer joining the swarm had the best download rate, whereas the last one suffered a lot from the limited amount (7) of upload slots of the peers.

The future work will consist of three main topics: the improvement of the testbed and its control functions, more complex trials with Nextshare platform including dynamic network configuration and more realistic arrival patterns for peers, and finally we aim to increase the amount of virtual peers to enable larger scale tests. These allow us to help improving Nextshare and enable research on BitTorrent enhancements.

Acknowledgments. This work was supported by EU FP7 P2P-Next project.

References

1. BitTorrent, http://www.bittorrent.org/.
2. J. A. Pouwelse, P. Garbacki, D. H. J. Epema and H. J. Sips, “The Bittorrent P2P File-Sharing System: Measurements and Analysis”, in proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS) (2005).
3. P2P-Next project web page, http://www.p2p-next.org
4. VTT Converging Networks Laboratory, http://www.cnl.fi/.
5. R. Canoni, P. Di Gennaro, V. Manetti and G. Venter, “Virtualization Techniques in Network Emulation Systems”, Lecture Notes in Computer Science, Volume 4854/2008, pp. 144-153, 2008.
6. Net:Netem - The Linux Foundation, http://www.linuxfoundation.org/en/ Net:Netem.
7. A. Keller, "Manual: tc Packet Filtering and netem," ETH Zurich, July 2006, http://tcn.hypert.net/tcmanual.pdf.
8. S. Hemminger “Network Emulation with NetEm”, *Proceedings of the 6th Australia's National Linux Conference (LCA2005), Canberra, Australia*, April 2005.
9. S. Jha, M. Hassan, "Engineering Internet QoS," Artech House, 2002.
10. NIST Net Home Page, http://snad.ncsl.nist.gov/nistnet/
11. Dummynet home page, http://info.iet.unipi.it/~luigi/dummynet/
12. L. Nussbaum, O. Richard, "Lightweight emulation to study peer-to-peer systems," Concurrency and Computation: Practice and Experience, volume 20, issue 6, pp 735 – 749, 2008.
13. L. Nussbaum, O. Richard. "A Comparative Study of Network Link Emulators," 12th Communications and Networking Simulation Symposium (CNS'09), http://www.loria.fr/~lnussbau/files/netemulators-cns09.pdf.

14. Net:Iproute2 - The Linux Foundation, http://www.linuxfoundation.org/en/Net:Iproute2
15. HTB Home, http://luxik.cdi.cz/~devik/qos/htb/.
16. Mod_python – Apache/Python Integration, http://www.modpython.org/.
17. J. J. Garrett, "Ajax: A New Approach to Web Applications", February 2005, http://www.adaptivepath.com/ideas/essays/archives/000385.php.
18. Prototype JavaScript framework: Easy Ajax and DOM manipulation for dynamic web applications, http://www.prototypejs.org/.
19. JavaScript Diagram Builder, http://www.lutanho.net/diagram/.
20. Mark Fewster, Dorothy Graham. "Software Test Automation". Addison-Wesley Professional.
21. Cem Kaner, James Bach, Bret Pettichord. "Lessons Learned in Software Testing: A Context Driven Approach". John Wiley & Sons.
22. Jie Hui and Lan Yuqing and Luo Pei and Guo Shuhang and Gao Jing. "LKDT: A Keyword-Driven Based Distributed Test Framework". International Conference on Computer Science and Software Engineering. Vol. 2, pp 719-722, 2008, http://doi.ieeecomputersociety.org/10.1109/CSSE.2008.1036.
23. World Speedtest.net Results, http://www.speedtest.net/global.php.
24. L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Dang, "Measurements, analysis, and modeling of BitTorrent-like systems", Proc. of the 5th ACM SIGCOMM conference on Internet Measurement, Berkeley , USA, 2005.
25. D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", SIGCOMM'04, Aug. 30–Sept. 3, 2004, Portland, Oregon, USA.
26. M. P. Barcellos, R. B. Mansilha and F. V. Brasileiro, "TorrentLab: investigating BitTorrent through simulation and live experiments", ISCC'08, 6-9 July 2008.
27. S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman and D. Chalmers, "The state of peer-to-peer simulators and simulations", ACM SIGCOMM Computer Communication Review, Volume 37 , Issue 2 (April 2007), Pages: 95 – 98.
28. P. Marciniak, N. Liogkas, A Legout, E. Kohler, "Small Is Not Always Beautiful," 7th Internation. Workshop on Peer-to-Peer Systems, February 2008, http://www.cs.toronto.edu/iptps2008/final/55.pdf.
29. PlanetLab, An open platform for developing, deploying, and accessing planetary-scale services, http://www.planet-lab.org/.
30. R. Deaconescu, R. Rughinis and N. Tapus, "A BitTorrent Performance Evaluation Framework", 5th International Conference on Networking and Services, 2009.
31. OneLab, Future Internet Testbeds, http://www.onelab.eu/.