TAMPERE UNIVERSITY OF TECHNOLOGY

MATTI TIAINEN

Network Environment for Testing Peer-to-Peer Streaming Applications

Master of Science Thesis

Examiners: Prof. Jarmo Harju
M.Sc. Jani Peltotalo
Examiners and topic approved in the
Faculty of Computing and Electrical
Engineering Council meeting on 9.12.2009

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Program in Information Technology
TIAINEN, MATTI: Network Environment for Testing Peer-to-Peer Streaming Applications
Master of Science Thesis, 104 pages.
January 2010
Major: Communications Networks and Protocols
Examiners: Prof. Jarmo Harju, M.Sc. Jani Peltotalo
Keywords: Peer-to-Peer, streaming, network emulation, virtualization, Linux traffic control

Peer-to-Peer (P2P) streaming applications are an emerging trend in content distribution. A reliable network environment was needed to test their capabilities and performance limits, which this thesis focused on. Furthermore, some experimental tests in the environment were performed with an application implemented in the Department of Communications Engineering (DCE) at Tampere University of Technology.

For practical reasons, the testing environment was assembled in a teaching laboratory at DCE premises. The environment was built using a centralized architecture, where a Linux emulation node, WANemulator, generates realistic packet losses, delays, and jitters to the network. After an extensive literature survey an extension to the Iproute2's Tc utility, NetEm, was chosen to be responsible of the network link emulation at the WANemulator. The peers are run inside VirtualBox images, which are used at the Linux computers to keep the laboratory still suitable for teaching purposes. In addition to the network emulation, Linux traffic controlling mechanisms were used both at the WANemulator and VirtualBox's virtual machines to limit the traffic rates of the peers. When used together, emulation and rate limitation resemble to the statistical behaviour of the Internet quite closely.

Virtualization overhead limited the maximum number of Virtual Machines (VMs) at each laboratory computer into two. Also, a peculiar feature in VirtualBox's bridge implementation reduced the network capabilities of the VMs. However, the bottleneck in the environment is the centralized architecture, where all of the traffic is routed through the WANemulator. The environment was tested reliable with the chosen streamed content and 160 peers, but by tuning the parameters in WANemulator bigger overlays might be achievable. In addition, a distributed emulation should be possible with the environment, but it was not tested.

The results from the experimental tests performed with the P2P streaming application proved the application to be functional in an environment that has mobile network conditions. The designed network environment is tested to work reliably, it enables reasonable scalability and provides better possibility to emulate the networking characteristics of the Internet, when compared to an ordinary local area network environment.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
TIAINEN, MATTI: Network Environment for Testing Peer-to-Peer Streaming Applications
Diplomityö, 104 sivua.
Tammikuu 2010
Pääaine: Tietoliikenneverkot ja -protokollat
Tarkastajat: Prof. Jarmo Harju, DI Jani Peltotalo
Avainsanat: vertaisverkot, suoratoisto, verkkoemulaatio, virtualisointi, Linuxin verkkoliikenteen valvonta

Vertaisverkkoihin perustuvat suoratoistosovellukset ovat kasvava sisällönjakelukanava. Suoratoistosovellusten ominaisuuksien ja suorituskyvyn testaamista varten tarvittiin luotettava verkkoympäristö, johon tämä diplomityö keskittyi. Tämän lisäksi ympäristöä käytettiin kokeellisiin testeihin sovelluksella, joka on kehitetty Tampereen teknillisen yliopiston Tietoliikennetekniikan laitoksella.

Käytännön syistä testiympäristö rakennettiin Tietoliikennetekniikan laitoksen tiloissa sijaitsevaan opetuslaboratorioon. Ympäristö on rakennettu käyttäen keskitettyä arkkitehtuuria, jossa Linux-pohjainen emulointilaite, WANemulator, luo verkkoon realistisen pakettihukan, verkkoviiveen ja viiveen vaihtelun. Kattavan kirjallisuuskatsauksen jälkeen Iproute2:n Tc-työkalun lisäosa, NetEm, valittiin toteuttamaan verkkoemulaatio WANemulatorissa. Jotta laboratoriota voitaisiin edelleen käyttää myös opetukseen, vertaisverkon jäseniä ajetaan VirtualBoxin levykuvien sisällä, joita taasen käytetään Linux-tietokoneilla. Verkkoemulaation lisäksi vertaisverkon jäsenien liikennemääriä rajoitettiin Linuxin verkkoliikenteen valvonnan keinoin sekä WANemulatorissa että VirtualBoxin virtuaalikoneissa. Verkkoemulaatio ja liikenteen rajoitukset yhdessä vastaavat melko hyvin Internetin tilastollista käyttäytymistä.

Virtualisoinnin mukanaan tuoma suorituskyvyn lasku rajoitti virtuaalikoneiden maksimimäärän jokaisella opetuslaboratorion tietokoneella kahteen. Tämän lisäksi VirtualBoxin siltatoteutuksen kummallinen ominaisuus huononsi virtuaalikoneiden verkkokapasiteettia. Varsinainen pullonkaula ympäristössä löytyy kuitenkin sen keskitetystä arkkitehtuurista, jossa kaikki liikenne reititetään WANemulatorin kautta. Valitulla suoratoistetetulla sisällöllä ja 160 vertaisverkon jäsenellä ympäristö testattiin luotettavaksi, joskin WANemulatorin asetuksia säätämällä suurempi vertaisverkko saattaisi olla saavutettavissa. Tätä ei kuitenkaan testattu.

Kokeellisten testien tulokset vertaisverkkoon perustuvalla suoratoistosovelluksella todistivat sovelluksen kykenevän toimimaan mobiilissa verkkoympäristössä. Työssä kehitetty verkkoympäristö mahdollistaa kohtuullisen hyvän skaalautuvuuden, se toimii luotettavasti ja vastaa paremmin Internetin olosuhteita kuin mitä tavanomaisella lähiverkkoympäristöllä voidaan saavuttaa.

FOREWORD

This Master of Science thesis was done in the Department of Communications Engineering (DCE) at the Tampere University of Technology (TUT). This work was supported by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

I want to express my appreciation for the supervisor of this thesis, Professor Jarmo Harju from TUT. He gave me the chance to participate in this interesting project and had the patience to let me finish the thesis properly. It has been a tremendous learning experience in the end of my studies, where I finally had the time and support to study the matters that were bypassed earlier in my studies.

I also want to express my gratitude to my co-workers at DCE, Lassi Väätämöinen and Jani Peltotalo, for the invaluable help and thoughts concerning this thesis. Also Tuure Vartiainen and Aleksi Suhonen have given me guidance along this long journey, for which they deserve my gratitude.

This thesis would not have finished without the support from my beloved girlfriend, who has understood the long nights in front of my laptop in those endless hours during the past six months. Thank you Anna-Kaisa for always being there for me.

I would also like to thank my parents, Terho and Seija, for giving me a solid background and life values to build upon. Neither do I forget my numerous fellow students, who I have met along my studies, you have given the content for my years at TUT.

On 23rd of December 2009, in Tampere, Finland.

Matti Tiainen
matti.tiainen@iki.fi

TABLE OF CONTENTS

List of Acronyms

ADSL	Asymmetric Digital Subscriber Line
AGPLv3	GNU Affero General Public License version 3
API	Application Programming Interface
bfifo	byte First In, First Out
BGP	Border Gateway Protocol
BIOS	Basic Input/Output System
BSD	Berkeley Software Distribution
CBQ	Class-based Queueing
CPU	Central Processing Unit
DCE	Department of Communications Engineering
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FIFO	First In, First Out
GNU	GNU's not Unix!
GPLv2	GNU General Public License version 2
GUI	Graphical User Interface
HSDPA	High-Speed Downlink Packet Access
HTB	Hierarchical Token Bucket
ICMP	Internet Control Message Protocol
IMQ	Intermediate Queueing Device

ISP	Internet Service Provider
LAN	Local Area Network
LGPL	GNU Lesser General Public License
LTE	Long Term Evolution
MAC	Media Access Control
NAT	Network Address Translation
NIC	Network Interface Card
OS	Operating System
P2P	Peer-to-Peer
PC	Personal Computer
pfifo	packet First In, First Out
PLC	PlanetLab Central
PRIO	Priority qdisc
qdisc	Queuing Discipline
RAM	Random-Access Memory
RED	Random Early Detection
RTP	Real-time Transport Protocol
RTP2P	Real-time Peer-to-Peer streaming system
RTSP	Real Time Streaming Protocol
RTT	Round Trip Time
Rx	Receive
SDS	Service Discovery Server

SFQ	Stochastic Fairness Queuing
SQL	Structured Query Language
TBF	Token Bucket Filter
Tc	Traffic Control
tfifo	time First In, First Out
TUT	Tampere University of Technology
Tx	Transmit
VDI	Virtual Desktop Image
VMDK	Virtual Machine Disk
VLAN	Virtual LAN
VMM	Virtual Machine Monitor
VNET	PlanetLab Virtualized Network Access
VoIP	Voice over Internet Protocol
VRDP	VirtualBox Remote Desktop Protocol
WAN	Wide Area Network

1 Introduction

Controlled network environment is vital for network research, as new ideas have to be tested before they can be released in production or published as scientific contributions. Test environments have been built for traditional client-server networks, but the recent advances in Peer-to-Peer (P2P) networking requires an environment suitable for P2P testing as well.

Especially interesting is the emerging concept of P2P streaming. P2P streaming differs from traditional P2P file sharing by requiring stringent real-time operation over the best-effort Internet. That and few other characteristics, are a challenge the P2P streaming software has to take into account. There are multiple operative solutions that offer P2P live or on-demand video services, but their weakest point is the long joining delay to the P2P overlay.

To address some critical issues for mobile environment, a Real-time Peer-to-Peer streaming system (RTP2P) for mobile networking environment [1] was built in the Department of Communications Engineering (DCE) at Tampere University of Technology (TUT). The purpose of this thesis is to build a reliable network environment suitable for a performance evaluation of the RTP2P. The focus is to observe how the RTP2P performs under realistic Wide Area Network (WAN) conditions.

Constraints exist on the selection of used methods. Due to patent pending issues the RTP2P software cannot be released outside TUT premises, the tests have to be performed on existing hardware in TUT, and the platform that the RTP2P runs on is Linux. Thus, the network environment will be built to the DCE teaching laboratory using open source software. The laboratory is in active use, which raised some other issues with configuration. Commercial software solutions are ruled out because of the tight budget of the project.

To build an environment acting like a wide-scale network, such as the Internet, is a challenge that has gained extensive effort in the research community. Complexity, scale, heterogeneity, and a fast growing pace of the Internet makes the task quite demanding. Different simulation and emulation tools exist for this purpose, so the most suitable ones were chosen to build the network environment for the RTP2P. With the help of virtualization the challenges of variable configurations in the teaching laboratory were overcome. After the environment is built, some experimental tests with the RTP2P were performed.

This thesis starts with a literature survey of the available network simulators and emulators. The most suitable ones among them are reviewed, after which an evaluation of the best option is made. Network link emulator NetEm and VirtualBox are used to build the network environment, which is explained next.

After the environment setup the RTP2P traffic characteristics are investigated and the reliability of the system is measured by various test runs. The results are presented, after which some experiments, with the RTP2P running in the environment, follows. At the end of the thesis, some discussion of the environment is given, and the conclusion sums up the work that was done.

2 TOOLS TO BUILD THE ENVIRONMENT

Scientific research procedure requires evaluation of the results and research methods. The tests also need to be reproducible and verifiable by other scientists. Three main methods used are analytical solutions, simulations, and live experiments [2].

Analytical solutions refer to mathematical models of the system, but they only work if the model is simple. Simulators can be used to address the faults in mathematical models, but they have their weak points as well. The preferable method, if possible, is to perform experiments with the actual system. [3] In the case of the RTP2P system this would require multiple mobile phones running on Linux operating system.

However, especially in Peer-to-Peer (P2P) environment, live testing with the actual system might prove to be difficult at very best. Even a rather simple P2P network consisting of a few hundred nodes raises problematic configuration, scaling, and resource issues. Also due to the nature of the Internet, predictableness of the results is an issue in real experiments. This leaves the field open for the fourth method, network emulation.

Network emulation is an intermediate solution between simulation and live experiments. In contrast to live experiments in a real system, emulation is also performed using the real application, but in emulated environment. [4] Compared with simulation, which is focused on abstract models, emulation focuses on reproduction of the real environment. This is done using another system to imitate the behavior of the real system.

The results received with emulation are more accurate than in simulation, and at the same time cost less and are more flexible than live testing. Flexibility comes from the possibility to change network conditions in order to see how the application performs in various conditions. Emulation fulfills the requirement of reproducibility, which live testing does not. Financial savings are acquired by emulating necessary hardware resources without the need for additional hardware purchases.

P2P environment itself introduces a number of challenges to the test methods. If live testing is used, after changing the parameters or a design of the test the results might not be reproducible due to the highly dynamic nature of peers in P2P environment. In emulation and simulation certain properties, such as peer dynamics, various connection types and link delays have to be properly modeled to obtain accurate results. [5] Also, the large scale of real-world P2P networks has to be taken into account [6].

To sum up, due to lack of resources only network simulation and emulation are considered in this thesis. The rest of this chapter is dedicated to network simulators and emulators, some of which are reviewed. At the end of the chapter an evaluation of the best possible tool for building the network environment for the RTP2P is done.

2.1 Simulators

Simulating network behavior on Internet scale is difficult. Tiny details that are usually abstracted away, might influence the results significantly. Therefore, simulations should only be used to prove the usefulness of some method, not to produce quantitative results over different solutions. [7]

According to Naicken et al. [8] simulation is the most commonly used method for P2P application research and overlay network analysis. They find simulation less expensive and less resource demanding than large-scale experiments, and more accurate than mathematical models, if properly constructed. Simulation in general has scalability unreachable with other methods.

However, the research in [3] and [8] proved most of the available simulators defective in one way or another. Many simulators reviewed lack the feature of adequate data gathering, most had sparse documentation, and usability was an issue as well. Even the scalability was not sufficient in some of the simulators. [8] In addition to these, most of the existing simulators are too low-level and make the transition from simulation code to experimental code quite difficult. [9]

Possibly for these reasons, most often the simulators are customly built for the research in hand. NS-2 [10] was the next common option, probably due to its familiarity to the researchers beforehand. Other interesting simulators stated in [3] and [8] included PeerSim [11] and PlanetSim [12]. However, surprisingly few researchers decided to use simulators in their studies, actually less than half of the research papers Naicken et al. reviewed. [8]

P2P researchers usually focus on higher layer overlay algorithm verification using TCP or UDP as a transport protocol instead of simulating the whole TCP/IP stack. [9] The reason for this, is that with too detailed network layer simulation scalability might degrade drastically, depending on the simulator architecture. That is why NS-2 is hardly suitable for large scale simulation of P2P systems. [3] Yet, abstraction of lower levels causes inaccuracy in the results. For this reason, simulation is always a trade-off between accuracy of the test and hardware resources. A short review of two most interesting simulators follows.

2.1.1 PeerSim

PeerSim [11] is an event-based Peer-to-Peer simulator implemented with Java programming language. It is built with scalability and dynamicity in mind by abstracting lower layer communications away, which enables simulation with millions of nodes. It focuses on overlay simulations, and both structured and unstructured overlays are supported. PeerSim is extendable and pluggable so that new extensions should be easy to implement. ASCII file with key-value pairs is used for flexible and easy configuration. Statistical data gathering is also available. [3]

Two simulation models, cycle-based and event-based, are supported. Both models have their own simulation engines. The difference between the models is in node invocation. In cycle-based model, nodes are invoked at the turn of each cycle, while in event-based model a certain event schedule is used. The latter allows concurrency and is thus more realistic but less efficient. [3] Another difference worth mentioning is in abstraction. Simplified cycle-based engine ignores the details of transport layer communication, while event-based model simulates that as well. [3]

The simulation is performed by the collection of nodes, that can hold one or more protocols. Nodes communicate by object method calls, which does not simulate a real network that well. The tested overlay algorithm might also need to be modified to comply with PeerSim's interface. In addition to that, most of the component libraries are mainly designed for overlay protocols forming a graph topology, which complicates implementations of other protocols. These and a couple of other issues might require additional work for specific overlay algorithms. [3]

PeerSim is published under GNU Lesser General Public License (LGPL) and is freely downloadable from PeerSim website at [11], which also offers some tutorials focused mostly on cycle-based model. Extensive Application Programming Interface (API) documentation is also available at the website. Unfortunately, the website lacks comprehensive guides for more accurate event-based model. [3] Third party overlay implementations and extensions to PeerSim are available from [11] as well.

In spite of the issues mentioned above, PeerSim has successfully been used in plenty of research papers and thus, might be worth a more closer look. However, Pujoll-Ahull et al. [13] found PeerSim documentation too sparse and the simulator itself too difficult to extend in a clear way. They also state that PeerSim is not really that usable in common simulation scenarios as its design is too static. Hence, they built another simulator called PlanetSim, which is introduced next.

2.1.2 PlanetSim

PlanetSim [9] is also a discrete event-based framework written in Java which is focused on overlay network and services simulation. The aim of the framework is to offer a general-purpose simulation tool for researchers, that is extensible, customizable, efficient, and modular. PlanetSim does not consider packet-level details but focuses on higher layer overlays. [13]

PlanetSim supports both structured and unstructured overlays. [3] It offers two options for implementing new overlay protocols, traditional algorithm-based and novel behavior-based method. The difference between them is in stabilization tasks the node performs after it has joined on the overlay. The latter is more advanced on many factors. [13]

PlanetSim aims to be easy to use as simulation is mainly specified in plain text-based configuration files. Java language makes it easy to learn and extend, but produces a performance penalty. The effect is reduced by careful code optimization and concentration only on higher level simulation. [13]

Although Naicken et al. [3] state that there is no statistical data gathering in PlanetSim, only network topology graphs are available, Pujol-Ahull et al. [13] argument otherwise. That might be a new feature implemented after P2P simulator survey [3] was held, but this was not investigated further for this thesis.

There exist a lot of different applications and services, that are built on top of overlays, so PlanetSim uses Common API [14] to enable the independence from the underlying overlay. Common API allows using the same application on top of different P2P algorithms. Extensions to PlanetSim are easy to implement, since it is using layered design, well-known pattern design, and clear extension points. The authors of PlanetSim have proved this by implementing Chord and Symphony overlays, but also third-party extensions exist. [13]

PlanetSim aims for code re-use from simulation code to experimental code. Network wrapper module takes care of network communication and allows interoperability with network testbeds such as PlanetLab. [9] PlanetSim is published under the LGPL license, it is freely downloadable, it seems to have active community, and it is constantly evolving. For these reasons, PlanetSim appears to be the recommended network simulator at the time of writing.

2.2 Emulators

Another, and most often more suitable option, to test application performance is network emulation. The greatest benefit of emulation over other testing methods is its flexibility. It is a cost-efficient solution, as network characteristics of an emulation layer can be varied without the need to change the network infrastructure [4]. The most common emulated network characteristics are network delay, packet loss, duplication, corruption, and reordering as well as available bandwidth.

At technical level, network emulation is a time critical task, that should be handled by kernel-programming to guarantee real-time actions [15]. Access to high resolution timers must also be available. This is a challenge in Linux environment, which is not a real-time system. Linux 2.6 kernel offers tick rate of 1000 Hz at best, while in 2.4 kernel the maximum value is only 100 Hz. With FreeBSD the maximum tick rate is higher, reaching to 10 kHz [4]. The tick rate affects accuracy of emulation and sets up limits to emulation values. For example, with 1000 Hz tick rate delays less than 1 ms and limitation of 10 Gbps network to 100 Mbps are not possible. [16]

Table 2.1: Network emulator categories.

Category	**Emulators**
Virtual network emulators	ModelNet, EmuLab
Experimental testbeds	PlanetLab
Network link emulators	Dummynet, NIST Net, WANem, NetEm

There are several network emulators available and they can be divided into three categories seen in Table 2.1. The first category consists of *virtual network emulators*. Their goal is to emulate the whole network cloud. Network topology description is supplied to the emulator, which uses a cluster of computers to emulate the network. Emulators in this category include ModelNet [17] and Emulab [18]. Virtual network emulators are quite complex on design and their setup is complicated. [4]

Second category contains *experimental testbeds* such as PlanetLab [19]. PlanetLab is geographically distributed testbed which emulates the real networking conditions in the Internet. As the conditions follow the behavior of the Internet, the network characteristics of PlanetLab cannot be controlled or modified. This might lead to a situation where application is only executed and the results are reviewed, when emulation in general enables deeper understanding of how the application works. [4] In a strict sense, PlanetLab should be counted as a live experiment, not as an emulator.

Third category consist of *network link emulators*. They are simpler than emulators in the first category as they only delay or drop packets coming in or going out from a specified interface. [4] Example network link emulators are Dummynet [20], NIST Net [21], WANem [22] and NetEm [23]. On the downside, network link emulators tend to require extensive manual configuration [24].

Depending on the source, the categorization of emulators varies. Other emulators than the ones mentioned above exist, but the ones mentioned seemed the most suitable for this thesis. These emulators are reviewed in the following sections.

2.2.1 ModelNet

ModelNet [25] is a large-scale software emulator used to evaluate distributed services. It allows researchers to deploy their software prototypes on a configurable, Internet-like environment. [26] The ModelNet core is based on Dummynet [20] with extensions to improve accuracy as well as support multi-hop and multi-core emulation. [25]

As it is impossible to capture the full complexity of the Internet, ModelNet uses three main methods to trade accuracy for scalability: progressive reduction of the emulated topology complexity, multiple virtual edge nodes run on one physical machine, and synthetic background traffic to dynamically change network characteristics. [25]

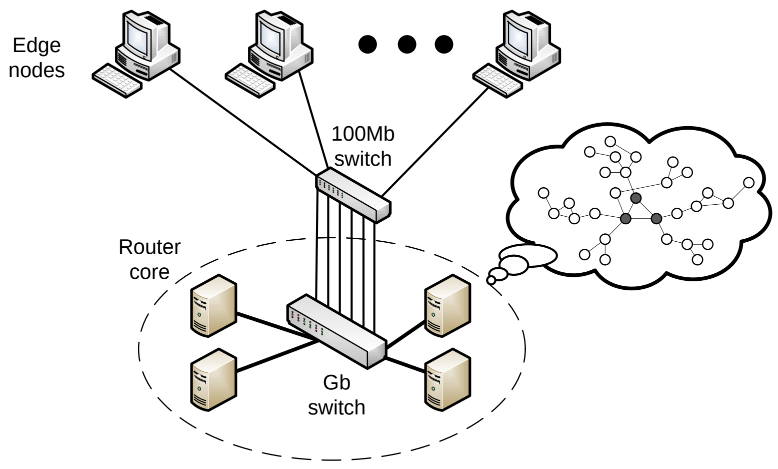


Figure 2.1: ModelNet architecture [25].

ModelNet architecture can be seen in Figure 2.1. It runs on a scalable Gigabit LAN cluster. Edge nodes run on user specified operating systems and are configured to route their packets through ModelNet core nodes, which emulate the Internet topology. Multiple core nodes on separate machines can be used to increase emulation bandwidth. The core nodes interoperate to emulate the wanted network behavior. Core nodes are implemented as loadable kernel modules on a FreeBSD machine and one core can accurately emulate 85 000 pkt/s. Target topologies are formed either from Internet traces, BGP dumps or by synthetic topology generators. [26]

The working principle of ModelNet is based on end-to-end application behavior, where a network topology is simplified to a network of pipes using a simple shortest path routing protocol. The pipes can be configured to use different queuing disciplines and specified bandwidth, latency and loss rates. With multiple core nodes, different characteristics can be applied by routing packets through separate cores. Some issues in scalability still exists, but greater challenge is the lack of resource isolation mechanisms in competing virtual nodes run on the edges. One virtual peer might use all the bandwidth available at the edge node with UDP traffic, so that the other virtual peers have no bandwidth at all. However, this could be prevented by using a separate traffic shaper on edge hosts. [25]

The largest emulation performed with ModelNet consisted of 1120 virtual nodes on four core nodes [27]. ModelNet has emulated topologies consisting of more than 10 000 links, although running traceroute or user-configurable dynamic routing is not possible due to ModelNet's architecture [28]. Static routing tables used by original ModelNet are not optimal as their sizes are in the order of $O(n^2)$. In [29] multiple spanning trees and negative caches are used to get rid of static tables to increase scalability. Using this enhanced technique ModelNet scales to tens of thousands of hosts. [29]

ModelNet is published under the GPL license, although some files use the BSD-style license. Only beta and alpha versions are available from the project web site [17] and guides on installing and using ModelNet are quite scarce. Some additional instructions can be found at [30] and [31], but the bigger issue is the age of ModelNet components.

The last contributions to the project seem to have taken place in 2005. Users have contributed some patches to make ModelNet work on newer operating systems, but those patches are also from a few years back. In addition to this, some of the required auxiliary applications are difficult to find and dependencies on ModelNet itself are quite tricky to solve.

2.2.2 Emulab

Emulab [32], formerly known as Netbed, is a distributed network testbed which offers a reproducible networking environment [33]. It combines simulation, emulation and live network experiments under a common framework. Emulab's time- and space-shared platform offers two kinds of nodes for researchers, emulated nodes on local clusters and geographically distributed wide area nodes. Access to these resources is gained by the help of virtualization. In contrast to traditional virtual machines, that aim to the virtualization of the architecture's instruction set, Emulab abstracts the whole network. [24]

The virtualization of resources guarantees Emulab's extensibility. Resources can be controlled and allocated regardless of their physical realization. The first local area implementation of Emulab was located in the University of Utah and the University of Kentucky. The sites were connected via Virtual Local Area Networks (VLANs) to provide security and performance isolation. [24] Currently, 374 test nodes are available at the main site in Utah, and more can be found at 24 sites around the world offering variable amounts of nodes [18]. Locations of these global sites can be seen in Figure 2.2.

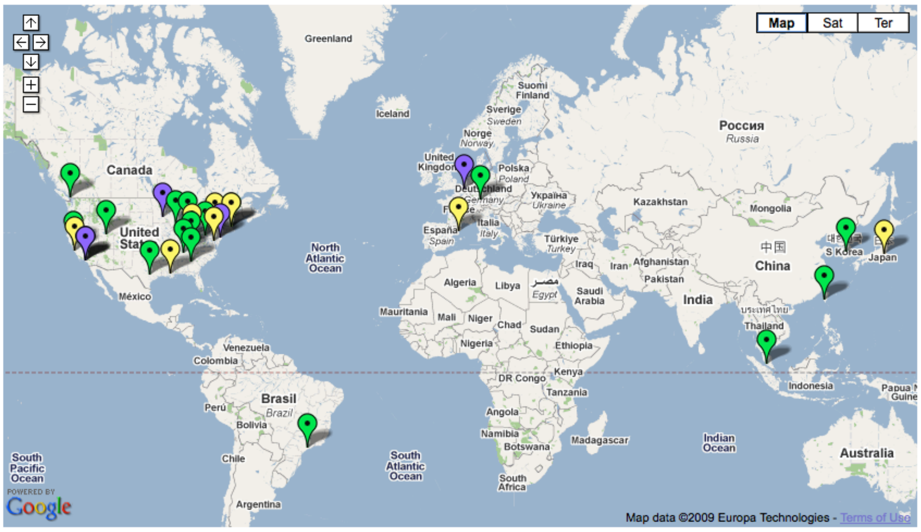


Figure 2.2: Emulab sites [18].

Simulation in Emulab is achieved by using the emulation facility of Ns simulator, Nse. Nse allows simulated nodes, links, and traffic to interact with real application traffic. Emulab uses Dummynet in FreeBSD or Traffic Control (Tc) in Linux [4], with VLANs to emulate wide area links in LAN environment. Distributed resources are achieved by nodes provided by other organizations around the world. These nodes run instances of Emulab software to offer Internet behavior emulation to the researchers. [24]

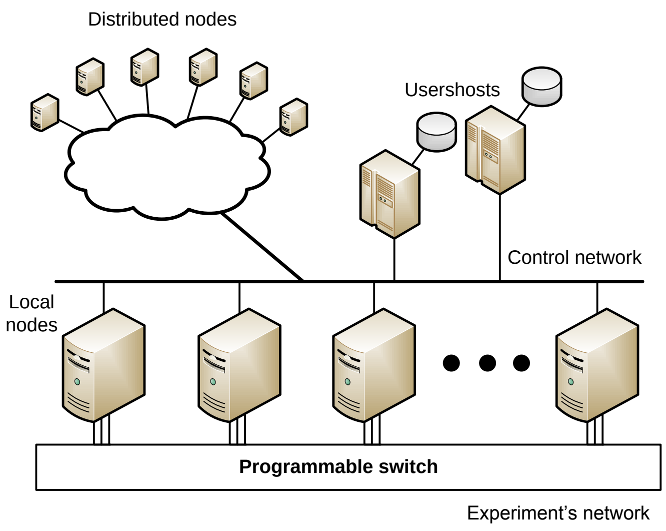


Figure 2.3: Emulab architecture [24, 33].

Figure 2.3 presents Emulab's architecture. User-defined topology is generated using a programmable switch, that also forwards traffic through nodes dedicated to emulation. Any node, whether local or distributed, can function either in edge node, traffic generator, or router mode [24], and the other nodes are available for running user experiments. [33] On local nodes Nse can be multiplexed as well to enable simulation. Usershost is a server, that controls the testbed and user privileges via control network. Distributed nodes are connected to Emulab testbed via control network as well.

Emulab can be used for research by using either already existing nodes located in Utah and around the world, or by building a local test network. Easier method is to obtain an account after certain prerequisites are fulfilled and just start experimenting by using existing hardware. There are no requirements to have any hardware of your own. Another option is to download Affero GPLv3 (AGPLv3) licensed Emulab software and build a local experimental testbed of your own. This method is far more demanding than just applying for an account, as it requires high level of proficiency and specific hardware to be used. However, the extensive documentation of the steps required to build your own Emulab instance can be found at the Emulab website. [18]

Emulab experiments are not tied to certain Operating Systems (OSs), as Linux, FreeBSD and Windows platforms are available. Even an option to load your own OS image to Emulab servers is possible. Most of the testbed configurations are done via web-interface, but some parts require logging onto a FreeBSD server to perform command line setup. The experiments in Emulab are configured using either Ns scripts or Java GUI. [18]

Recent upgrades in Emulab have improved the virtualization by enabling multiplexed hosts on a single physical machine so that high performance, high fidelity, and near to total transparency to applications is retained. Original Emulab was bound to maximum of 100 node experimentations, but the above mentioned improvements allow at least two thousand nodes to be reliably instantiated. Out of platform Virtual Machine Monitors (VMMs), Xen [34] is partially supported in Emulab and VMware [35] is under consideration. [28]

Emulab seems to be an efficient and versatile option to automate a test network setup. Emulab has been around since the start of the decade, it is constantly developed and support is available, although mostly for FreeBSD [28]. Wide usage of the system in research groups around the world implies a stable testbed environment. In addition to all these, access to PlanetLab slices is possible via Emulab. Details on this can be found at Emulab webpage [18], which offers the best and most up-to-date information on the system.

2.2.3 PlanetLab

PlanetLab [36] is an overlay-based testbed distributed geographically around the world, where computational power resides on the edges [37]. PlanetLab aims to provide a realistic snapshot of the real Internet. Using PlanetLab, institutions can gain access to a large and growing platform with a modest resource contribution. The size of the testbed with heterogeneity of locations and systems, offers network dynamics which are hard to achieve in local testbeds. [33] At the time of writing PlanetLab consists of 1042 nodes on 488 sites [19], which are presented in Figure 2.4.

PlanetLab grew from the need for an overlay-based testbed running on a new service-oriented network architecture. The overlay approach enabled designing disruptive technologies without the need for chancing the underlying Internet [37]. PlanetLab approach is different from traditional testbeds, as it is suitable for both the scientists that want to

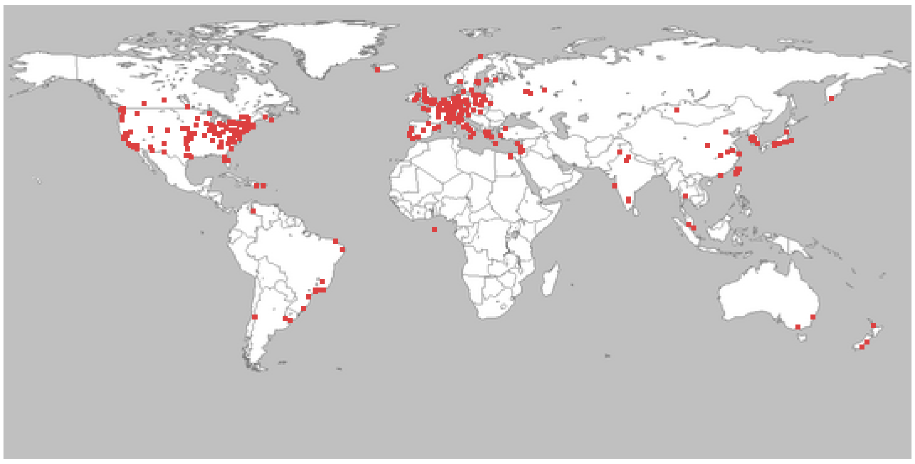


Figure 2.4: PlanetLab sites [19].

develop new services, and clients who want to use them. [36]

Four design principles exists. In PlanetLab, services are supposed to run continuously, control over resources is distributed, overlay management is unbundled, and APIs are designed for application development in mind. The distributed nodes are divided into slices, which are used by separate researchers. This enables shared and simultaneous use of the PlanetLab resources. [36]

The overlay in PlanetLab serves two purposes: applications running on PlanetLab are structured as overlays, but simultaneously the overlay is used for controlling and managing the testbed itself. [36]

PlanetLab is designed along three main dimensions. Firstly, the overlay network is supposed to be large, in the order of 1000 sites. This allows the wide deployment of services and measurement tools. Secondly, overlay consists of two main software components. Virtual Machine Monitor (VMM) runs on each node and specifies a controlled interface against which services running on PlanetLab are written. Management service is used to control the overlay and testbed in general. The third dimension is the overlay's mode of operation. In the development of the platform a long-term view is taken, where the overlay is concurrently a research testbed and a deployment platform. [36]

The PlanetLab platform consists of two types of components as seen in Figure 2.5. PlanetLab Central (PLC) is the core of the system dedicated to controlling the nodes. On com-

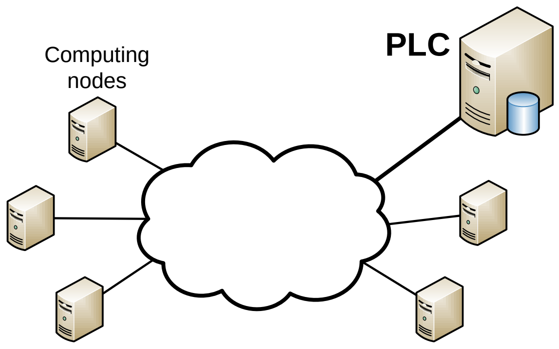


Figure 2.5: PlanetLab architecture [33].

puting nodes, in other words the end nodes, the users can run their experiments. The nodes are spread among participating sites. PLC acts as a server for the nodes, where nodes fetch their initial software, additional packages, and software updates. A database containing all relevant information on the nodes and users of the system, also exists on PLC. The nodes are running a custom version of Linux with a set of management programs.

Users run their experiments on a slice, that is a virtualized instance of the node resources. Linux-Vservers [38] are used to gain file system isolation, while VNET [39] performs the traffic isolation between slices. In addition, the Tc traffic shaper is used to guarantee fair bandwidth usage between users. [33] More detailed explanation of PlanetLab architecture can be found at [40].

Only the PlanetLab authority has root-access to the nodes to ensure security, normal users use slice architecture described above [36]. There has been proposals to replace the Vserver-architecture with Xen [34] and the long term goal of PlanetLab from the start has been a fully decentralized approach, where PLC is no longer required [37].

In order for an institution to gain access to PlanetLab resources, they have to sign the consortium membership agreement and contribute hardware and networking resources for the project. This hardware is dedicated to the testbed, and it must fulfill certain requirements. Joining the PlanetLab Consortium is free of charge for academic institutions and non-profit organizations. Industrial partners pay a yearly fee depending on their membership category. Individuals cannot join the consortium, they have to gain access via an institution. [19] However, these third-party users can still use services built on top of PlanetLab.

Currently, PlanetLab is not an optimal emulation tool. It offers no control over the network conditions such as bandwidth, latency, or packet loss, as they are "real" depending on the conditions of the Internet at the moment [36]. This leads to a situation where only a single configuration is possible, so the results lack generalization [4]. Slices are isolated from each other, but they still might affect the results as one of them could gain the control of a node [41]. There is no reproducibility in traditional sense, only a snapshot of current conditions is given [33].

Even PlanetLab correspondence to real network conditions is controversial, as PlanetLab nodes tend to have much more bandwidth than the typical end user does [37]. This has lead to efforts to offer emulation on PlanetLab nodes. For example, [33] is an interesting project implementing Dummynet on PlanetLab nodes.

Although PlanetLab's main contribution is to build a distributed wide-scale testbed, it is possible to build a local private PlanetLab into a laboratory. It can be used, for instance, on learning the PlanetLab architecture or developing new features for it without the need to gain access to the Consortium [42]. This MyPLC software is downloadable from PlanetLab website at [19]. For local deployment it is recommended to have CentOS5 computer, which is turned into a Vserver capable host [43].

To sum up, PlanetLab has never been just a testbed for emulating network conditions. It has higher goals to grow into an overlay-based planetary-scale service deployment platform, just as the Internet itself grew on top of the telephone system [36]. PlanetLab is best used to validate the results gained in simulation. The greatest benefit PlanetLab offers is at qualitative experience in learning from the real world. The PlanetLab platform continues to grow, and only time will tell will it one day become a successor to the Internet altogether. [37]

2.2.4 Dummynet

Dummynet [44, 45] is a tool designed for testing networking protocols. The main platform is FreeBSD, but ports to Mac OS X and Linux exist. Emulation with Dummynet can be performed either on the user's workstation or in machines acting as bridges or routers. [20] Dummynet was the first major network link emulator in a wide scale use.

Dummynet consists of two modules, packet classifier ipfw and emulation component dummynet. In addition, a front-end component /sbin/ipfw is used to configure the emulator. [44] Ipfw is used to intercept packets in their way through the protocol stack. The packets are then passed through queues or pipes, which perform the actual emulation. A Pipe is fixed-bandwidth channel and a queue is a chain of packets associated with a weight. Queues are connected to pipes, whose bandwidth they share in proportion of their weight. [20]

Only traffic on adjacent protocol layers is modified, so Dummynet configured to run between TCP and IP layers leaves UDP traffic untouched. Filtering rules are available to affect only selected traffic, for instance based on destination port. [45] With multiple rule sets also UDP or other type of traffic can be affected as well. Using proper ipfw configuration, traffic can be injected through the pipes multiple times, which emulates complex topologies rather well. [44] The same instance of Dummynet can be shared by multiple users of a system without interruptions to each other. Queue and pipe configuration, as well as ipfw rules, can be modified in runtime. [33]

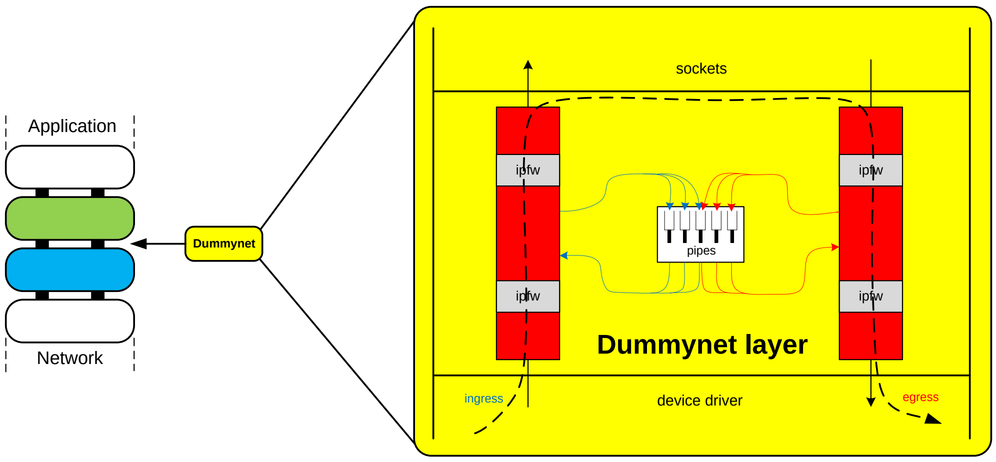


Figure 2.6: Dummynet emulation principle [33].

Just as NIST Net, Dummynet does all filtering and queuing itself. [16] In contrast to NIST Net and NetEm, Dummynet hooks up to both upstream and downstream traffic, as seen in Figure 2.6. [33] In the figure, traffic flow through the network stack is presented with a dash line, ingress packet capture with a solid blue line, and egress packet capture with a solid red line. Ipfw classifies the packets into pipes, which in turn perform the emulation. After this the packet is returned to the network stack, where either more emulation rounds are done or the packet may continue its journey.

Dummynet is only capable of modeling constant packet delay, and only a constant bandwidth limitation on a single link is possible. Varying delays using user space scripts might be feasible, but it would violate the real-time support. [15] Dummynet is scalable, as there are systems running Dummynet with thousands of active pipes and queues with a total throughput of over 100 000 packets per second. [44]

Dummynet has been significantly extended since its original design. Today it offers various queue management schemes, per-flow emulated links and emulation of complex topologies are possible, even trace-driven dropping is implemented. Dummynet has also been used as a traffic shaper, even though it was not designed for it. Dummynet is a core component of Emulab and ModelNet and it has been planned to use it in PlanetLab nodes [33] as well. [44]

Dummynet, published on the BSD license, is part of FreeBSD and Mac OS X kernels. And thus, just loading necessary kernel modules is enough for its usage in those platforms. In June 2009 a port to Linux was made, and the source code for it can be found at the Dummynet web site. The required modifications for the original Dummynet to build the port are explained in [33].The port still has some instabilities due to its novel nature. [20]

Dummynet is a standard solution for emulation in FreeBSD world. As it is part of the mainstream kernel, support and continuity of the emulator is guaranteed. Recent activities with PlanetLab might make Dummynet worth a closer look. For example, a comparison of the Linux port with other Linux based network link emulators might be interesting.

2.2.5 NIST Net

NIST Net [46], "network-in-a-box", is a specialized Linux kernel 2.4 router. The working principle of NIST Net can be seen in Figure 2.7. It emulates the whole network in a single hop by selectively applying user defined network effects on the traffic passing through the router [46].

NIST Net enables common network effects, that are emulated through three-tier emulator entries. Matching rule, required effects, and wanted statistics of the packets can all be configured. Matching rules cover most of the fields in IP and higher level protocol headers.

A fairly wide set of network effects is available with different options, covering the most common networking scenarios. Thousands of simultaneous entries can be used and each packet flow is treated separately. [46]

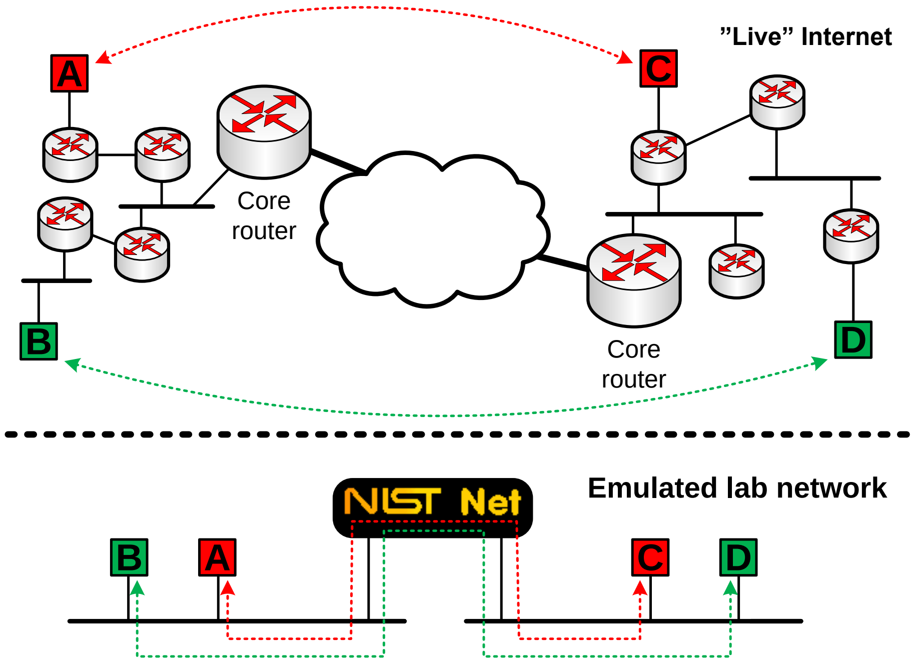


Figure 2.7: NIST Net as a "network-in-a-box" [46].

NIST Net architecture consists of two main parts, a loadable kernel module and a set of user interfaces. The module enables runtime changes in the emulator state without interrupting active connections and separates NIST Net code from the base kernel to a large degree. By using hooks to the Linux kernel all ingress packets are passed to the kernel module. After the emulation effects are applied, the traffic is returned to the Linux network stack. External hooks to NIST Net module allow external code to be used in conjunction with the emulator as well. Separation of kernel and user interfaces in the emulator itself allows multiple processes to control the emulator concurrently. [46]

NIST Net uses one of the alternative time sources available in PC architecture to provide a high speed clock of 8192 MHz. It makes the emulation more accurate, but also has a negative effect on performance because of the high interrupt load. It also makes NIST Net port on other architectures impossible, NIST Net is for x86 architectures only. [16]

NIST Net has two user interfaces, a simple command line and an interactive graphical one. [46] The command line can be used for scripting, where the X Window interface is more suitable for monitoring specific traffic streams passing through the router. In addition to these interfaces, captured traffic traces can be used as a basis of the emulated network conditions. [21]

RplTrc [15] claims to be a separate emulator from NIST Net, but it re-uses some of NIST Net's code. RplTrc differs from NIST Net by using only unlimited length traces of real network traffic as input. RplTrc takes into account the performance characteristics of real networks, which NIST Net does not emulate, such as self-similarity of cross traffic and long range dependence. [15] In [15] NIST Net's table based delay generation is claimed to be inaccurate. This issue is fixed by using real network traces in RplTrc.

RplTrc intercepts the traffic of a Linux TCP/IP stack between the Layers 2 and 3. Then a kernel module registered as a Layer 3 packet handler is used to perform the time critical task of packet delaying. The trace reader user space module is used to load packet action traces to the kernel that drives the emulation. [15]

In [46] NIST Net was found applicable up to Gigabit line rates even with packet rates exceeding 1 million pkt/s. RplTrc, on the other hand, has been operating at maximum of 100 Mbps [15]. NIST Net was originally developed on kernel 2.4, but some effort has been made to port it on 2.6 as well. Even though NIST Net kernel module should be quite independent, the notable change to 2.6.14 Linux kernel headers broke the official NIST Net build. A patch for 2.6.22 kernel can be found at [47].

NIST Net project is public domain, and can be downloaded at [21]. The emulation tool is widely used in several research papers and laboratory setups. The project page activity is focused on the first half of the decade and thus, is quite aged. A newer project, called NIST Net, Next Generation can be found at SourceForge [48], but the last activity there is from a couple of years back. RplTrc can be obtained at [49], but it seems to have been discontinued as well.

2.2.6 WANem

Wide Area Network Emulator (WANem) [22] is used for emulating the Internet over LAN. It is built especially with application development and testing on mind, and it enables the emulation of the most common network characteristics. WANem is released under the GPLv2 license and is built on top of free and open-source software. [22]

As with NIST Net, WANem can also be seen as "network-in-a-box", but it is built with simplicity of use in mind. The current 2.2 version is delivered only on a re-mastered Knoppix kernel 2.6.24 Live CD. Image for VMware Player and Server is also available. Emulation rules are generated using simple web-interface, either in basic or advanced mode, but access to a more versatile command line interface is also available. In addition to these, WANanalyzer module in WANem offers some characteristics of current network status between the emulator and remote machines to be collected. WANem works only on i386 architecture. [22]

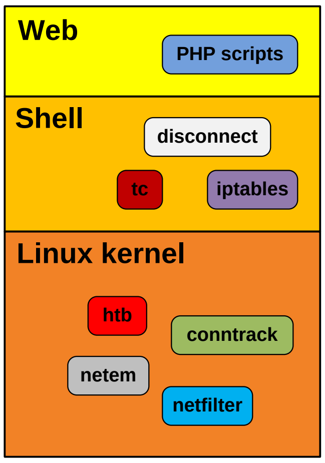


Figure 2.8: WANem architecture [22].

WANem components are presented in Figure 2.8. These tools behind WANem's user interface are open source and thus, can be used separately, but WANem just brings them together in an easy-to-use package. The main functionality is produced with Iproute2's Tc utility in combination with its NetEm extension.

However, the emulation of disconnecting hosts is something not seen in other emulators. WANem uses Linux conntrack module to follow the state of the connections going through the emulator, and models the unreliability of networks by terminating the connections the way the user has configured.

WANem is somewhat different to the rest of the emulators reviewed, as there is no white paper written on it. The reason for this seems to be that it has grown from the industry and not from university projects. Thus, there is no reliable data available on how large a network it can be used to emulate. Nevertheless, the project seems to be active and developers respond to the user's questions on the project forum in [22]. Also the documentation there is quite thorough, so WANem usage is quite straightforward.

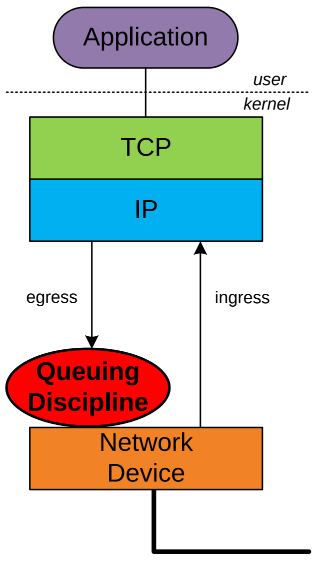
2.2.7 NetEm

NetEm [16] is yet another WAN emulator for LAN environment. NetEm grew from the fact, that most tools available for testing protocols and software in controllable environment emulating the Internet, were either expensive, proprietary software, or limited research projects. Therefore, it uses Tc facilities in widespread Linux 2.6 kernels to accomplish network emulation. It was originally built to evaluate TCP enhancements in 2.6 kernels over high speed links, but it has found usage in other emulation areas as well. [16]

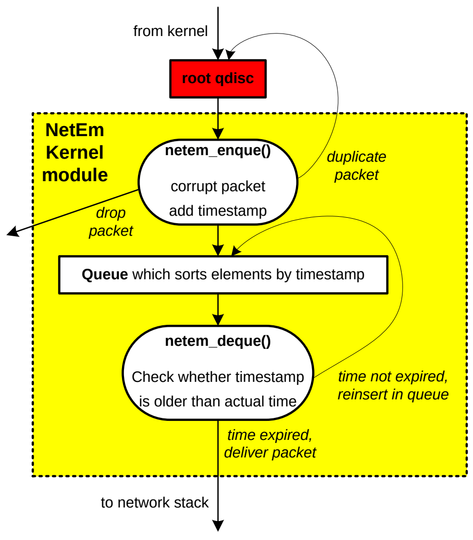
NetEm reuses some emulation code from open source NIST Net, but the emulators are different by architecture. NetEm integrates on Linux traffic shaping mechanism that bounds its emulation to egress traffic, while NIST Net operates on ingress packets. NetEm consists of two parts, a small kernel module for queuing discipline (qdisc) and command line utility for configuration. Netlink socket interface [50] is used in communication between these two. In Figure 2.9 the emulation principle of NetEm is presented. In Figure 2.9(a) the location of qdiscs in Linux is shown, and NetEm kernel module operation principle can be seen in Figure 2.9(b).

NetEm was build to be configured using Tc on the command line, but there exists an external GUI as well. MasterShaper [52] simplifies the configuration by hiding complex Tc commands behind its intuitive Web interface. Another way to have GUI is to use WANem reviewed earlier, which ships with a web interface.

Configuration of NetEm is done using Tc. The same network characteristics are tunable,



(a) Linux qdisc location [16]



from kernel
root qdisc
NetEm Kernel module
netem_enque()
corrupt packet
add timestamp
duplicate packet
drop packet
Queue which sorts elements by timestamp
netem_deque()
Check whether timestamp is older than actual time
time not expired, reinsert in queue
time expired, deliver packet
to network stack

(b) NetEm kernel module [51]

Figure 2.9: NetEm emulation principle.

as with other network link emulators; packet delay, loss, duplication, corruption, and re-ordering. Netem_enque(), visible in Figure 2.9(b), performs these actions to the packets. Rate control is achieved with other qdiscs in Linux kernel, which are configured with the same Tc tool than NetEm. As NetEm itself is a layered qdisc, it can be used simultaneously with other qdiscs. This allows for an extensible and wide array of options to shape the traffic passing through the emulator. In addition to all these, NetEm can be used in network bridges or routers, so both Layer 2 and 3 emulation is possible. [16]

Even so, NetEm cannot emulate Internet perfectly, because it is too complex for NetEm to handle. NetEm works best on single flows, not on extensive networks. Multiple limitations exist, the most important of which are explained next. The usage of Pseudo-Random Number Generator (PRNG) in NetEm impacts emulation results, and not all devices are ready for sudden bursts of traffic the NetEm generates at high loads. [16] The data structures used in NetEm might not be suitable for storing large amounts of delayed packets. [15]

The packet delaying mechanism in NetEm might cause unintentional re-ordering of different size packets. If two packets arrive nearly simultaneously at the emulator and they differ in size significantly, the smaller packet might get priority over the larger one regardless of their arrival order. [53] NetEm emulation cannot be done on incoming path, which limits its usefulness in a case where the user wants to perform emulation on the device where NetEm is running [33]. To enable emulation on ingress traffic, Intermediate Queuing Devices (IMQ) [54] might be used, but the extra overhead and high CPU usage of such solution might become an issue [4].

On the positive side, the timer issue existent on older kernels seems to have been solved in the recent mainstream kernels. High resolution timers are available from kernel 2.6.16 onwards [55] and they offer finer granularity not dependent on the kernel tick rate. If this feature is enabled, NetEm will use it by default on 2.6.22 or later kernels [23].

As with NIST Net, NetEm shares the same issues of self-similarity of cross traffic and long range dependence than were discussed in Section 2.2.5. Trace Control for NetEm (TCN) is a port of the trace reader module found in RplTrc to NetEm, and it can be found at [56]. Both RplTrc and TCN use real network traces to obtain more accurate emulation results. Usage of TCN is fairly well documented [57], although outdated on some areas. Some example traces are available at TCN website and more trace files can be generated, although it is a relatively complicated procedure.

In contrast to RplTrc, TCN includes trace generation tools, and their usage is documented in [57]. The white paper written in German explaining TCN more thoroughly can be found at [51]. There were discussions on NetEm mailing list [58] to include TCN module in NetEm, but when asking the author of TCN about the matter, he answered that this was not done due to architectural reasons.

In [4] NetEm using high resolution timers was found the most accurate among network link emulators. In their test runs with maximum size Ethernet frames of 1500 bytes, NetEm performs accurately up to 500 Mbps speeds. Performance in speeds higher than that, are still tolerable in contrast to other emulators. The intelligent algorithms used in high resolution timers outperform the ones used in Dummynet and NIST Net in latency and bandwidth limitation accuracy. High resolution timers also guarantee a lower overall CPU usage in NetEm compared with other emulators. [4] TCN, on the other hand, has been tested to work reliably at rate of 80 000 pkt/s [51].

NetEm is part of Linux 2.6 kernel and iproute2 [59] package. Thus, obtaining NetEm is easy, as it is already in most of the kernels. As being part of GNU/Linux, it is open source.

Usage examples of NetEm can be found at its web site [23]. A thorough introduction to Linux traffic control capabilities and the usage of Tc tool can be found at [60]. Answers to more advanced questions are browsable on NetEm mailing list archives at [58].

2.3 Evaluation of the tools

After an extensive literature survey it is evident that there are plenty of options to choose from for RTP2P testing environment. As mentioned at the beginning of this chapter, only simulation and emulation options were considered, as resources were limited for live testing.

Simulation is a convenient method for piloting the concept of a new application. Simulation scales to the limits other methods cannot reach, and are hence used to find the limits of the researched system. It is also a cost-efficient method for testing how an application would perform in a complex networking environment, but results depend heavily on the accuracy of the model built.

Nevertheless, for this thesis, simulation was not a suitable option. As the working prototype of RTP2P application already existed, building a model of the RTP2P was seen as extra work. In fact, building the model might be a topic for another thesis altogether.

Even though the only option left is network emulation, it is at the same time the best option for testing RTP2P performance. As the RTP2P is in a prototype phase and the tests performed with it are preliminary by nature, a stable and configurable environment is required to locate the defects and boundaries of its operation.

The emulators at the first two categories, virtual network emulators and experimental testbeds, seemed at first the most promising. They are wide-scale emulators capable of emulating the behavior of the Internet more accurately than network link emulators. PlanetLab and Emulab seem the most interesting for testing an application performance on an Internet-like wide-scale network. Emulab might be a better option as it promises reproducibility and stable emulation architecture for the tests, which PlanetLab by design cannot offer. Besides, PlanetLab nodes can be used via Emulab as well.

Unfortunately, as the RTP2P application had some patenting issues unsolved at the time of this thesis, neither Emulab nor PlanetLab could be used. This left only ModelNet to build a wide-scale P2P network environment.

However, ModelNet was not used because of several reasons. Firstly, the lack of resource isolation would have required a separate traffic shaper to be used, or even some of the network link emulators. Secondly, there has been no activity in ModelNet project for years, the software itself is quite aged, and there is no guarantee of its operability on recent kernels. Thirdly, ModelNet seems to work only on static topology, which does not suite RTP2P's dynamic overlay creation. Finally, the documentation of the emulator was inadequate, so the setup did not succeed with reasonable effort. Still, the idea about the virtualization of the peers on a single machine was tempting.

Besides, the wide-scale emulators lack the ability to model certain properties in P2P networks, such as peer dynamics, various connection types, and link delays. In addition, especially in PlanetLab, there might arise issues with random CPU usage and fluctuating available bandwidth due to its architecture. [5]

So, only network link emulators were usable for the testing environment. They seem to be the most suitable for the purpose of this thesis, as the network environment for RTP2P has to be built on local premises on existing hardware. They are no longer in a prototype phase and are of production quality. They are freely available and they are already in extensive use in the research community. [4] The results obtained from using network link emulators might not be the most realistic, but they are sufficient for the first analysis of RTP2P system.

Network link emulation does not correspond to wide-scale emulation, because it lacks proper background traffic to model the unpredictability of the network. Still, emulating certain parameters like packet loss rate or network delay on a limited bandwidth link, serves as a good starting point. In addition, most P2P applications are used on edge nodes and thus, realistic core network emulation is not necessary. The link between the edge node and Internet Service Provider (ISP) is the bottleneck in most of the cases [61]. Emulation of this link should be sufficient to gain the overall idea of the application performance.

RplTrc and TCN try to provide more Internet-like results for network link emulators by using traces on the basis of the emulation, but it is dubious whether it is worth the trouble.

As Floyd and Paxson [7] state, the Internet topology is so complex and ever-chancing that producing similar effects on a laboratory environment is an extremely challenging task. When there are wide-scale emulators like PlanetLab to do the job more reliably, why bother trying to build one of your own?

Network link emulators are sufficient for preliminary testing of RTP2P. By using them, some estimates for RTP2P performance limits can be found. By building a local network environment the stability, configurability, and reproducibility of the results can be assured. Only the decision, which emulator to use, is left before the assembly can begin.

WANem is an interesting project with its intuitive web interface and host disconnection emulation, but as it is delivered only on Live CD it was discarded after a few bad experiences with unreliable CD drives. In P2P environment connection tracking required for disconnections was found too demanding for the server to handle, too. Installation to hard drive might be possible enabling WANem use on server hardware, but as plain NetEm is capable to the same as WANem, it was not chosen.

In a desire to concentrate on Linux platform, Dummynet was ruled out in the first place. The choice between NIST Net and NetEm was quite evident, as NIST Net seems to be abandoned by its developers, while NetEm is part of the mainstream Linux kernel. In fact, NetEm can be seen as a successor in Linux for NIST Net. In spite of their architectural differences, they offer pretty much the same emulation options as can be seen from [4]. In the same comparative study Nussbaum and Richard [4] found NetEm, running in the recent Linux kernels, the most reliable and accurate network link emulator available.

The continuing support for NetEm, understandable and comprehensive manual, and browsable mail archives for more detailed instructions offer enough information for its deployment. If the complex configuration interface is too demanding to handle, MasterShaper GUI could be used. When the constraints discussed in Section 2.2.7 are kept in mind, NetEm is the proper tool for the testing environment. Still, NetEm requires reasonable expertise in Linux advanced routing and queue disciplines for proper usage.

On a side note, recent efforts on Dummynet integration to PlanetLab and Linux port, might have improved its usefulness on network link emulation. Especially in the case where RTP2P is needed to be further tested using PlanetLab, this might become current. As Tc in

PlanetLab is already used on traffic shaping [33], for reliability's sake Dummynet might be a better solution for handling the link emulation than NetEm.

3 LABORATORY SETUP

This chapter explains the work done to build the network environment. Knowledge gained from the literature survey in Chapter 2 is combined with general Linux administrative effort to assemble the environment piece by piece. A short comparison of different virtualization solutions is also given.

The research group had performed preliminary test runs on the RTP2P system to prove some of its features, but thorough performance testing had to be postponed due to lack of time. As the laboratory environment does not reflect to real usage scenarios for the system, an emulator was needed to add realistic delays, packet losses and other irregular behaviors similar to the Internet. This thesis focused on building the realistic environment. As the teaching laboratory at DCE premises had been used for preliminary test runs, the same resources worked as a basis for the testing environment as well.

The laboratory is in active use by networking students at TUT, which complicated the construction of a permanent and reliable environment in the laboratory. Unscheduled configuration changes or even reinstallation of operating systems on PC's at the laboratory might take place without warning. The full schedule of the laboratory leaves no time for tedious reconfiguration in case more test runs are needed in the future. This lead to an idea of virtualization.

3.1 Virtualization

Virtualization is the physical resource partition into an arbitrary number of virtual guests [41]. In other words, the main idea in virtualization is to use the available resources better by combining multiple servers into one. This technique is called server consolidation. [62]

However, for this thesis, more important feature of virtualization is its ability to encapsulate the resources dedicated to the guests.

A virtualization layer allows running guest Operating System(s) (OS) on a host system. Depending on the virtualization solution this layer possibly introduces overhead to the application performance run on the guest OS. Host machines work as a Virtual Machine Monitor (VMM), also known as hypervizor, that is responsible for the virtualization. [62] Virtual machine (VM) is used as a synonym for a virtual guest in this thesis.

The original design of x86 architecture did not include virtualization, so alternative means to perform virtualization were necessary. These virtualization solutions can generally be divided into three categories seen in Table 3.1, differing in the way the virtualization layer is implemented. [62]

Table 3.1: Hypervizor categories.

Category	**Hypervizors**
Full-virtualization	KQEMU, KVM, VirtualBox, VMware
Para-virtualization	Xen
Virtualization at OS level	Linux-Vserver, OpenVZ

First category is *full-virtualization*, which virtualizes the entire hardware stack. It is very flexible as it supports almost any OS for x86 architecture, but it is not very efficient due to hardware emulation. Nevertheless, it is the most realistic as it virtualizes full operating systems [61]. Most of the hypervizors, such as KQEMU [63], KVM [64] and VirtualBox [65] in this category are based on QEMU [63]. [62]

The next category contains the *para-virtualization* solutions, where only some of the physical resources of the host machine are used. The key in para-virtualization is to make the guest OS aware that they are virtualized. This typically requires modifications to the quest OS kernel. The benefit of the modifications is that the hypervizor offers almost direct access to some of the physical resources of the host machine. Para-virtualization is a highly efficient technique, but quite invasive because of the needed kernel modifications. Xen [34] is the most common product in this category. [62]

The last category is *virtualization at operating system level*, where the virtualization layer runs on top of current OS. It is the most efficient of the three, as there is only one kernel in execution at any given time. This concurrently makes it the least flexible, as the same kernel has to be used in both host and guest systems. Linux-Vserver [38] and OpenVZ [66] are examples of this category. PlanetLab nodes, for instance, use Vserver to gain the isolation of slices. [62]

Recently, x86 has received additional extensions that enable the implementation of classic VMM [67] without the need for artificial tricks. KVM uses these extensions to turn Linux into powerful hypervizor. Recent Xen revisions are capable of using them, too. [62]

Out of the commercial virtualization solutions, VMware [35] has to be mentioned. VMware falls into the first category as it provides full-virtualization. VMware is a de-facto virtual environment, mainly focused on enterprise server virtualization, but they offer workstation solutions as well. However, the workstation versions that are capable of building a virtual machine, required the purchase of a license, which made them unsuitable for this thesis. Although the peers could not be built using VMware, the powerful server responsible for the emulation was built on top of VMware ESX. This server is called WANemulator from now on. Still, a hypervizor for peers needed to be found.

The comparison of open source virtualization solutions explained in [62], found Linux-Vserver the most efficient way to perform virtualization, giving almost no virtualization overhead. Vserver might have been suitable for building the environment for this thesis, as there is no need to use other kernel than Linux. Unfortunately, Vservers do not help in the issue of possible reconfiguration in the laboratory, because their configuration is done inside the physical computers. The same applies to OpenVZ.

A solution offering a full platform virtualization would be better, as with them a new operating system can be built inside a disk image and stored elsewhere when not needed. Deployment of this image can be automated by bash scripts, that modify the setup the virtual machines require. Thus, the virtual environment can be built rapidly, even after full reinstallation of the teaching laboratory has occurred.

In platform virtualization the operating system is separated from the underlying platform resources. By selecting a suitable platform virtualization solution, no extensive changes in host OS will be necessary, which keeps the laboratory intact and still suitable for teaching. Xen was found the next best solution in [62]. Xen is capable of platform virtualization.

The issue in Xen, that guest OS kernels have to be patched is not a problem, as there are multiple pre-built Linux distributions available to choose from. One of them should have fulfilled the requirements for the network environment. However, the architecture Xen uses is problematic. Thin virtualization layer is installed between hardware and VMs, requiring a boot up of the physical machines to run Xen. It might be possible to run non-virtualized OS on the side of Xen, but as Xen is designed with server virtualization in mind, this might not be wise in performance's sake. The default configuration of Xen also captures the network configuration on a host. This might have required a lot of configuration effort to keep the laboratory PCs in use for teaching as well.

Only full-virtualization solutions seemed suitable for this thesis. KVM is shipped with Linux kernel, and is in other means an interesting product, but [62] found it unsuitable for heavy networking situations. When KQEMU was also seen inappropriate for production use, VirtualBox seemed the only option to use. VirtualBox was already installed onto the laboratory computers and it offered all the necessary features required. VirtualBox is also designed to be run on host OS as a normal process, but still it supports direct hardware virtualization and usage of multi-core CPUs. Furthermore, the author of this thesis was familiar with VirtualBox, which decreased the learning curve.

VirtualBox, as any other full-virtualization product, is not very efficient in terms of overall performance, but at least networking efficiency is high, as stated in [62]. In the same study scalability of virtualization was found to be heavily dependent on the nature of the workload. RTP2P is quite resource consuming both in terms of CPU usage and network traffic, so it was unclear how VirtualBox copes with RTP2P. By decreasing the amount of VMs on one PC and peers in the overlay, reliable and stable virtualized testing environment should be achievable.

Usage of virtual machines solved the reconfiguration issue. As long as physical hardware remains the same, it is only necessary to copy a single file containing VM image to the computers, install VirtualBox software by the help of shell script, and run the tests. The guest operating system can be chosen regardless of the OS running on the host computer.

To some extent, it might be even possible to install the network environment to other premises as well, if certain prerequisites are fulfilled. By intelligently choosing the parameters in virtual machines, the existing wiring in the laboratory could be used, which revoked the need for extensive reconfiguration as well. No need for complex VLAN setups, just reboot the laboratory into virtual environment and start testing.

3.2 Logical structure

In Chapter 2 NetEm was found to be the most suitable emulator. Using NetEm at WANemulator, a traffic shaped network environment was built. Centralized solution, seen in Figure 3.1, for emulation was chosen mainly because of its simplicity. Using a powerful and dedicated Linux server running as the emulator was easier to build and less error prone than distributed emulation performed at P2P network nodes. As [4] points out, a dedicated router concentrating only to the emulation should handle P2P traffic. Still, emulation at the end nodes is also possible as Tc tool and NetEm is installed there as well.

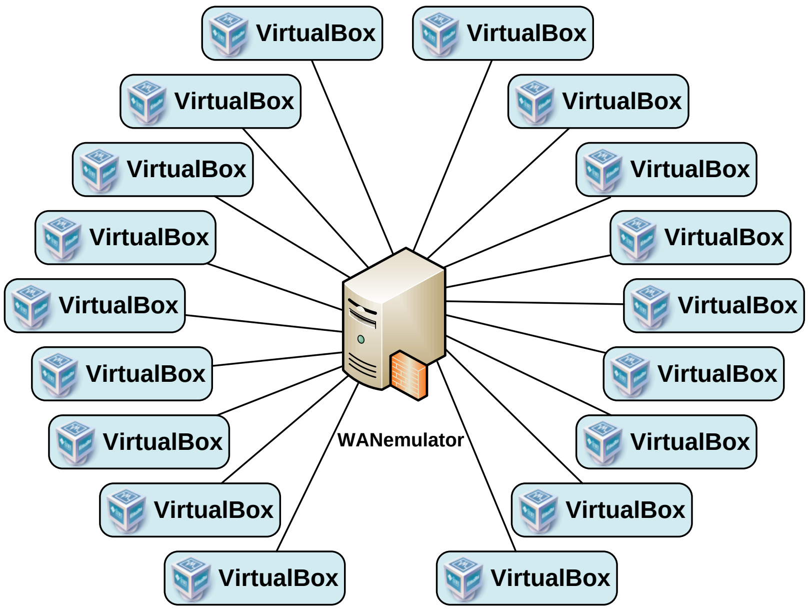


Figure 3.1: Network environment components.

However, P2P environment might cause some unexpected traffic due to its nature. P2P network is in many ways more complex than normal client-server environment. P2P network consists of logical overlays established over existing physical network. The overlay links are completely virtual, and not tied to the physical links. All end nodes are potential neighbors to each other, which will increase the connection numbers and thus, traffic going through the router. By keeping the amount of peers in the overlay small should keep the connections and traffic rates low enough. Still, too low amount of peers would invalidate the idea of testing the scalability of the RTP2P software.

If connection tracking is required in the router, it might overload. Luckily, this is needed only if stateful firewalling is necessary. RTP2P does not require firewalls, it runs on LAN dedicated for the testing environment. The neighbor links are usually stored to a table in a node, which requires a lot of memory in case of a large network.

Peer churning is also an issue, as peers might suddenly leave and join the network. Therefore, each peer establishes multiple neighbor links just in case, which in turn increases the memory needed at peer. This might became an issue with mobile devices, but the virtual machines are running on modern PC's where memory exhaustion should not be a problem. The issue can also be circumvented with clustering in the overlay, as RTP2P does.

The network environment consists of two parts, as represented in Figure 3.1. *WANemulator* is the actual emulation node, which serves at the same time as a router and DHCP/DNS server. WANemulator is used in routed mode, where routing is used to force the traffic through the emulator. The other part consists of *virtual machines* running on physical computers in the laboratory.

Figure 3.2 presents the physical setup of the system, where multiple virtual machines are running simultaneously on one host computer. Inside a virtual machine the actual RTP2P application is multiplexed as well. This offers quite versatile setup when it comes to peer numbers in the P2P overlay. As RTP2P is built on Linux, the virtual machines are running Linux as well. RTP2P does not support IPv6, so the environment was built for only IPv4.

3.3 Physical structure

The test environment is built into the DCE teaching laboratory, which consists of 17 Dell Optiplex 755 desktop computers. The computers have one Intel Core2 Duo 2.33 MHz CPU, 4 GB of 800 MHz DDR2 memory and an integrated Intel 82566DM Gigabit LAN network

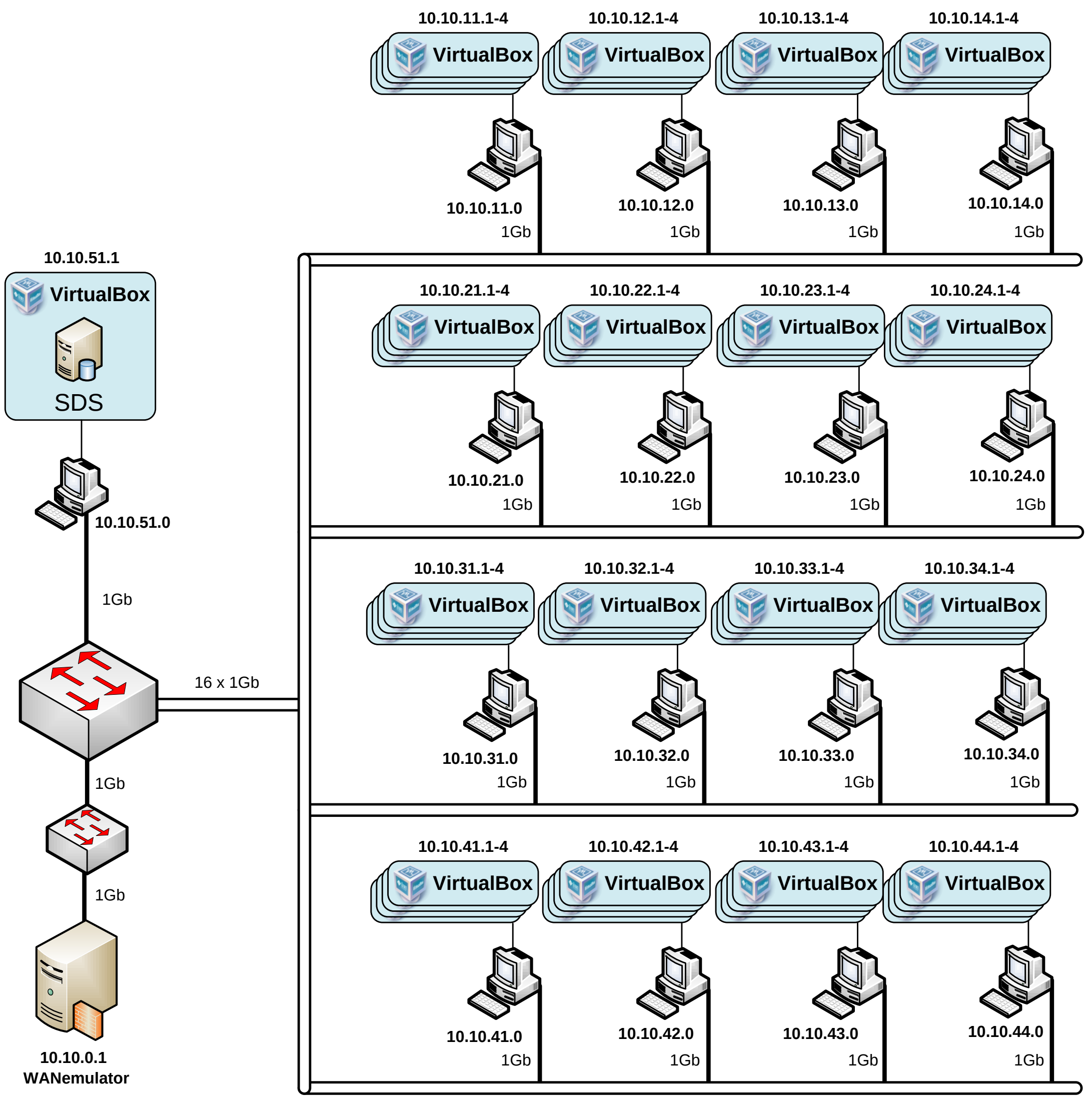


Figure 3.2: Testing environment setup.

card. Usage of both CPU cores is enabled from BIOS, as is the Intel VT-x support for hardware virtualization.

Host operating system installed onto the computers is CentOS5 GNU/Linux kernel i686 2.6.18-128.7.1.el5PAE and the used hypervizor is Sun VirtualBox 3.0.4 r50677. The testing environment uses the private address space of 10.10.0.0/16. The host computers have IP addresses for administrative purposes. Hosts are configured to the same network as VMs making the the capability tests performed in Chapter 4 possible.

The network setup can be seen from Figure 3.2. All the equipment is connected together through a Cisco Catalyst 3750 switch and Ethernet wiring in the laboratory is capable of 1 Gbps. WANemulator is running on VMware ESX server as virtualized router, DNS and DHCP server. It uses two Intel Xeon 2.50 GHz CPUs, 8 GB of physical memory and two Gigabit network interfaces using vmxnet network driver. The other interface, not seen in Figure 3.2, is used to remotely control the environment and it serves as an uplink to the Internet for the laboratory PC's, too. Uplink is needed when the software at host or guest PCs needs to be updated from the package repository.

WANemulator has Debian Lenny 5.0.2 OS with GNU/Linux 2.6.26-2-686 bigmem kernel to enable 8 GB of memory. Debian was chosen as it provides the most up-to-date package repository and is a widely supported Linux distribution. WANemulator kernel is compiled with 250 Hz resolution as it was the default option. With earlier kernels, the 250 Hz resolution on WANemulator might have caused inaccurate emulation results, but high resolution timers enabled in recent kernels solved the issue. In [4] it was found that when high resolution timers are used, the tick rate does not affect the results.

WANemulator performs simple NATing using iptables [68] for VMs and hosts, in case they need access to the outside world. RTP2P itself in its current configuration does not require access to remote peers outside the laboratory, but to gain access to package repositories or remotely control the environment, NAT is necessary. Dnsmasq v2.45 [69] is used as a DNS forwarder and a DHCP server on WANemulator, as it was a straightforward method of building the test network.

With Dnsmasq, network options can be delivered to virtualized clients using DHCP based on the MAC address of the Network Interface Card (NIC). The delivered options include also modified routing tables to the clients seen in Table 3.2, which forces all the IP packets to go through the WANemulator.

Table 3.2: Routing table used in the virtual machines.

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.0.0       10.10.0.1       255.255.192.0   UG    0      0        0 eth0
10.10.0.0       0.0.0.0         255.255.0.0     U     0      0        0 eth0
0.0.0.0         10.10.0.1       0.0.0.0         UG    0      0        0 eth0
```

This duplicate routing table entry to the lowest quarter of the local domain (10.10.0.0/18), is necessary to force all traffic to these addresses through the WANemulator, discarding network switch behavior. This made traffic shaping possible. Also ICMP redirects needed to be turned off from the router and clients, to guarantee the wanted behavior. The physical machines also received their network configuration via DHCP, but they do not have duplicate routing table entry. This ensures, that any control traffic necessary to them during a test run does not go to WANemulator and hence, disturb the run.

Irqbalance is installed to balance the high interrupt load generated by routing decisions on two CPU cores at the WANemulator. Also VMware Tools are used on WANemulator to ensure high performance. As NetEm comes preinstalled with Linux kernel, there is no steps required for its installation.

On virtual guests, Debian Lenny 5.0.2 operating system with GNU/Linux 2.6.26-2-686 kernel running on 250 Hz tick rate was installed. Some tweaking was necessary to obtain the hostname from DHCP. Debian tends to discard the hostname by default and uses its predefined value received on installation. RTP2P only worked on IPv4, so IPv6 was disabled from guests. The operating system at virtual hosts was kept as minimal as possible, but still capable of running multiple RTP2P console instances.

The laboratory computers are capable of 64-bit operating system, but the CentOS installed on them was only 32-bit. Even though VirtualBox would allow running a 64-bit guest operating system on 32-bit host, this was not tested. This would have caused additional overhead [70] and would not have given that much advantage in performance with this hardware. Real benefits from the 64-bit environment comes only with proper server hardware, which was not available.

As virtual machines are the backbone of the client side of the environment, more detailed explanation of their configuration is necessary. The description of VirtualBox setup follows.

3.4 VirtualBox setup

Even though any significant changes on the host PCs running CentOS were not required, it was still necessary to setup the virtual machine configuration on every host. Setup was first experimented on one computer. When the setup was discovered stable and suitable for the testing environment, a shell script was composed to automatically deliver this setup to all PCs.

VirtualBox was updated to the most recent version, which no more required using tun/tap interfaces in bridged mode and offered multiple other improvements as well. KVM module, loaded on some of the computers, had to be disabled before VirtualBox could be run, as two simultaneous hypervizors running at the same time is not recommended.

Separate user account for research projects was installed onto the host PCs to separate all the data used in this thesis from regular users. The account will serve other research projects as well in the future. Root account could have been used for the same purpose, but it was kept only for maintenance use. Hereby, as the sensitive data is encapsulated inside virtual machines and VMs are run on separate user account devoted solely on research purposes, the confidentiality of the environment is ensured.

Now, when the host computers were prepared for virtual machines, the next step was to build the necessary configuration for them. As there are total of 17 computers where the setup has to be done, a few shell scripts were created to automate this process. Using these scripts it is straightforward to build the environment after the laboratory has been reinstalled for some reason or another. The configuration of VirtualBox can be divided into two parts, *virtual image* and *virtual machine* configuration.

3.4.1 Virtual images

As mentioned earlier, the guest operating systems are encapsulated into virtual disks. The format used is VirtualBox's own Virtual Desktop Image (VDI) even though open Virtual Machine Disk (VMDK) could have been used. This would have made the disks interoperable with VMware, but as it was thought unnecessary, the native container for VirtualBox is used. Two different kinds of disks are prepared. The one used with Service Discovery Server (SDS) is larger in size and more close to a traditional hard drive in features. The other is more modified to suit the purpose of multiple simultaneous VMs on the same physical host.

Two disks are necessary as SDS uses SQL database for storing information of each RTP2P test run. Test runs, also known as services, require a large amount of disk space. The SDS disk is used in normal mode, so it remembers its state on reboots. It is dynamically expanding virtual hard drive, that has lower initial disk size, while it still retains the scalability the SQL database needs.

Table 3.3: Settings of an immutable image.

```
UUID:                   f1811eda-85b3-41b5-a679-4d5bc3c03733
Accessible:             yes
Description:
Logical size:           700 MBytes
Current size on disk:   700 MBytes
Type:                   immutable
Storage format:         VDI
Location:               /home/research/.VirtualBox/HardDisks/client.vdi
```

The image settings seen in Table 3.3 for the other 16 host computers required some extra thought. Multiple VMs on the same host requires the client image to be in immutable mode. In the immutable mode, all changes to the disk are lost when the virtual machine is powered off. The benefit of immutable mode is that the virtual machines running on a host PC can use the same disk image.

When virtual machines are running, the changes are stored in temporary files on host computers and these are overwritten when the VM is rebooted. This simplifies the setup significantly. The size of the image was kept fixed at 700 MB to ensure its proper functionality in multi-VM situation. In the case when change in a client image is needed, the only step required is to make the changes in a master image and then copy it to the hosts thus, overwriting the old image on them.

The decision of using VDI image is not permanent, as there exists tools that can change the format of the image. If other virtualization solutions like VMware or Xen are desirable, the images prepared for VirtualBox can be converted to forms that the other hypervizors use.

3.4.2 Virtual machines

The same division as with virtual images applies to virtual machine configurations as well. There are two different kinds of setups, one for the VM running SDS and original data source, and another for other laboratory computers. The most important settings used for the latter are illustrated in Table 3.4.

Table 3.4: Virtual machine settings.

```
Name:             client441
Guest OS:         Debian
UUID:             1cff6904-d10c-49d0-84cc-84a30a4874cc
Config file:      /home/research/.VirtualBox/Machines/client441/client441.xml
Memory size:      1536MB
VRAM size:        8MB
Number of CPUs:   1
Boot menu mode:   message and menu
Boot Device (1):  HardDisk
ACPI:             on
IOAPIC:           on
PAE:              off
Time offset:      0 ms
Hardw. virt.ext:  on
Nested Paging:    on
VT-x VPID:        on
Primary master:   /home/research/.VirtualBox/Machines/client441/Snapshots/
                  {3fc4adb6-c881-4a9c-aac0-17aa7dbd9140}.vdi
                  (UUID: 3fc4adb6-c881-4a9c-aac0-17aa7dbd9140)
NIC 1:            MAC: 080027229441, Attachment: Bridged Interface ’eth0’,
                  Cable connected: on, Trace: off (file: none), Type: 82543GC,
                  Reported speed: 0 Mbps
```

The example in Table 3.4 is from VM identified as client441, which has IP address 10.10.44.1 in Figure 3.2. All the other VMs, excluding the one running SDS, have similar settings. Only two VMs are ran simultaneously on one host computer due to reasons explained more thoroughly in Chapter 4. That is why 1536 MB of memory is available for one virtual machine, and still almost 1 GB is left to ensure proper function of the host as well. When two VMs are run simultaneously, both VM processes use one core from the host CPU. VirtualBox is capable of using more cores for one VM, but as the host has only one dual-core CPU, this feature is not used.

ACPI stands for Advanced Configuration and Power Interface, it is used by OS to recognize hardware and manage power. It is turned on by default. I/O Advanced Programmable Interrupt Controllers (IOAPIC) allow OS to use more than 16 IRQs to avoid IRQ sharing for improved reliability. As the hosts have only 4 GB of memory, the need for Physical Address Extension (PAE) is not necessary. Different time offset could be enabled to run guest operating systems in the past or the future, but this is not necessary.

The next three settings, hardware virtual extensions, nested paging, and VT-x Virtual Processor Identifiers (VPID), concern the hardware virtualization enabled by Intel processors at the hosts. VirtualBox does not require these settings to be turned on, but in general they improve the performance of the VM. The first offers hardware support for any modern CPU, but nested paging requires at least Core i7 (Nehalem) processors. Even though the laboratory computers do not support this feature, it is turned on as it does not seem to degrade the performance. VPID can greatly accelerate context switching and is therefore turned on.

Also the immutable image operation of VMs can be seen from Table 3.4. The master hard disk designated to the VM is not the actual read-only disk client.vdi, but a temporary file used to store changes between poweroffs. This allows other virtual machines running on the same host to use the same immutable image. VMs just load the contents of client.vdi at boot and use the temporary file from there on. As long as the VM is not powered off the changes to the temporary file are kept, even reboot of the guest OS is possible.

The last setting in Table 3.4 is one of the most important ones at the setup. VirtualBox offers many different options to choose from for virtual networking, but bridged networking is the only practical one for the testing environment. The guest OS sees a software bridge interface at Layer 2, which is connected to the physical interface at the host. With this feature full Ethernet connection to the network is achieved for the VM, which is essential for proper function of RTP2P. Emulated Intel PRO/1000 T Server is chosen as it offers a Gigabit link for the interface.

When it comes to the VM running SDS and original data source for the RTP2P service, which has IP address 10.10.51.1 in Figure 3.2, there are only two differences compared with the other VMs setup explained above. Firstly, it has more RAM available, 2560 MB. Secondly, it uses both cores of the host computer's CPU. This ensures it never runs out of memory and it always has enough CPU power available. This is necessary as the MySQL database is using reasonable amount of CPU time in the case where a large P2P network is in operation.

3.5 VirtualBox operation

VirtualBox has a versatile command line utility, VBoxManage, to control the virtual machines. As the host computers and VMs running on them are configured and used remotely over SSH, this ability is invaluable.

VirtualBox GUI is the front-end most end-users use, but to gain access to all the configuration settings, VBoxManage has to be used. Although VBoxManage can also be used to start, stop, and control running VMs, for this thesis it is mostly used to configure and view the settings of VMs. All the registering, creating, and modifying of both virtual images and virtual machines are done using shell scripting and VBoxManage.

Another front-end used to start the virtual machines remotely is VBoxHeadless. VBoxHeadless does not produce any visible output on the host, but instead only delivers VirtualBox Remote Desktop Protocol (VRDP) data. VBoxHeadless is used if virtual machines are run on servers and the user wants to avoid using a graphical user interface on the host. As the guest operating systems on virtual machines are accessed via SSH only, VRDP on the VMs was turned off.

VirtualBox is an extremely diverse virtualization software, so only minor part of the capabilities of the hypervizor are used for this thesis. An intrigued reader can find more about VirtualBox capabilities from [70].

4 ENVIRONMENT CAPABILITY TESTING

Before the testing environment is used for delaying and shaping the traffic, it needs to be proved that it actually functions reliably. Thus, in this chapter the network environment capabilities are proved. The natural solution for this would be using the RTP2P system itself.

4.1 Using RTP2P

Testing was started by analysing the traffic characteristics of RTP2P. The observations of the RTP2P system stated in this chapter are based on [1], which explains the working principle of the application. The operation of RTP2P is not repeated in this thesis, only relevant parts of RTP2P for succesful tests with the environment are mentioned.

Using Wireshark [71] on one of the virtual machines the traffic during a live test could be captured and analyzed. Wireshark reveals that RTP2P uses TCP to carry RTSP traffic for signaling between SDS and peers, and the streaming traffic goes over UDP in RTP packets. As signaling traffic is only minor part of the data flow in the capture, TCP traffic is not interesting at this time. The UDP packet size distribution is seen from Figure 4.1, which indicates UDP packet lengths having normal distribution with a heavy tail. Out of the capture, it was seen that port on the sender side is 50247 and on the receiver side 8234.

The packet rate of the data stream depends on the video encoding at the original data source. The media used in these tests was encoded with FFmpeg using H.263+ codec for video and AAC codec for audio. A RTP packet was generated at original data source once every 55 ms, which was ensured from the Wireshark capture. 55 ms gap between packets corresponds a packet rate of approximately 18 RTP packets encapsulated into UDP per second.

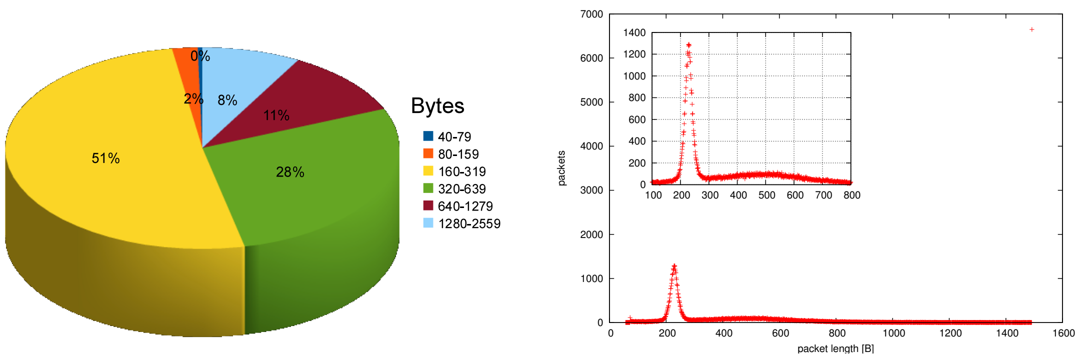


Figure 4.1: UDP packet lengths from RTP2P run.

As all the other peers joining in the overlay use the same video, the rate at them is also 18 pkt/s.

However, as P2P environment is complex as discussed in Chapter 3, the actual rate the networking interface has to handle might be much higher. The traffic is uploaded and downloaded simultaneously, and there might be many neighbours requesting the data. The RTP2P system currently limits the upstream connections to eight peers, but the average amount of traffic going through the interface still has to be estimated.

The estimate of the traffic rate was discovered using Bwm-ng [72] monitor running on both WANemulator and virtual machines during a live test run. There were five peers running on each virtual machine and two virtual machines on each PC, resulting in the total of 160 peers in the overlay. In addition to these, original data source and SDS were running together on one extra virtual machine. A new peer joined to the overlay every 20 seconds, so the maximum amount of traffic in the environment was flowing after 54 minutes. Streamed video content lasted for 62 minutes, which is the length of one test run in this thesis.

All the traffic was routed through WANemulator, but no emulation was performed. As retransmission is one of RTP2P's novel features, two test runs were performed, one without retransmission and another where the feature was turned on. The results seen in Figure 4.2 represent the total traffic going through an interface, in other words a sum of all upstream and downstream traffic. The term router corresponds to WANemulator in the figures.

From the results in Figure 4.2, the growing trend of the traffic can be seen. This is caused

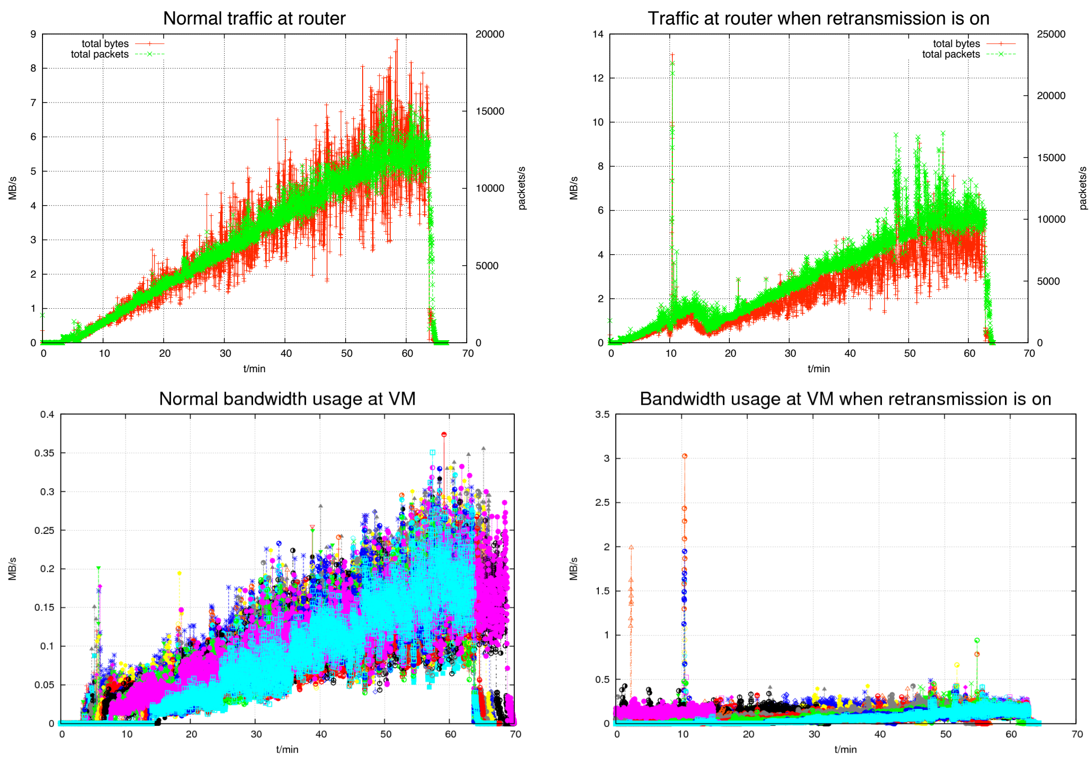


Figure 4.2: Traffic generated in RTP2P run.

by joining peers while the run progresses. The largest amount of traffic is flowing through the WANemulator near the end of the test run, when all the peers have joined the overlay. The packet rate follows the throughput value.

In retransmission run seen in Figure 4.2, one spike of traffic can be detected at the beginning of the run, which is most probably due to a completely lost segment in the stream, causing high amounts of retransmissions. The spike can be seen at the router and the virtual machine figure.

The bottom row in Figure 4.2 shows traffic measured at the interface on virtual machines. The growing trend of the traffic can be seen in here as well, in the end each virtual machine has five peers running. The colors represent different virtual machines, so there is total of 32 different colors in the VM figures, even though not all of them can be seen due to the high amount of measurement data. The VM figure in retransmission run reveals that not all of the peers cause the spike of traffic seen at the WANemulator, but only the few that have lost the segment.

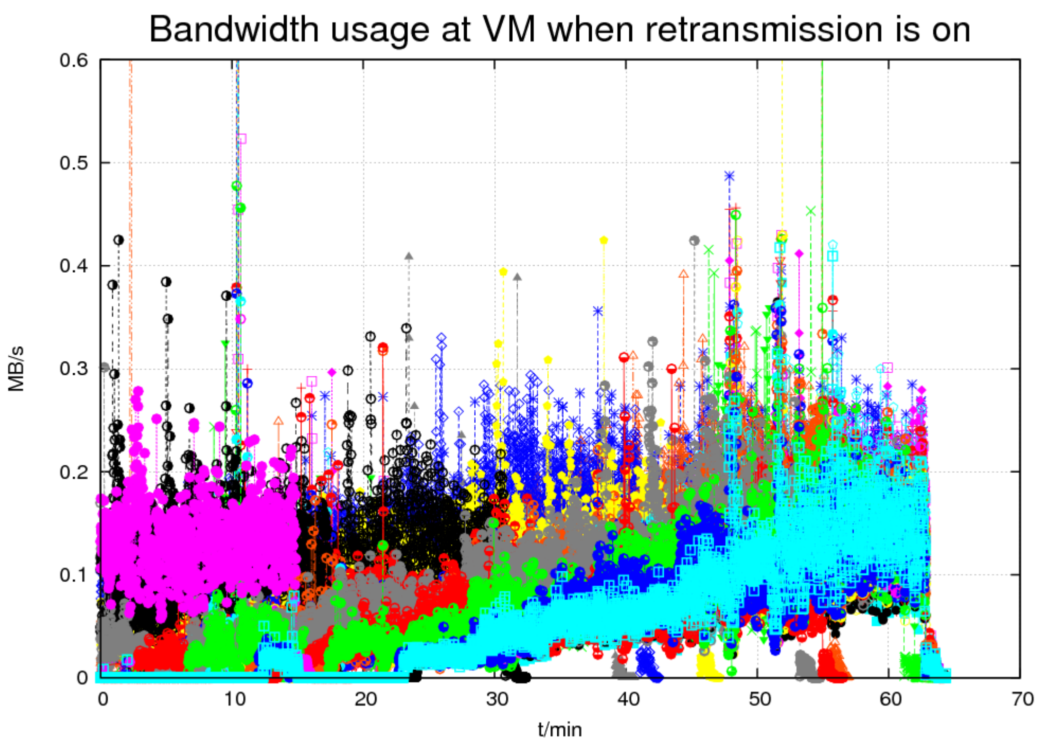


Figure 4.3: Retransmission run, virtual machine traffic zoomed in.

In Figure 4.3 the highest peaks in retransmission run are cropped, so that the trend of increasing traffic can be seen. Clearly, retransmission has caused more traffic than in the normal run, some virtual machines have even sent and received higher amount of traffic right from the beginning. This is interesting, as then they do not have that many peers running, which normally causes the increasing trend of traffic. From the figure it can also be seen that some of the virtual machines have ended their transmissions before the streamed content has finished.

Figure 4.4 proves that the increased traffic seen in Figure 4.2 is caused by joining peers. In the figures on the left retransmission is not used, while in the figures on the right retransmission is enabled. Data for Figure 4.4 is gathered from the same test runs than for Figure 4.2. The data on the top row in Figure 4.4 is gathered from SDS database, and it reveals that the total traffic rate on one peer is on average 25 KBps in both cases. These values remain quite same, recardless of the size of the overlay. This could be verified by browsing through some older test runs performed before the WANemulator was used.

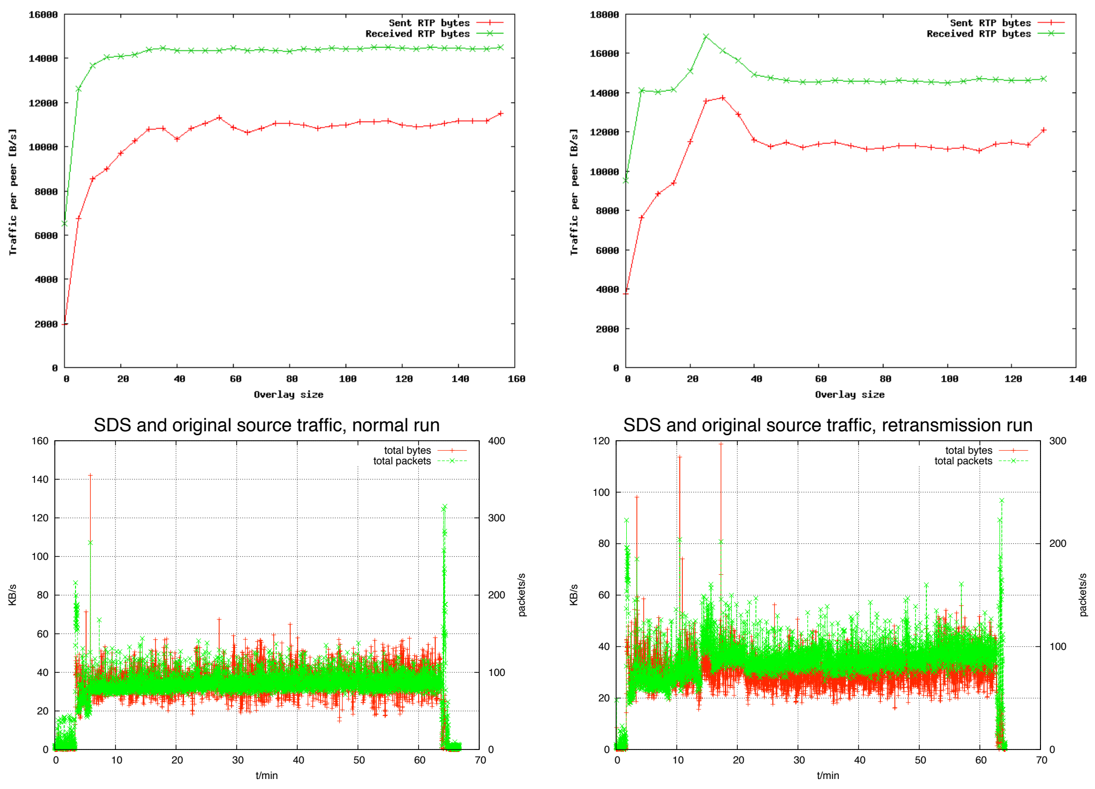


Figure 4.4: Traffic at one peer.

The bottom row figures in Figure 4.4 are received with Bwm-ng, but this time also SDS traffic is summed up with the original data source traffic in the results. Therefore, the traffic rate seems to be somewhat higher on average than 25 KBps. The peaks in the packet rate at the end of the stream in both runs can be explained by simultaneous peer communication to SDS. Nevertheless, it can be stated that the average traffic one peer produces stays rather constant.

When the average rate of 25 KBps is multiplied with 160 peers, total traffic rate of 4 MBps is received. As this total traffic passes through the WANemulator's interface twice, due to the architecture of the network environment, the total traffic should be on average 8 MBps.

Figure 4.2 reveals that only the highest spikes reach that high, while the average seems to be closer to 6 MBps at the end of the run, when all the peers have joined into the overlay. Not all of the peers are transmitting or receiving packets at the same time, as can be seen from the figures represented above. The traffic at one peer varies over time, but in the case where a segment in the stream is lost, high traffic spikes can occur at the WANemulator.

The randomness of P2P environment in this matter complicates the tests.

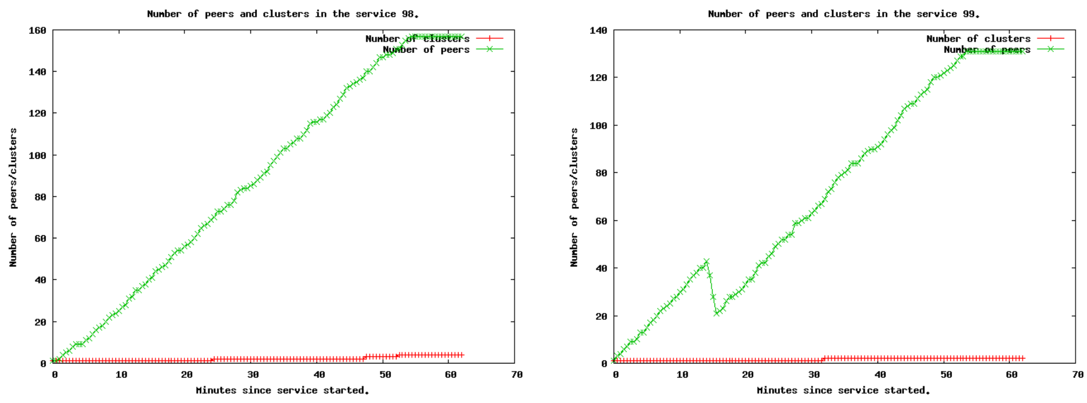


Figure 4.5: Peer joins to the overlay.

Figure 4.5 shows the growing trend of peers in the overlay at the test runs discussed above. Service 98 is not using retransmission while service 99 is. The services are the same than were seen in Figures 4.2 and 4.4. In service 99, a glitch in peer overlay joins can be seen after 15 minutes. This glitch can also be seen in Figure 4.2.

RTP2P has recovered from the situation, but some peers have left the overlay. Due to this, service 99's total peer count is lower than in service 98, meaning less than 140 peers when there should be 160 peers at the end of the test run. This was seen also in Figure 4.3, where some of the virtual machines had no traffic in the end of the run.

The randomness of P2P environment might cause different results in two consecutive test runs with the same parameters, so more test runs were performed. A total of six additional runs were performed, three with normal setup and three with retransmission turned on. The results are gathered in Figure 4.6. This time only the total traffic going through the WANemulator was measured.

From Figure 4.6, it can be seen that the total traffic in these runs is at the same scale than in service 98 and service 99, but this time the abrupt traffic spikes exist on the normal setup as well. There have been problems in services 92 and 94, as the traffic rate has not increased constantly. Also service 95 has had some difficulties at the time instant of 25 minutes. Irregular peer joins in services 92 and 94 can be verified from the SDS data seen in Figure 4.7.

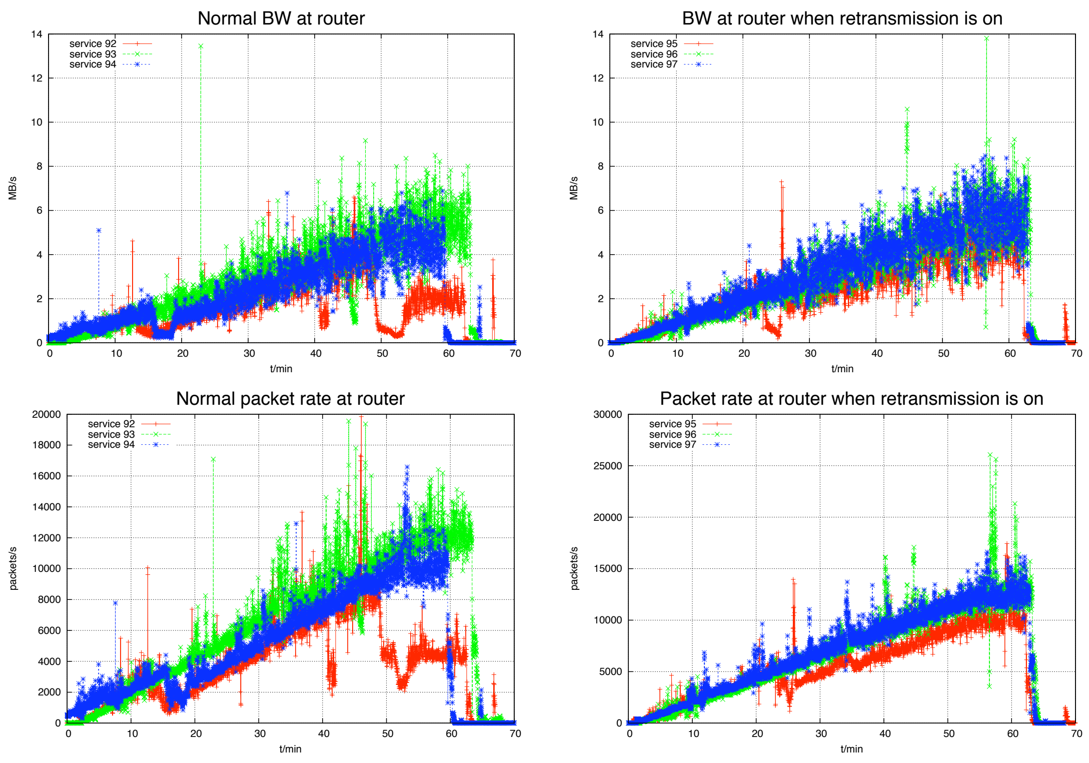


Figure 4.6: Traffic at WANemulator from multiple RTP2P runs.

Unfortunately, the behavior seen in Figure 4.7 is ordinary for RTP2P, and was existing already in the earlier test runs performed without WANemulator. This behavior is not influenced by the retransmission feature, as it happens both when it is turned on or off. The reason for peer drops from the overlay is the lack of an automatic reconnection feature in RTP2P. When a peer cannot receive the missing segments of the stream from its neighbors within configured time, it simply disconnects from the overlay. Most probably, the reason for this behavior is in strict timers used in RTP2P to achieve a small buffer for mobile devices. Therefore, retransmission does not help in the situation.

In real life, this is equivalent to the case, when the user quits the service immediately when a clearly noticeable break in the video stream occurs, and does not reconnect to the service later. Alternatively, the user could manually reconnect when streaming stops. In any case, the user should not be forced to reconnect multiple times, RTP2P should be able to handle it automatically. Adding this feature to RTP2P should be considered.

However, the lack of this reconnection feature made the validation of the network environ-

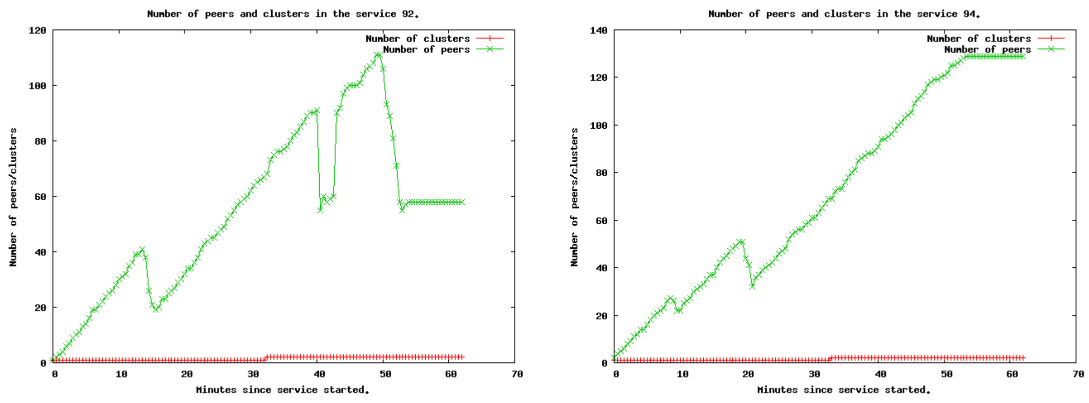


Figure 4.7: Difficulties at peer joins.

ment capabilities using RTP2P difficult. The reliability using RTP2P could not be reached, so other means had to be found to see what the environment is capable of.

4.2 Benchmarking the environment with traffic generators

Traffic generators are regularly used in scientific research when some features of a system have to be tested. They offer tunable parameters to test different kind of features in the system, and the behavior of the generators is known. Many options exist, but D-ITG [73] and Iperf [74] were seen the most suitable ones for this thesis.

The greatest benefit of D-ITG compared with other traffic generators is that it can produce variable length packet sizes using different distributions, if desired. This was the case with RTP2P, so a similar packet length distribution had to be found. UDP traffic seen in Figure 4.8 was generated using normal distribution, 242 byte mean value, and 20 byte variance. The UDP traffic that RTP2P produces is also shown as a reference.

As it can be seen, generated traffic is a close match to the real traffic. The total number of packets sent in both graphs were similar, but 100% of the generated traffic is concentrated on the range of 160–319 bytes, when only 51% of the real traffic was inside this range. This explains the higher peak of packets, seen in generated traffic graph. Still, it is a close enough approximation and thus, scalability tests were mainly performed using those values.

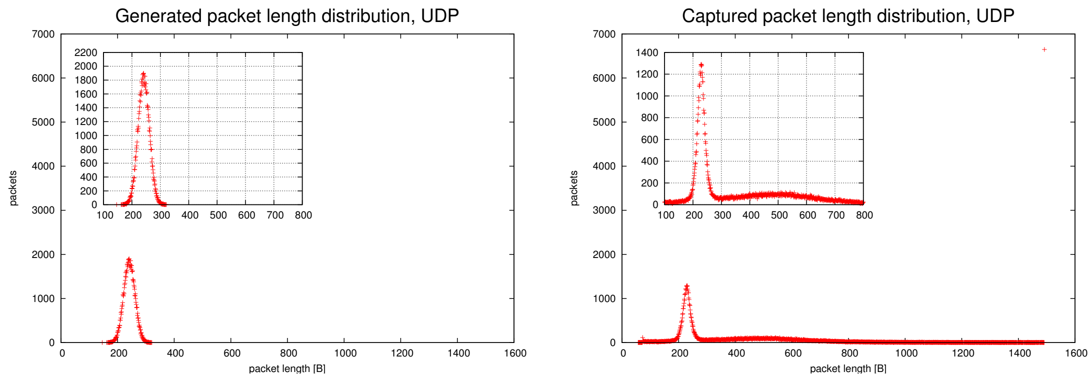


Figure 4.8: Generated versus captured UDP traffic.

However, a worst-case scenario with the Ethernet frames of 1492 bytes was also used to assure the usefulness of the testing environment. A 1492 bytes long Ethernet frame is the maximum, that RTP2P generates, as can be seen from Figure 4.8. If the environment performs steadily under 1492 byte frames, it will work reliably with RTP2P system.

When the testing progressed, some downsides from D-ITG were found. One of the biggest ones was the observation, that it uses binary log files to store the data. The size of the log grows to gigabytes in longer runs on higher bandwidths. This is not suitable with virtual machines, that have limited amount of space. Actually, even D-ITG itself could not open a file too large, which limited the length of test runs performed with D-ITG to 30 minutes.

Table 4.1: Tools used for environment testing.

Tool	**Purpose of use**
D-ITG	Traffic generator
Iperf	Traffic generator
Bwm-ng	Live bandwidth monitor
Top	Linux task and resource monitor

Therefore, Iperf was also used to make the results fool proof. Variable sized traffic cannot be generated with Iperf, but it verifies that D-ITG values are correct. With Iperf, overnight runs could be performed in order to have more reliable results. Iperf was used in full-duplex mode, where both ends send simultaneous traffic just like D-ITG does. This ensured the comparativeness of results between these two traffic generators. In addition to D-ITG and Iperf, a couple of other tools were used to verify the results. They are all summarized in Table 4.1.

Bwm-ng was used to examine the actual traffic flowing through interfaces. CPU load and memory usage was monitored using Top. The results using these two tools are not logged, but they are commented in the following sections when necessary. Both traffic generators were highly CPU intensive, as they utilized almost all the available CPU time. This corresponds closely to the operation of RTP2P, which makes the results closer to real emulation situation.

Before progressing to the tests and their results, a few remarks on the operation of D-ITG are in order. It seems to be operating in bidirectional mode, but it only reports half of the rates, that are actually flowing through an interface. The reported value is most probably average of the bidirectional results, but an explanation for the output could not be found from the D-ITG manual.

Full-duplex operation is desired as RTP2P operates with the same principle. Another thing worth noticing is, that the rate that a 1 Gbps interface can handle is 1 Gbps both ways. Hence, 2 Gbps is the total traffic, that the interfaces should be capable of handling in this environment.

The generated traffic rate in D-ITG can be tuned by -C parameter, which controls the constant inter-departure time (IDT) of the packets. For the sake of simplicity, IDT is expressed as the packet rate, which is then converted to time interval value with Formula (1).

$$IDT(ms) = \frac{1000}{packet\_rate} \tag{1}$$

D-ITG also adds 42 bytes of headers to the packet size expressed with either -c or -n parameter. The parameters correspond to either constant (-c) or normal distributed (-n) packet sizes. Thus, a configuration of -c 1450 produces the wanted 1492 byte constant packet size, while -n 200 20 gives the packet sizes seen in Figure 4.8. No fragmentations of packets existed in the traffic, as the packet sizes in all runs were lower than the Ethernet Maximum Transmission Unit (MTU) of 1500 bytes. Ethernet jumbo frames were not used.

With more complicated configurations D-ITG could have been used to generate some bursts of traffic in random intervals like RTP2P does, but these tests were deliberately kept simple and bursts were not modeled. The main idea of this chapter is to find the performance limits of the environment, while Chapter 5 will concentrate on testing with RTP2P application.

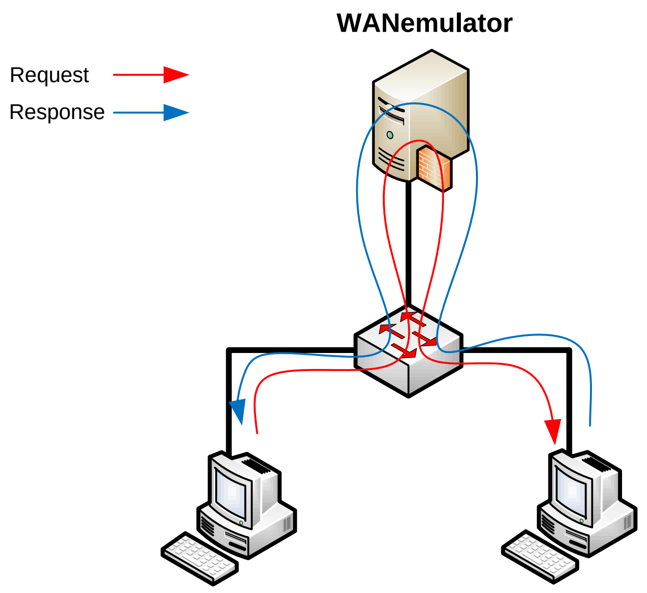


Figure 4.9: Full-duplex traffic flow using D-ITG.

In Figure 4.9, another issue that was discovered during the tests can be seen, namely the amount of traffic going through the interface in the WANemulator. As it only uses one interface for both Receiving (Rx) and Transmitting (Tx) traffic on a normal full-duplex mode, the traffic D-ITG generates actually passes the interface two times for one way. For some reason D-ITG sends traffic simultaneously from both ends, but it only reports one way traffic. Therefore, the traffic rate D-ITG outputs has to be multiplied by four to get the actual amount of traffic at the WANemulator.

In other words, in the best possible scenario, the WANemulator could handle 2 Gbps of total traffic, as it has one full-duplex 1 Gbps interface in use. When only two PCs are sending traffic through the WANemulator, their total traffic cannot exceed this 2 Gbps limit, even though their own interfaces might be able to deliver more. Hence, maximum Rx and Tx rate at the interfaces of those two PCs is 500 Mbps, half the amount they are capable of. As the amount of PCs increase, they still cannot exceed the 2 Gbps limit at the WANemulator, meaning the PCs can use even lower proportions of their available bandwidth. Thus, the link between a switch and the WANemulator might become a bottleneck.

In the following sections, limits of the testing environment are found in incremental manner like Floyd and Paxson [7] recommend. The tests performed can generally be divided into two categories. First, in Sections 4.2.1 through 4.2.4 VirtualBox VM's capabilities are located using only Layer 2 switching. Then, in Sections 4.2.5 and 4.2.6 WANemulator is added to the network and Layer 3 routing is used to find the limits of the environment in general.

First, a rough estimation of the limits is located using a few 30 second test runs. The results shown in the tables are average values of multiple consecutive runs. Then, in the latter part with routing enabled, both D-ITG and Iperf are used to validate the environment performance. These longer runs are performed only on the latter part, as it corresponds more closely to the real situation with RTP2P testing.

The reader should also pay attention to the units used. The figures seen earlier in this chapter used bytes, when the tests in the following sections are performed using bits. Traffic generators understand bits better, therefore this change was done. To convert the bps values into Bps, the results just have to be divided with eight as there are 8 bits in a byte.

4.2.1 PC-to-PC through switch

In the first scenario only physical PCs were used. No duplicate routing table entries were used, so the traffic was passing only through the switch and never reached the router. Therefore, both ends had 1 Gbps of full-duplex bandwidth available. The scenario is expressed in Figure 4.10.

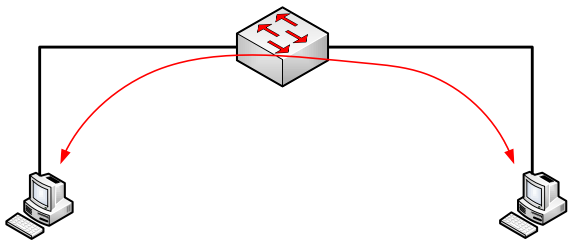


Figure 4.10: PC-to-PC through a switch.

The results from the runs can be seen in Table 4.2. The first column indicates how many times the test was run, packet size tells what was the configured packet size, and IDT was used to control the sending rate with D-ITG. The rest of the columns state the results.

Table 4.2: Results from the PC-to-PC run.

Avg of x runs	Packet size (bytes)	IDT	kbit/s	packets/s	packet loss	avg delay (ms)	avg jitter (ms)	deviation (ms)
9	-c 1450	-C 100000	809892	69818	0,83%	1,13	0,01	2,66
8	-c 1450	-C 60000	691824	59640	0,33%	0,60	0,02	1,68
3	-c 1450	-C 30000	47847	29978	0,00%	0,11	0,02	0,34
8	-n 200 20	-C 100000	144785	90740	0,63%	0,95	0,01	2,12
3	-n 200 20	-C 80000	127618	79961	0,05%	0,30	0,01	0,73
4	1450	1000m	912451	IPERF	1,34%	IPERF	0,04	IPERF
4	200	1000m	159167	IPERF	2,58%	IPERF	0,01	IPERF

The last two rows show the results received with Iperf. As Iperf uses different parameters than D-ITG, some of the results were not available. With Iperf also the UDP buffer size setting is limited to maximum 256 KB, which might have some effect on the results. The requested traffic rate in both of the Iperf runs was full-duplex 1000 Mbps, but the packet sizes between the two runs were different.

A successful test can be identified with having similar value in IDT column, and either in packets/s column in a case of D-ITG, or kbit/s column in a case of Iperf. Also the packet loss and delay values should be small. The results indicate that physical network interfaces can perform pretty close to the 1 Gbps limit with low losses and delays. The results received are full-duplex, so received transfer rates are actually going both upstream and downstream at PCs at the same time.

The limiting factor seemed to be the achievable packet rate. Neither with constant size 1450 byte or variable sized packets, the requested rate of 100 000 packets per second (pkt/s) was not reached. With iperf this rate could not be calculated as Iperf does not offer statistics of the packet rate. However, the traffic rate indicates, that also the packet rate is quite similar with Iperf than with D-ITG.

Packet losses are higher in results received with Iperf, but this is due to a couple of peaks in one or two runs. As Iperf tests were not run often enough, there is no reason to conclude much from them. In fact, Iperf was used in this test only to confirm the throughput magnitude, which proved to be similar than with D-ITG. When the requested rate using D-ITG was set at a lower rate than what was achieved with 100 000 pkt/s, packet loss and delay values became steady and tolerable.

The test was performed also with simultaneous traffic from two PCs to other two PCs. The results from these runs were of the same magnitude as with only one data flow through the switch, which proves that the switch can handle the traffic. The limits of the switch were not investigated further, as it will not become a bottleneck in the setup with WANemulator.

In brief, the switch does not limit the operation of physical interfaces, the limits of PC interfaces are reached first. The maximum reliable packet rate is 60 000 pkt/s with 1450 byte packets and 80 000 pkt/s with variable sized packets. These correspond to traffic rates of 692 Mbps and 128 Mbps respectively. In bytes, the rates are 86,5 MBps and 16 MBps, which far exceeds the values RTP2P requires.

It is noticeable, that with Iperf same traffic rates cannot be used with 1450 and 200 byte packets. If Iperf is set to transmit 200 byte UDP packets with 692 Mbps, the packet rate increases so high, that the interfaces on the PCs will get completely clogged.

4.2.2 VM-to-VM through switch

When the physical interfaces of PCs were found out capable of the RTP2P traffic, the next step was to find the overhead caused by virtualization. The setup is similar to the one presented in Section 4.2.1, but now with one VM running on a PC in both ends. The setup is shown in Figure 4.11.

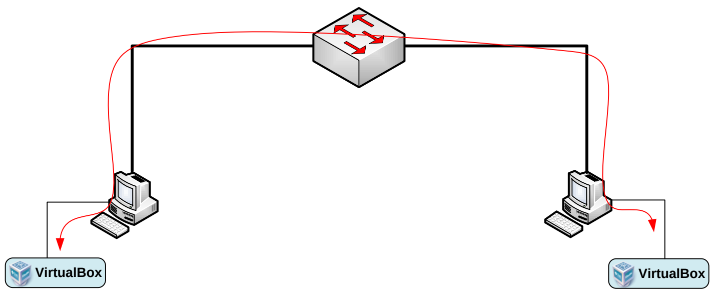


Figure 4.11: VM-to-VM through a switch.

The results seen in Table 4.3 were quite surprising. The results are over ten times lower than without virtualization, which cannot be explained even with unreasonably high virtualization overhead. One possible explanation might be that the virtualized interface cannot reach 1 Gbps speeds, but is only capable of 100 Mbps. This is against the specification of VirtualBox, which states that the interface used is capable of 1 Gbps. Even changing the virtualized interface to other than Intel PRO/1000, did not help.

Table 4.3: Results from the VM-to-VM run.

Avg of x runs	Packet size (bytes)	IDT	kbit/s	packets/s	packet loss	avg delay (ms)	avg jitter (ms)	deviation (ms)
4	-c 1450	-C 100000	79892	6887	1,27%	9,14	0,07	1,67
3	-c 1450	-C 6000	69553	5996	0,07%	1,01	0,10	1,16
3	-n 200 20	-C 30000	10889	6823	0,69%	18,19	0,06	3,07
3	-n 200 20	-C 6000	9573	5998	0,03%	0,76	0,08	1,26
4	1450	95m	67806	IPERF	1,08%	IPERF	0,10	IPERF
3	1450	47m	41141	IPERF	0,02%	IPERF	0,11	IPERF

Another notice concerns Iperf results. As with the case with two PCs communicating through a switch in Section 4.2.1, the results with VMs running on PCs could not reach the requested traffic rate. When the configured rate was raised to arbitrary high levels, the rate of 100 Mbps and over could be reached, but only at the expense of higher packet loss rate. This tells that the VM interfaces are not limited to 100 Mbps, as higher rates are achievable.

When investigated further with the help of the Bwm-ng tool, the reason behind this awkward behavior of Iperf was identified to be due to an unstable packet rate. Another possible reason might be the limited UDP buffer size. Nevertheless, Iperf could not reach rates similar in magnitude than in Section 4.2.1, when virtualization was not used. This confirms the D-ITG results, so some strange limitations in VirtualBox network interfaces must exist.

The unexpected results with virtual machines can be summarized to maximum packet rates of only 6000 pkt/s. With 1450 byte packets, this equals to 70 Mbps and with variable sized packets the traffic rate is 10 Mbps. The rates in bytes are 8,75 MBps and 1,25 MBps. It should be sufficient for RTP2P if the amount of peers at one VM is kept low enough. To conclude, the results received with these lower rates are reliable and stable.

4.2.3 VM-to-VM through virtual bridge at PC

The mysterious behavior of virtualized interfaces had to be further investigated. Now only the performance of the virtualized bridge was tested, so only two virtual machines were run on one PC. The setup is illustrated in Figure 4.12, and the results of the tests are shown in Table 4.4. Bwm-ng was used to investigate, that there was no traffic at the interface on the physical PC where the VMs were operating. It means that the virtual bridge operates correctly, and the traffic is flowing only inside VirtualBox.

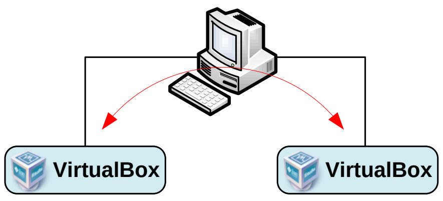


Figure 4.12: VM-to-VM using only a virtual bridge at PC.

The results indicate that the virtual bridge limits the transfer speeds to approximately 5000 pkt/s at one VM. This approximates to 58 Mbps with 1450 byte packets and 8 Mbps with variable packet size. Higher rates cause a significant increase in packet losses and delays.

Table 4.4: Results from the run where VMs communicate only through a virtual bridge.

Avg of x runs	Packet size (bytes)	IDT	kbit/s	packets/s	packet loss	avg delay (ms)	avg jitter (ms)	deviation (ms)
4	-c 1450	-C 10000	62213	5363	3,51%	11,77	0,14	1,80
6	-c 1450	-C 5000	57982	4998	0,03%	1,06	0,09	1,90
4	-n 200 20	-C 10000	9429	5909	0,75%	21,32	0,13	4,93
4	-n 200 20	-C 5000	7979	4999	0,00%	0,68	0,08	1,43
4	1450	116m	43676	IPERF	0,13%	IPERF	0,15	IPERF
3	1450	58m	40949	IPERF	0,04%	IPERF	0,32	IPERF

Lower results compared with the case in Section 4.2.2 can be explained by the capabilities of the bridge. When only one VM is running on a PC as in Section 4.2.2, the actual traffic going through the bridge is double the rate indicated in Table 4.3, due to full-duplex principle. In the case when two VMs are running simultaneously on a host, the actual rate the virtual bridge has to handle is four times the ones reported in Table 4.4, Rx and Tx to both ways. This might seem odd at first, as Tx from one VM should be Rx to the another VM. Nevertheless, by using Bwm-ng it could be verified that in the interfaces of the VMs was running the amount of traffic as indicated in here.

The reasons behind this operation remain ambiguous without deeper knowledge about the architecture of the virtual bridge in VirtualBox. Iperf results had the same issue as in Section 4.2.2, the requested values could not be reached. This indicates issues in VirtualBox network interface implementation, as in general, Iperf operates reliably. With Bwm-ng it was proved that Iperf actually does send the traffic it reports.

4.2.4 Multiple VMs through switch

The next step was to evaluate the performance when there are two VMs running on one host. In Section 4.2.3 two VMs on one host were communicating only through the local virtual bridge, but this time they are communicating through a switch to a VM located at different physical PC. The scenario can be seen from Figure 4.13. Now it has to be kept in mind, that there exists the limit of 5000 pkt/s for one VM because of the virtual bridge.

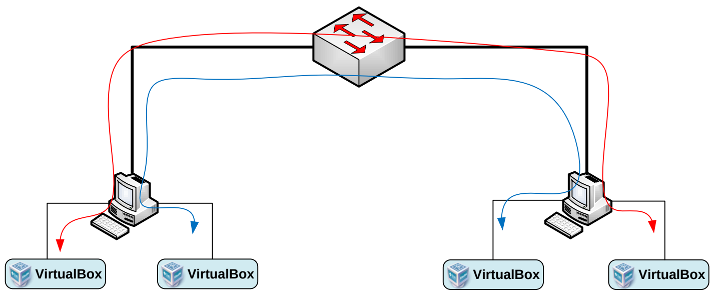


Figure 4.13: Two simultaneous VM-to-VM connections through a switch.

The results shown in Table 4.5 are in line to the ones received in Section 4.2.3. As there are now also two VMs running simultaneously on one host, packet rates over 5000 pkt/s increase the packet loss and delay to an intolerable level. When the packet rate was dropped to 4000 pkt/s, the results remained within acceptable limits. Therefore, the maximum packet rate of 4000 pkt/s for one VM can be concluded as a reliable limit with two VMs on one PC. Iperf still had the same issue as earlier, it could not reach the configured rate. Reason for this can be most probably found at the virtual bridge implementation.

Table 4.5: Results when two VMs are running simultaneously on a host.

Avg of x runs	Packet size (bytes)	IDT	kbit/s	packets/s	packet loss	avg delay (ms)	avg jitter (ms)	deviation (ms)
6	-c 1450	-C 6000	67422	5812	1,32%	6,65	0,11	3,68
3	-c 1450	-C 4000	46391	3999	0,05%	0,85	0,11	1,78
3	-c 1450	-C 3000	34807	3001	0,01%	1,08	0,16	2,08
5	-n 200 20	-C 6000	9498	5952	0,53%	9,36	0,36	8,68
3	-n 200 20	-C 4000	6386	4000	0,01%	1,16	0,14	3,86
3	-n 200 20	-C 3000	4789	3000	0,01%	2,97	0,16	7,82
3	1450	47m	39791	IPERF	0,10%	IPERF	0,29	IPERF
4	200	7m	5741	IPERF	0,03%	IPERF	0,19	IPERF

A scenario, where more than two VMs are running simultaneously on one host is expressed in Figure 4.14. Before advancing to the results received with three VMs, a few remarks of the CPU usage with VirtualBox are in order.

As mentioned earlier, both RTP2P and traffic generators are highly CPU intensive, they use all the available CPU time. The host computers have one CPU with two cores. When one virtual machine instance acts like any other process on the host, it is normally only capable of reserving one core at a time. This has been the case in earlier tests where only one VM is running, leaving half of the CPU idle.

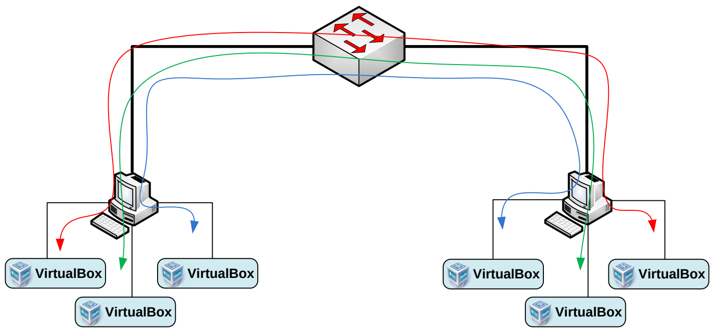


Figure 4.14: Three simultaneous VM-to-VM connections through a switch.

When two VMs are running on one host simultaneously, the CPU is fully loaded. This is expressed in Table 4.6. The results are captured during an Iperf run, and load averages are for the past 1, 5 and 15 minutes. Dual-core CPU is capable of load averages of 2.00 without the need to delay the processes, 1.00 for each core. As the results show, the CPU has been overloaded with a small degree already with only two VMs running. When the CPU is overloaded, it just delays the processes it cannot handle in time.

Table 4.6: CPU load with two VMs running simultaneously.

```
## PHYSICAL HOSTS UPTIME ##
oplab-11.rtp2p.local
 05:33:59 up 1 day, 14:00,  2 users,  load average: 2.18, 2.12, 2.13
oplab-12.rtp2p.local
 05:39:11 up 1 day, 14:00,  0 users,  load average: 2.18, 2.19, 2.18

## VMs UPTIME ##
oplab-111.rtp2p.local
 07:33:52 up 18:59,  0 users,  load average: 2.09, 2.13, 2.09
oplab-112.rtp2p.local
 07:34:19 up 18:58,  0 users,  load average: 2.14, 2.12, 2.09
oplab-121.rtp2p.local
 07:40:38 up 19:09,  0 users,  load average: 2.24, 2.20, 2.16
oplab-122.rtp2p.local
 07:39:22 up 19:42,  0 users,  load average: 2.19, 2.19, 2.21
```

An overloaded situation is not the most desirable in the long run. The host computer still should handle it, but the processes decelerate, as mentioned above. However, if the CPU is heavily overloaded, strange things might happen, so it should be avoided. Even more serious situation would be if the memory runs out, but luckily neither RTP2P nor traffic generators exhaust the available RAM. There is plenty of free RAM available, even in a full scale test with RTP2P.

The average results with three VMs running simultaneously at one PC were not that different from the ones when only two VMs were running. This was the case if the 5000 pkt/s limit of the virtual bridge was respected. If that limit was exceeded, the results were inferior in comparison with the case when only two VMs were running.

Nevertheless, fairness among VMs became an issue with more than two VMs. When investigating the CPU and memory usage with Top on the host computer, it shows that the CPU is overloaded. The transfer rate one VM achieves depends heavily on the CPU time it receives. Three or more VMs simply cannot use two CPU cores fairly, which makes setup of over two simultaneous VMs at one PC unsuitable for RTP2P testing.

To conclude, with too many VMs the CPU power required will become an issue. Over two VMs cannot be ran on dual-core CPU reliably. Also the limits of the virtual bridge have to be respected. To ensure reliable operability with two VMs, a packet rate of 4000 pkt/s per VM should not be exceeded. This corresponds speeds of 47 Mbps and 7 Mbps respectively. In bytes the values are 5,88 MBps and 0,88 MBps.

Even though the rates were lower than in Section 4.2.2, because of the safety margin, they are still good enough for RTP2P. Figure 4.2 reveals, that only the highest peaks in a RTP2P test run where five peers were running on one VM exceed this limit. The average packet rate at one VM is less than 1000 pkt/s.

4.2.5 PC-to-PC through WANemulator

Now that the limits of the virtual machines are found, next step is to see what WANemulator is capable of. As with Layer 2 tests, explained in Sections 4.2.1 through 4.2.4, let us start with the physical machines first. The setup can be seen in Figure 4.15. In the setup two PCs are communicating through the WANemulator, which is not performing any emulation. For testing purposes also the host computers had modified routing tables to force the traffic through the WANemulator. Normally they did not have the extra rule installed.

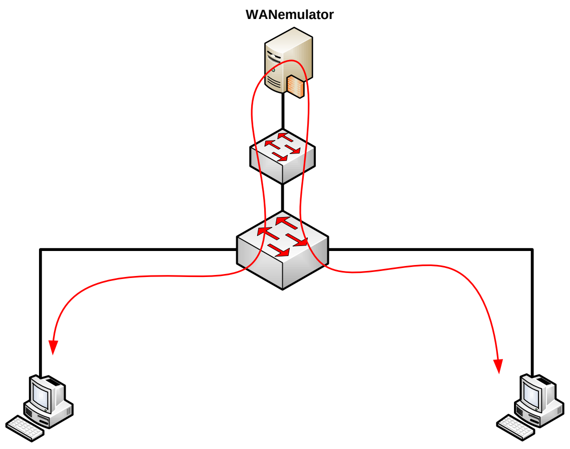


Figure 4.15: PCs routed through the WANemulator.

Table 4.7 reveals, that maximum achievable packet rate is around 30 000 pkt/s, even though the packet loss with that rate is too high. It has to be remembered, that the traffic rate D-ITG outputs corresponds to four times the traffic at the interface on WANemulator. Therefore, 327 Mbps D-ITG reports means 1,3 Gbps at the WANemulator. At first, this seemed odd, as the WANemulator's interfaces should be capable of 2 Gbps.

When the issue was investigated further, the reason was identified as overfull buffers in the WANemulator. This is a normal behavior in routers. When the packet queues are full, incoming packets are dropped which increases the packet loss. The length of the queue can be increased so that packet loss does not occur, but the delay of the packets increases. The size of the queue is a tradeoff between packet loss and delay.

When the packet rate was lowered, at around 10 000 pkt/s, tolerable results were achieved. This corresponds to 40 000 pkt/s at the router interface, as explained earlier. Figure 4.2 revealed, that in the worst case scenario WANemulator must handle a maximum of 20 000 pkt/s, which in turn corresponds to 5000 pkt/s of single full-duplex traffic flow. It seems, that WANemulator can reliably operate at double the rate, that is required with RTP2P using 160 peers.

The rest of the tests concentrated on validating the results with the required 5000 pkt/s rate. Running D-ITG with the packet rate of 5000 pkt/s revealed, that in long runs the packet loss

Table 4.7: Results from the physical PCs with routing turned on.

Avg of x runs	Packet size (bytes)	IDT	kbit/s	packets/s	packet loss	avg delay (ms)	avg jitter (ms)	deviation (ms)
3	-c 1450	40000	321339	27702	30,49%	38,81	0,06	11,79
3	-n 200 20	40000	53932	33787	15,43%	7,50	0,04	12,94
3	-c 1450	30000	327426	28226	5,87%	2,26	0,05	3,91
3	-n 200 20	30000	43528	27273	8,86%	4,42	0,04	10,79
4	-c 1450	20000	226755	19548	2,20%	0,92	0,03	3,13
7	-n 200 20	20000	31226	19566	2,00%	0,74	0,04	3,23
3	-c 1450	10000	115624	9968	0,29%	0,63	0,05	0,52
6	-n 200 20	10000	15867	9940	0,57%	0,53	0,06	2,18
4	-c 1450	5000	57888	4990	0,17%	0,61	0,03	1,15
6	-n 200 20	5000	7962	4989	0,17%	0,47	0,03	1,60
30min	-c 1450	5000	57653	4970	0,60%	0,76	0,04	3,57
9h	1450	58m	57586	IPERF	0,71%	IPERF	0,02	IPERF
30min	-n 200 20	5000	7915	4959	0,81%	0,67	0,04	4,53
9h	200	8m	7961	IPERF	0,49%	IPERF	0,01	IPERF

and delay increase slightly. The same effect is seen with nine hour Iperf run in Table 4.7, but the results still remain at tolerable level. Notice the similarity with Iperf and D-ITG run traffic rates. It seems, that in long runs the problems seen earlier are evened out and also Iperf can achieve the requested speeds.

To conclude, WANemulator can handle the traffic RTP2P generates with 160 peers. Short test runs indicate, that WANemulator might be reliable with double the rate, but this was not proved with longer test runs. When the peer amount and traffic rate through the WANemulator interface increases, the packet loss rate presumably will increase, even without any artificial delay added. The main reason for this is the length of the queue in the WANemulator, which can be modified to enhance the results.

4.2.6 Multiple VMs through WANemulator

Only the situation most closely resembling the RTP2P case is left. The setup differs from the one seen in Section 4.2.5 so that now there are two virtual machines running on each PC. The setup is similar than was seen in Section 4.2.4, but now the traffic is routed through the WANemulator. Figure 4.16 illustrates the scenario.

WANemulator is not performing emulation, it just acts as a router. The main idea of this section is to prove the operability of two VMs at one PC in routed situation, not to build a full scale RTP2P network. The traffic rate limits WANemulator can handle are already proved in Section 4.2.5, now the traffic limits of the testing environment in general are

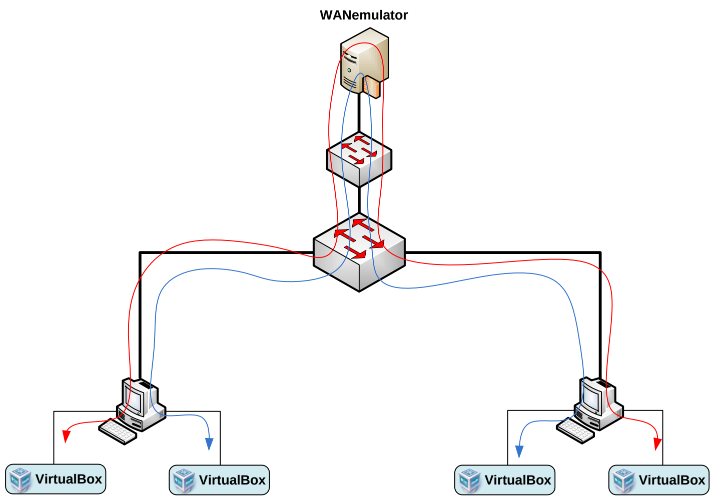


Figure 4.16: Multiple VMs routed through the WANemulator.

ensured. Emulated scenario is investigated in Chapter 5.

A short summary of the results received so far, follows. Two VMs running on the same host can reach 4000 pkt/s maximum each, which totals 8000 pkt/s at one host PC. WANemulator can handle only 5000 pkt/s, so the rate one VM generates has to be decreased. To play safe, the total traffic was limited to 2000 pkt/s at one VM. This corresponds to 4000 pkt/s from one physical host, ending up to 16 000 pkt/s going through the WANemulator's interface. This reflects quite well the traffic the RTP2P generates, as can be seen from Figure 4.2. The results with 2000 pkt/s rate are summarized in Table 4.8.

Table 4.8: Results of multiple VMs with routing turned on.

Avg of x runs	Packet size (bytes)	IDT	kbit/s	packets/s	packet loss	avg delay (ms)	avg jitter (ms)	deviation (ms)
6	-c 1450	2000	23153	1996	0,22%	1,18	0,12	1,87
7	-n 200 20	2000	3182	1994	0,31%	1,05	0,12	2,73
30min	-c 1450	2000	23113	1992	0,38%	1,29	0,22	3,46
9h	1450	24m	23229	IPERF	0,40%	IPERF	0,15	IPERF
30min	-n 200 20	2000	3184	1995	0,25%	1,02	0,12	2,89
9h	200	4m	3810	IPERF	0,40%	IPERF	0,10	IPERF

A noticeable issue not seen in the average results shown in Table 4.8, is the fairness of the results with 2000 pkt/s. All the VMs received the same proportion of the bandwidth, also packet loss, delay, and jitter were the same regardless of which VM was investigated.

As the traffic rate was lowered from total 5000 pkt/s to 4000 pkt/s, packet loss dropped to

0,4% in both 1450 byte and variable packet length tests. Jitter increased slightly compared to a situation without virtualization, but it is still at low level. Iperf results correspond closely to the ones received with D-ITG, proving the fact, that the network environment is reliable and provides consistent results at the rates used.

4.3 Summary

In this chapter the limits of the testing environment were located. On one physical host only two virtual machines can be running simultaneously, as there is not enough CPU time for more VMs. Available memory does not get exhausted, but the lack of CPU power when more than two VMs are running, cause unfair results in adjacent VMs. When two VMs are running simultaneously at one physical host, one VM cannot exceed the packet rates of 4000 pkt/s due to a feature in VirtualBox's virtual bridging and virtualized interfaces.

This inferior operation of VirtualBox network interface, remained a mystery. In [62] VirtualBox was found capable of the rate the physical interface offers, but they had used some special network driver implementation, enabling a more closer communication with the physical network interface. In general, virtualization overhead in CPU intensive processes, like the ones performed in this chapter, should be at most 10,6% [75]. However, tests in [75] were performed with VMware and Xen, so the results received here might just be VirtualBox features.

Even though there were issues in the VirtualBox networking, it was not the biggest issue. The maximum packet rate of 4000 pkt/s at one VM cannot be used if all the 17 PCs in the laboratory are running, as the WANemulator can only handle 10 000 pkt/s coming in and 10 000 pkt/s going out. WANemulator was tested reliable with these rates, but it might be capable of handling traffic double than that. It would just require tuning the packet buffer sizes at WANemulator. The centralized architecture where a single emulation node is used, introduced an artificial bottleneck in the P2P network.

In the short test runs, there were some inconsistencies with Iperf, but they evened out in the overnight tests. These long test runs proved the environment reliable, when traffic rates shown in Table 4.9 were not exceeded. When the size of the RTP2P overlay is limited to

Table 4.9: The maximum traffic the environment can handle reliably.

	Virtual Machine	**WANemulator**
Packet rate	4000 pkt/s	20 000 pkt/s
Traffic rate, 1450B	47 Mbps	232 Mbps
Traffic rate, 200B	7 Mbps	32 Mbps

160 peers the environment is reliable. The environment should handle 320 peers reliably as well, but this was not proved. Also, if the streamed content would be encoded differently, the traffic rate peers generate would change.

Even more peers than that might be possible, as the results received here do not fully correspond to a RTP2P run. Traffic generators send packets at constant rate when traffic with RTP2P is burstier. The packet queues will certainly overflow with constant packet rate causing higher packet losses, but with RTP2P the queues are not full all the time. This leads to the lower probability of packet losses, and therefore, the environment is capable of more peers. Still, bursty traffic with RTP2P also tends to fill buffers occasionally, which has to be taken into account.

5 RTP2P PERFORMANCE TESTING

Now that the limits of the testing environment have been found, it is time to start performing actual traffic shaping using NetEm on WANemulator. Before that, however, a short introduction to Linux traffic control is in order. After that NetEm queuing discipline is presented. Then the network metrics used in testing are explained, and finally in Section 5.4, the RTP2P tests are done. A short summary concludes this chapter.

5.1 Linux traffic control

The path of a packet when it is exiting Linux PC is shown in Figure 5.1. The packet comes from TCP/IP stack, then it is classified by a Tc classifier to a queuing discipline (qdisc). Default qdisc is a FIFO queue, whose length is 1000 packets [60].

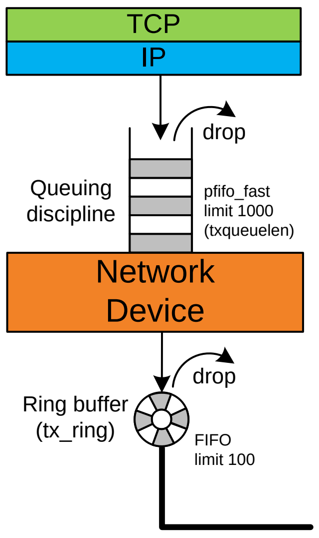


Figure 5.1: The default setup for egress traffic in Linux.

The physical network interface also has its own FIFO buffer for outgoing packets, which is normally a ring buffer. Depending on the network driver, its queue length might be configurable. Unfortunately, in our setup WANemulator is running on VMware and it uses vmxnet network driver. The driver does not allow changing the ring buffer value, it is fixed to a default 100 packets [76]. If the queues fill up, arriving packets might get dropped.

Only egress queues are shown in Figure 5.1, as traffic shaping in Linux can be only done on traffic exiting the interface. Shaping means controlling the transmission rate. Another action only occurring on egress traffic is scheduling, which is also known as the prioritization of packets. Both are performed with the help of different kinds of qdiscs.

Traffic policing can be applied on ingress traffic, that is traffic entering the interface. In general, only packet dropping or marking are possible for ingress traffic. Both of them can be performed on egress traffic, too. Intermediate Queuing Devices (IMQ) [54] enable shaping and scheduling for ingress traffic as well, but they are not considered here as they cause extra overhead and higher CPU usage at PC.

Table 5.1: Tc queuing disciplines.

Classless qdiscs	
[p|b]fifo	First In, First Out, packet or byte
pfifo_fast	FIFO offering three-band queue, default qdisc
RED	Random Early Detection
SFQ	Stochastic Fairness Queuing
Classful qdiscs	
CBQ	Class Based Queuing
HTB	Hierarchical Token Bucket
TBF	Token Bucket Filter
PRIO	Classes of different priorities
NetEm	Network emulation

Qdiscs can be classful or classless. When a qdisc is *classful*, it can contain other qdiscs. In contrast, a *classless qdisc* can only be attached to the root of the device, and other qdiscs cannot be attached to a classless qdisc. The root can have only one qdisc attached at a time, but if the qdisc attached to the root is classful, more qdiscs can be attached to this qdisc. Different qdiscs available with Tc are summarized in Table 5.1. By default Linux uses pfifo_fast. Classful qdiscs use filters to classify traffic into their subclasses. Filters reside within qdiscs, so they do not perform the shaping, they only direct traffic based on user specified rules.

Using classful qdiscs a qdisc tree can be built. Different types of packets can be classified into branches using filters, so that each branch performs a different kind of manipulation to the packet flow. The tree seen in Figure 5.2 is formed using commands in Table 5.2. It has three classes for traffic, traffic from network 10.0.1.0/24 goes to class 1:11, from 10.0.2.0/24 to class 1:12, and all the other traffic goes to class 1:13. Filtering can be based on other information, than the source IP of a packet, for example UDP traffic can be directed to one branch and TCP traffic to another. For this thesis filtering by using IP addresses was sufficient, even though UDP filtering was also successfully tested.

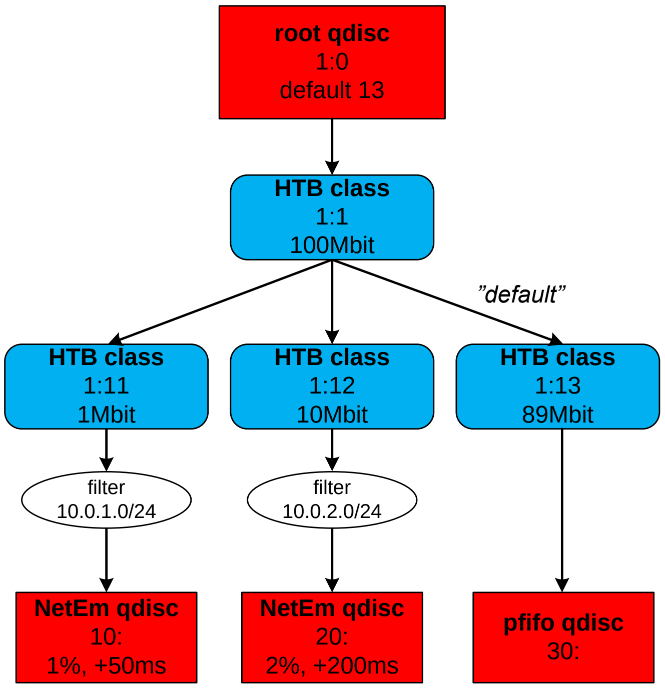


Figure 5.2: A qdisc tree formed by using classful qdiscs.

The qdiscs under classes 1:11 and 1:12 are NetEm qdiscs, which perform network emulation only to the packets from the specific networks. The default class, 1:13, does not perform emulation, as it gets pfifo qdisc attached to it by default. Traffic from 10.0.1.0/24 is throttled to 1 Mbps, delayed 50 ms, and experiences 1% packet loss. Packets from 10.0.2.0/24 have a higher bandwidth limit, 10 Mbps, and the traffic suffers 2% packet loss with 200 ms delay.

Shaping in Hierarchical Token Bucket (HTB) occurs only in leaf classes, inner and parent classes focus on performing a token borrowing mechanism. In Figure 5.2, class 1:1 is a parent as it has children, and classes 1:11, 1:12, and 1:13 are leaves as they do not have any HTB children classes.

The class and qdisc configuration can be seen from Table 5.2. Class 1:11 is limited to 1 Mbps and class 1:12 to 10 Mbps. The default class 1:13, uses the rest of the 100 Mbps rate the parent class 1:1 has. However, not all of the leaves operate constantly at this maximum rate, so there is free bandwidth available at parent class 1:1. Free bandwidth means free tokens at class 1:1. If a class needs more bandwidth than its rate limit allows, it can borrow tokens from its parent, up to its ceil rate.

Table 5.2: Tc commands used to build a tree.

```
## ROOT QDISC ##
tc qdisc add dev eth0 root handle 1:0 htb default 13

## MAIN CLASS ##
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 100Mbit ceil 100Mbit

## DEFAULT CLASS ##
tc class add dev eth0 parent 1:1 classid 1:13 htb rate 89Mbit ceil 100Mbit

## EMULATED SPECIAL CLASSES AND NETEM QDISCS ##
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 1Mbit ceil 100Mbit
tc qdisc add dev eth0 parent 1:11 handle 10: netem loss 1% delay 50ms
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 10Mbit ceil 100Mbit
tc qdisc add dev eth0 parent 1:12 handle 20: netem loss 2% delay 200ms

## FILTERS FOR SPECIAL CLASSES ##
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 \
        match ip dst 10.0.1.0/24 flowid 1:11
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 \
        match ip dst 10.0.2.0/24 flowid 1:12
```

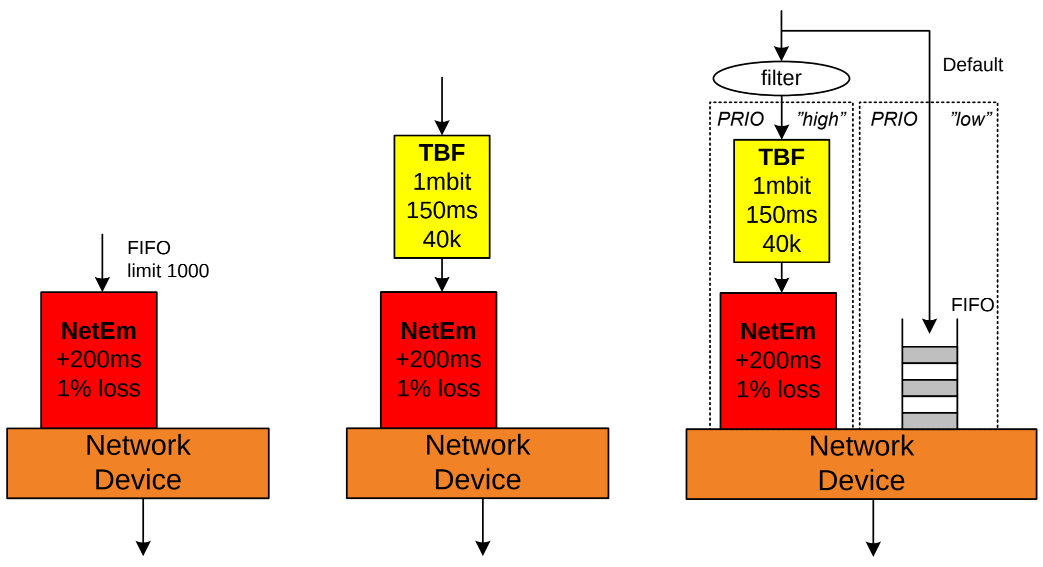
By borrowing tokens from its parent, for example class 1:11 might use momentarily 100 Mbps even though its maximum rate is much lower. This requires that the other classes have no traffic at all. Normally there is traffic in all of the classes, so the borrowing happens in the proportion of the configured rates. In the case shown in Figure 5.2, class 1:13 can borrow the most as it has the highest rate, 89 Mbps.

Even more complex setups are possible with HTB, as the tree can be easily extended. Other qdiscs have different features, and depending on the situation, the best of them can be chosen. More information on Linux traffic control can be found from [77].

5.2 NetEm queuing discipline

As can be seen from Table 5.1, NetEm is one of the available classful qdiscs. NetEm by default uses a variant of pfifo to store packets based on their time stamps [58]. This queue is known as tfifo.

NetEm can be attached to a root qdisc or to another classful qdisc. The only limitation of NetEm is, that it cannot be attached to itself, so stacked NetEm qdiscs cannot be used. There really is no reason for this, as all the emulation parameters can be given to one NetEm instance. Stacked NetEm instances are a different thing than adjacent NetEm qdiscs in different classes, as in Figure 5.2, they should not be mixed up. In addition to these, NetEm's FIFO queuing can be replaced with some other qdisc to achieve some other type of queuing.



(a) NetEm with FIFO queuing

(b) NetEm with TBF queuing

(c) NetEm with priorities and TBF queuing

Figure 5.3: Some configurations with NetEm.

Figure 5.3 reveals some different ways on how NetEm can be used. In the simplest scenario in Figure 5.3(a), NetEm is directly attached to root qdisc, its tfifo queue length is set to 1000 packets, packets are delayed 200 ms, and 1% packet loss is applied. In Figure 5.3(b), tfifo is replaced by TBF, which allows rate controlling. TBF parameters control the size of its bucket, not the emulation properties like latency.

Third option seen in Figure 5.3(c), is similar in principle with the HTB example in Section 5.1. This alternative allows different kind of traffic to be filtered into separate bands, which can all have different emulation options. Now PRIO qdisc is used to give different priorities to the traffic, and only some of the traffic is diverted to NetEm with TBF queuing. The rest of the traffic passes by the default band untouched. With PRIO a maximum of 16 classes can be used. If more classes are required, other classful qdiscs like HTB should be considered.

Table 5.3 lists available parameters for NetEm. The length of the FIFO queue in packets is configurable using the *limit* parameter. Emulated *delay* in milliseconds can have jitter and correlation values, and even the *delay distribution* can be changed. In addition to the four distributions the NetEm provides, users can generate their own distribution table based on experimental data and use that instead. [23]

Table 5.3: Available NetEm parameters.

```
Usage: ... netem [ limit PACKETS ]
                 [ delay TIME [ JITTER [CORRELATION]]]
                 [ distribution {uniform|normal|pareto|paretonormal} ]
                 [ loss PERCENT [CORRELATION]]
                 [ corrupt PERCENT [CORRELATION]]
                 [ duplicate PERCENT [CORRELATION]]
                 [ reorder PERCENT [CORRELATION] [ gap DISTANCE ]]
```

Packet loss is given in percentages and its correlation can be used to emulate packet burst losses. The *corruption* parameter introduces a single bit error in a packet at random offset, and *duplication* is specified the same way as packet loss. *Reordering* can be defined in two ways, by reordering a certain percentage of packets or by reordering every Nth packet. Tfifo queuing might also reorder packets if they have high jitter values, but this can be prevented by chancing tfifo to pfifo. Reordering requires some delay to be functional. [23]

NetEm has one issue related to the timers it uses. In high speed networks, a low tick rate will cause bursty traffic, even if NetEm is not configured to emulate burstiness. Every time the timer expires, NetEm relays a burst of packets to the ring buffer of the network interface. If the buffer limit is too low, some of these packets might be dropped, depending on the network driver [58]. High resolution timers used in recent Linux kernels decrease the probability of this, as they use the most precise timer available. However, if the traffic rate is high enough, bursts might still occur.

5.3 Typical WAN characteristics

Emulation of the Internet conditions requires knowledge of the real world situation. The most important factors for real-time streaming are packet delay, jitter, and loss. Even though there is no "typical" values for the metrics [7], some assumptions can be made. *Packet delay* can be estimated most precisely of these three. Formula (2) [78] shows how nodal delay (d_{nod}) for a packet is formed. Nodal delay is the total delay a packet experiences at one node.

$$d_{nod} = d_{proc} + d_{queue} + d_{trans} + d_{prop} \tag{2}$$

Processing (d_{proc}) and transmission delays (d_{trans}) occur in the networking device, they are generally negligible in modern high-speed WAN networks. Queuing delay (d_{queue}) is slightly more complicated matter, but it can be controlled with different queuing disciplines explained in Section 5.1. Propagation delay (d_{prop}) is expressed in Formula (3) [78]. D corresponds to physical distance between devices and S is the speed of propagation. S depends on the physical medium the packet travels on, for fiber optics it is about 2/3 of the speed of light in a vacuum.

$$d_{prop} = \frac{D}{S} \tag{3}$$

One way theoretical propagation delay from Helsinki to New York is about 33 ms, when to Stockholm it is only 2 ms. Round Trip Time (RTT) is double the delay. However, in practice the RTT delay is larger than that, as the queuing delay component caused by cross-traffic in the network increases the total delay. End-to-end delay depends on the number of routers along the path, but on long distance connections the propagation delay is the one that matters the most. This is especially true if there are any satellite hops along the path.

In LAN networks the total RTT delay is usually 1 ms or less, as was seen in Chapter 4 measurements. RTT in WAN networks hardly ever exceeds 500 ms, if there is no satellite hops on the path. *Jitter* is the variation of consecutive RTT delays, which is caused mostly by insufficient bandwidth or congestion in the path.

Congested routers along the path also cause *packet losses*. Packet loss depends heavily on the current networking conditions, which are different around the world. For example, connections in Europe might encounter 0% loss, when concurrently in North America the loss might be 4%. On the next day, the losses might be the other way around. End-to-end loss rate is hard to estimate due to routing.

There exists multiple Internet "weather report" services, that attempt to monitor the state of the Internet. One of them is the PingER project [79], that mainly uses ping facility to gather data around the world. Table 5.4 shows some average values from January to October 2009. As mentioned above, the values vary in different parts of the world.

Table 5.4: WAN metrics with 1000 byte packets [79].

Continent	packet loss	avg delay (ms)	avg jitter (ms)	deviation (ms)
Inside Europe	1,11%	37,72	26,68	22,42
Inside North America	0,17%	59,38	51,01	41,30
North America → Europe	0,91%	174,19	33,92	28,43
Europe → North America	0,38%	150,91	43,59	29,57

In general, packet losses less than 1% are considered good and 1–2,5% acceptable. With higher percentages TCP connections start behaving poorly. The limit is even stricter with real-time communications such as Voice over IP (VoIP), where packet losses in excess of 1,5% are considered poor. [79]

Elastic applications like email or web browsing are highly tolerant to delay and jitter, but multimedia applications are stricter on the matter. For instance, VoIP requires RTT delay less than 400 ms, but human ear cannot distinguish delays smaller than 150 ms. [78] The RTT jitter values of 0–40 ms are good, 40–100 ms acceptable, and over 100 ms poor [79]. Many streaming applications use a playout delay to help with the delay and jitter issue, but in all cases that is not possible.

All in all, the current network conditions in Europe and North America seem to be suitable even for real-time streaming applications. Historical data gathered by the PingER project indicates, that the trend is towards better and faster connections in the future.

Available bandwidth the peers have is the last factor to be considered. In typical P2P situation peers are behind ADSL connections. This means they might have different upload and download bandwidths available, typically the upload rate is lower than download. For download 1 Mbps could be a common bandwidth for most of the peers, when 10 Mbps, 24 Mbps, or even 100 Mbps connections might exist. Upload bandwidths normally do not exceed 2 Mbps, but the typical value is lower than that.

In the case of typical RTP2P overlay, SDS will most probably have higher bandwidth than normal peers. As real-world P2P streaming application usually uses less than 400 Kbps [5] of bandwidth, even the slowest 1 Mbps connections should be sufficient. Depending on the coding of the streamed content, the bandwidth RTP2P requires might vary. With the content used in this thesis the average bandwidth one peer uses stays below the 400 Kbps threshold, as Figure 4.4 in Chapter 4 shows.

Network characteristics considered above are true for landline networks, but the growing numbers of *cellular broadband* end-users complicate the situation. Cellular users might have as low as 384 Kbps connections with latencies in the order of hundreds of milliseconds even inside one country. Figure 5.4 reveals some estimations of the latency values in cellular networks. Also jitter in a cellular network is higher, but packet loss typically stays close to the values of landline networks.

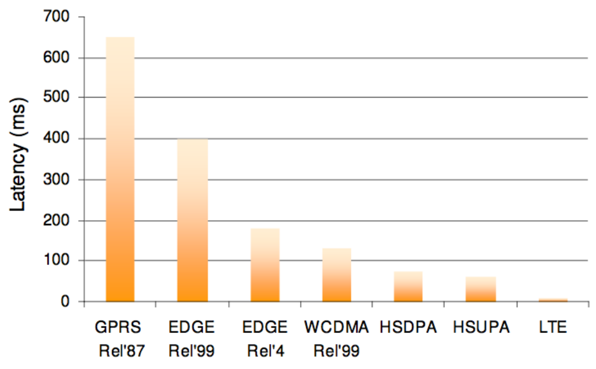


Figure 5.4: Some latency estimations in cellular networks [80].

From Figure 5.4, the future trend in cellular network latencies can be seen. The future Long Term Evolution (LTE) networks will have similar latency values than landline networks. The first LTE networks have just been opened in Sweden. However, the current mobile network conditions in Finland for packet data are far from the values achievable with LTE in the future. Sometimes the theoretical values the Internet Service Providers (ISP) promise for customers are not met. This had to be tested.

A brief ping test from DNA network to TUT domain using HSDPA cell phone with a 384 Kbps bandwidth gave RTT latency of 396 ms with a 98 ms deviation. The values are far from the theoretical maximums HSDPA should be capable of. In a similar test with 24 Mbps Sonera ADSL connection the latency was only 25 ms with 9 ms deviation. This illustrates the difference between landline and cellular networks well.

As the difference between data in Figure 5.4 and the empirical ping test indicate, it is difficult to estimate average latency values for mobile networks. This issue is not DNA specific, it concerns all the other ISPs as well. The network metrics depend heavily on the ISP used, signal strength, the locations of antennas, and many other things. The only sure thing is, that the latencies will shorten when cellular networks evolve. Nevertheless, the results received with ping indicate the current 3G network conditions and they will serve as reference metrics for the following tests.

5.4 Experimenting with RTP2P software

Time has come to perform the actual performance tests with RTP2P. A few experimental test runs are performed, with and without the retransmission feature of RTP2P. The following tests reveal how RTP2P will perform in cellular environment. The lack of automatic reconnection, mentioned in Chapter 4, hindered these tests as well, so no extensive experimentation was done. The purpose of this section is just to prove that RTP2P can operate on mobile conditions.

Chapter 3 summarized, that the WANemulator should be capable of the traffic generated by 320 peers. The type of the streamed content also matters. However, with that many peers, the CPU power at host PCs became a bottleneck. With too many peers the CPUs of the PCs were overloaded and the RTP2P applications acting as peers started core dumping. Some bugs in the SDS were also discovered. The lack of CPU time at host PCs caused unfair delays between peers at a virtual machine, and also some other irregular behaviors were encountered.

After an extensive search, a setup of 160 peers with the maximum cluster size of 70 was found the most stable one. This corresponds to five peers at one virtual machine and two virtual machines at each of the 16 PCs. While searching for the optimal setup, cluster sizes smaller than 70 seemed to cause more peers to drop from the overlay. In a smaller cluster a peer has less neighbors to ask for the streamed data. If a peer cannot find partials in time, it simply shuts itself down and exits the overlay [1]. This is where the reconnection feature would have been useful.

Even though the peer amounts used in preliminary tests before this thesis could not be reached, the setup with 160 peers is sufficient to prove the RTP2P performance in a traffic shaped network. The setup used in this chapter is the same than the one introduced in the beginning of Chapter 4, so the figures in that chapter are comparable to the ones seen in this section.

First, the performance of RTP2P is tested in a scenario where the network metrics correspond to a mobile environment. However, the available bandwidth for peers is not limited in the first scenario. Next, emulation rules are not used and RTP2P is tested in a scenario where the bandwidth is limited. In this scenario the traffic is still going through the WANemulator, it is just not delayed and it does not experience emulated packet losses. Finally, both of the setups are combined and a full scale cellular environment test is performed.

5.4.1 Emulated scenario

The setup for the first phase is seen in Table 5.5. PRIO qdisc is used to form four bands, into which traffic is categorized. The first two bands, 1:1 and 1:2, are automatically created and are not shown in the table. They take care of the maintenance traffic for the network environment, while the last two bands, 1:3 and 1:4, perform the emulation. Two different NetEm rule sets are formed, rule 40: is for mobile RTP2P clients and rule 30: is for SDS and the original data source.

Table 5.5: Tc parameters used in WANemulator for the emulated environment.

```
## ROOT QDISC, 4 PRIORITY BANDS ###
tc qdisc add dev eth1 root handle 1:0 prio bands 4

## TWO DIFFERENT NETEM INSTANCES ##
tc qdisc add dev eth1 parent 1:3 handle 30: netem loss 0.1% delay 20ms 13ms limit 1000
tc qdisc add dev eth1 parent 1:4 handle 40: netem loss 0.5% delay 200ms 50ms limit 2000

## FILTERS TO DIVERT TRAFFIC INTO BANDS ##
tc filter add dev eth1 protocol ip parent 1:0 prio 3 u32 \
        match ip src 10.10.51.1/32 flowid 1:3
tc filter add dev eth1 protocol ip parent 1:0 prio 3 u32 \
        match ip dst 10.10.51.1/32 flowid 1:3
tc filter add dev eth1 protocol ip parent 1:0 prio 4 u32 \
        match ip src 10.10.0.0/16 flowid 1:4
```

Mobile peers experience greater loss and delay, for them an uniform 1% loss and 400 ms delay with 100 ms jitter is configured. SDS and original data source have better networking conditions, that correspond to landline network. Filters divert traffic into correct NetEm qdiscs based on IP address. No traffic shaping is performed and the NetEm instances use default tfifo queuing. The packet limit for mobile peers had to be increased into 2000 to ensure no packets were artificially dropped due to a full tfifo queue at the WANemulator.

The peers experience double the rates that are configured with Tc, as bidirectional traffic passes the WANemulator twice and the emulation is performed both ways. This is due to the architecture of the network environment, the matter was explained in Chapter 4.

Table 5.6: Ping results with 1450 byte packets, 200 pkt/s.

Scenario	packets sent	packets received	packet loss	time elapsed	RTT min (ms)	RTT avg (ms)	RTT max (ms)	std dev (ms)
No emulation	4500000	4499913	0,0019%	513 min	0,31	1,03	138,52	0,49
Mobile	4500000	4455118	0,9974%	465 min	286,38	401,60	595,82	40,80
Landline	4500000	4490966	0,2008%	466 min	11,06	42,01	668,02	10,75

The emulation values were verified with an overnight ping run, the results of which are shown in Table 5.6. Even though the virtual machines running the RTP2P peers had only 0,5% packet loss configured in Table 5.5, Table 5.6 show that they have experienced approximately 1% loss. The results in Table 5.6 indicate that NetEm gives quite a good accuracy in packet loss and average delay. The jitter values cannot directly be seen from the table, but shorter ping runs proved NetEm accurate in jitter as well.

Figure 5.5 shows the data on WANemulator received from successful test runs with RTP2P.

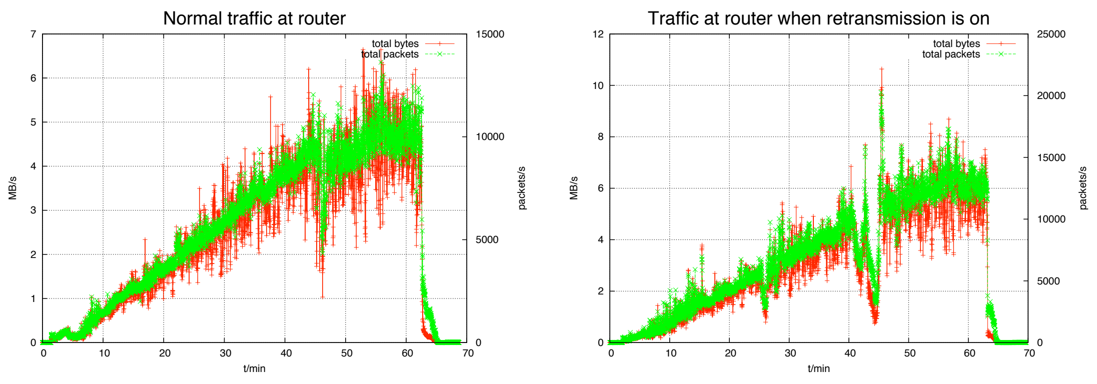


Figure 5.5: Traffic in the emulated environment.

The results are quite the same than in Figure 4.2, when there was no emulation used. In both of the emulated runs, almost all of the peers successfully joined in the service and remained there until the end of the stream.

Occasionally, NetEm's queue at WANemulator reached over 1000 packets, but the limit of 2000 packets was never reached. As the limits were not exceeded, no superfluous packet loss appeared. Accuracy of NetEm was verified from the statistics of Tc tool, and configured packet loss rates similar to the ones in Table 5.6 were observed.

5.4.2 Limited bandwidth without emulation

Next step is to disable the emulation rules in the WANemulator, and see how the peers survive the situation where they do not have enough bandwidth to operate. The traffic is still going through the WANemulator, but no emulated behavior, such as packet loss or delay, is applied.

To gain limited rates for both upload and download traffic, traffic shaping has to be done in two separate places, in the WANemulator and directly at the peers. As egress traffic at the WANemulator is ingress traffic for VMs, also the ingress traffic for VMs can be shaped. Egress traffic at a VM can be limited locally.

The chosen scenario corresponds to real world situation quite well, as now both upload and download bandwidth for each peer is limited. Egress shaping at peers forms the queues into the user's end and not into the router, which is exactly as it should be. This way a peer cannot exceed its upload capacity. Even though the upload capacity is limited, a peer might still download from multiple sources, which is seen as peaks in ingress traffic. To prevent this, WANemulator was used to limit the traffic sent to a peer.

Despite the fact that rate limitation is possible, it can only be done on a virtual machine basis. As there are multiple RTP2P peers running on one VM, the rate that can be configured is actually used by all of them. That is an issue, that cannot be easily circumvented as there were not enough resources available. Multiplexed RTP2P instances were a necessity to gain higher peer amounts for more realistic usage scenarios. Especially if the retransmission feature is turned on, lost segments on one peer might cause traffic spikes, so that the other peers at VM experience delays, even though they have not lost any segments.

Table 5.7: Tc rules for rate limitation at virtual machines.

```
## ROOT QDISC ##
tc qdisc add dev eth0 root handle 1:0 htb default 20

## MAIN CLASS ##
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 1.1Mbit ceil 1.1Mbit

## CLASS FOR RTP2P TRAFFIC ##
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 1Mbit ceil 1.1Mbit

## CLASS FOR MAINTENANCE TRAFFIC ##
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 0.1Mbit ceil 1.1Mbit

## FILTER TO DIVERT RTP2P INTO ITS CLASS ##
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
        match ip dst 10.10.0.0/16 flowid 1:10
```

Rules used to limit the egress traffic at VMs can be seen in Table 5.7. HTB is used to limit the RTP2P traffic in class 1:10 to 1 Mbps. Class 1:20 is for maintenance traffic coming outside the 10.10.0.0/16 domain. If there is no maintenance traffic, RTP2P can borrow bandwidth from the maintenance class up to the ceil rate of 1.1 Mbps.

HTB might not be available in normal end-user devices, where queuing is done using a simple FIFO queue. In this test it does not matter, as HTB is only used to shape the rate of the traffic. All the RTP2P traffic is actually filtered into HTB class 1:10, which internally uses pfifo qdisc. As long as the default packet limit of 1000 packets is not reached, shaping works just as a normal FIFO queue would do. Actually, there was hardly any maintenance traffic in class 1:10 during the tests, so it can be neglected.

Table 5.8: A bash script to enable traffic shaping at WANemulator.

```
#!/bin/bash
DEV=eth1
NUM_ROWS=4
NUM_PCS=4
NUM_VMS=2
RATE=1000
CEIL=$(($RATE+100))
TOTAL_RATE=$(((1+$NUM_ROWS*$NUM_PCS*$NUM_VMS)*$RATE+100))
QDISC="sudo tc qdisc add dev ${DEV}"
CLASS="sudo tc class add dev ${DEV}"
FILTER="sudo tc filter add dev ${DEV} parent 1:0 protocol ip"

## ROOT QDISC, TOTAL RATES AND MAINTENANCE TRAFFIC CLASS ##
$QDISC root handle 1:0 htb default 10
$CLASS parent 1:0 classid 1:1 htb rate ${TOTAL_RATE}Kbit ceil ${TOTAL_RATE}Kbit
$CLASS parent 1:1 classid 1:10 htb rate 100Kbit ceil 500Kbit

## RULES FOR EACH VIRTUAL MACHINE ##
for ((i=1; i<=$NUM_ROWS;i+=1)); do
   for ((j=1; j<=$NUM_PCS;j+=1)); do
      for ((k=1; k<=$NUM_VMS;k+=1)); do
         $CLASS parent 1:1 classid 1:${i}${j}${k} htb rate ${RATE}Kbit ceil ${CEIL}Kbit
         $QDISC parent 1:${i}${j}${k} handle ${i}${j}${k}: pfifo limit 1000
         $FILTER  prio 2 u32 match ip dst 10.10.${i}${j}.${k}/32 flowid 1:${i}${j}${k}
      done
   done
done

## RULES FOR SDS AND ORIGINAL DATA SOURCE ##
$CLASS parent 1:1 classid 1:511 htb rate ${RATE}Kbit ceil ${CEIL}Kbit
$QDISC parent 1:511 handle 511: pfifo limit 1000
$FILTER  prio 1 u32 match ip dst 10.10.51.1/32 flowid 1:511
$FILTER  prio 1 u32 match ip src 10.10.51.1/32 flowid 1:511
```

In Table 5.8, the script used to form Tc rules at the WANemulator is shown. A three-layer tree is formed. HTB parent class 1:1 has all of the virtual machines in their own classes as its children, NetEm qdiscs are attached to these child classes. Every VM had to be put into its own class, because otherwise the rate limitation based on destination IP address of a packet would not have been possible. A class 1:10 for maintenance traffic is also formed. Packets are filtered into correct classes based on their destination IP address.

The teaching laboratory, where the environment was built, has sixteen PCs and there are two virtual machines at each PC, with the addition of one VM running SDS and original data source on one extra PC. Therefore, the total rate the parent class 1:1 has is 33 times 1 Mbps, totalling to 33,1 Mbps. Each VM has 1 Mbps maximum rate and 0,1 Mbps is reserved for the maintenance traffic. The borrowing feature of HTB was not wanted, so the ceil rates in child classes, serving one VM, were only 1,1 Mbps. This corresponds more closely to the configuration in the real world, where ISP does not want users having more bandwidth than they are paying for.

The bandwidths configured at the WANemulator are the same as at the VM end, which means symmetric upload and download bandwidths for VMs. If asymmetric speeds are wanted, the rates could be easily changed at either end. In the scenario chosen, SDS and original data source have higher bandwidth available as they share 1 Mbps bandwidth, while five peers at other VMs have to divide their 1 Mbps. Therefore, each peer running on a VM has a calculated value of 200 Kbps for both upload and download.

The packet queue limit of 1000 packets in each class is more than adequate, as each VM has its own queue. SDS and original data source running at 10.10.51.1 have a higher priority filter than the VMs, so that all traffic destined or sourced from 10.10.51.1 is treated first.

The rate limits used are not corresponding to a real-world use scenario. Modern 3G devices use higher rates than 200 Kbps, as in Section 5.3 was explained. The rate used here was chosen solely to represent the behavior of RTP2P when rate limitation occurs. If the streamed content would have a higher bitrate, peers would require more bandwidth and a more realistic traffic rate limit could be used.

Figure 5.6 reveals the findings at virtual machines with rate limitation in both of the cases. At the top row, the total traffic rate is shown, the next row reveals the impact of egress shaping, and at the bottom row limited ingress traffic is shown. Total traffic is ingress and egress traffic summed together. As in similar figures in Chapter 4, the colors in the figures indicate different virtual machines.

Especially from the egress figures, the effectiveness of HTB rate limitation can be seen. As there was no maintenance traffic, the peers had 1,1 Mbps rate available, which corresponds to the 0,1375 MBps rate seen from the figures. In a normal run, traffic shaping has started to occur at a time instance of 40 min, when with retransmission it started 10 minutes earlier. The shaping is not as evident in ingress figures than in egress, but the lack of traffic spikes exceeding 1,1 Mbps reveal that shaping has occurred.

Some test runs were also performed without the ingress rate limitation to VMs. In these test runs sudden bursts of traffic had occurred, as the incoming traffic was not limited. This had a significant effect on the total traffic rate at virtual machines. With ingress shaping,

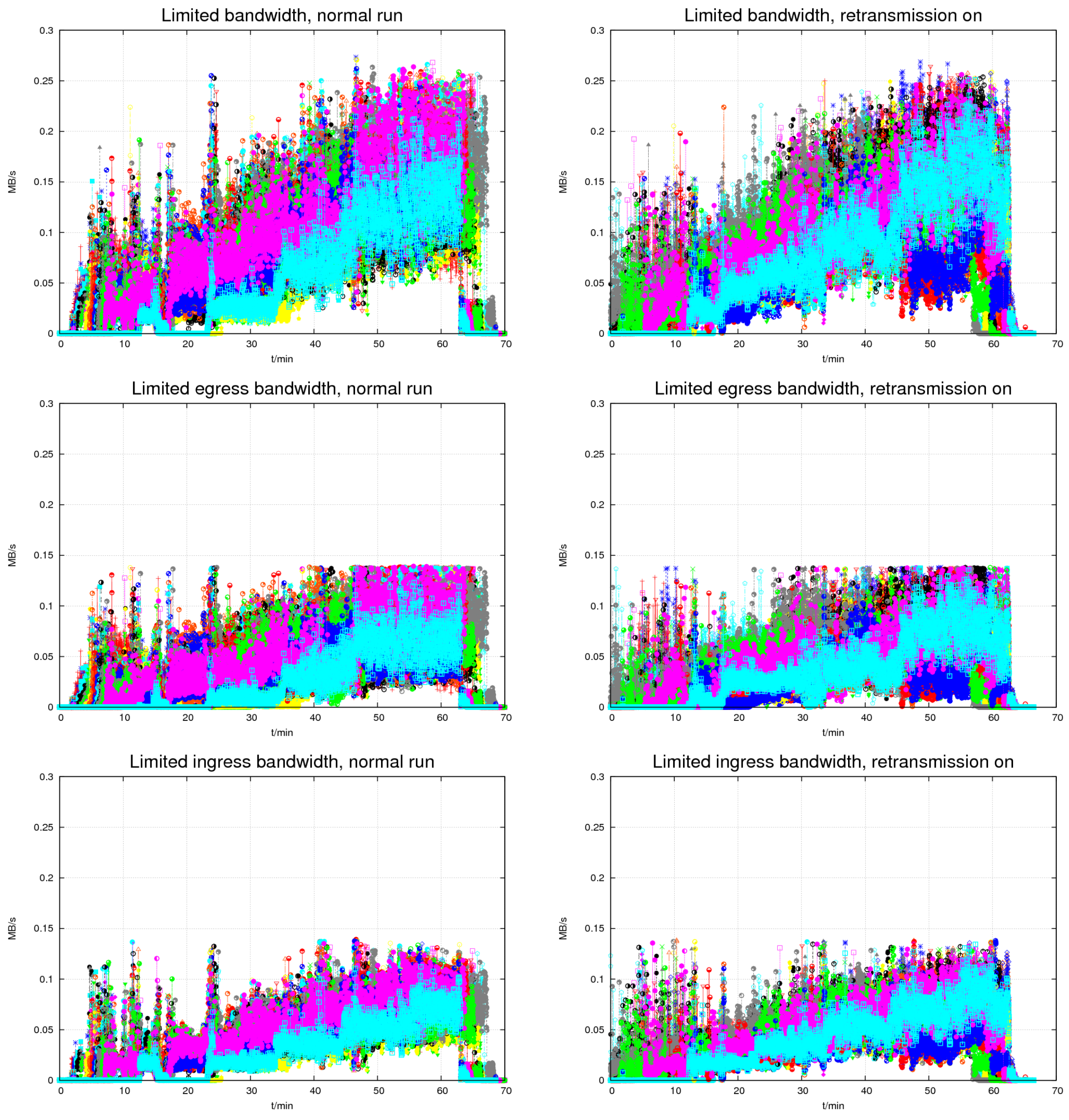


Figure 5.6: Rate limited results at virtual machines.

the total traffic remains quite steady in both cases, with and without retransmission.

Tc statistics tell, that no packet loss has occurred. HTB shaping has only put packets into queues, that might cause some packet delay. In the worst case scenario, the queue forms both at peers and at the WANemulator, so the packet has to wait in a queue twice. This has happened, which simultaneous ping with RTP2P traffic on one peer reported. The highest queuing times were as high as 600 ms, but most of the traffic had insignificant delays.

Peer count in the overlay could not be reliably gathered from the SDS database, as it indicated only half the amount of peers, than could be seen running in virtual machines at the end of the runs. There seemed to be some issues in the script used to gather the data from the database, but a specific spot in the script could not be identified. Therefore, no figure could be drawn on overlay joins.

However, with an SSH script the amount of running peers at each of the VMs could be counted. The script indicated about a 20 peer shortfall from the expected 160 in both of the runs.

5.4.3 Mobile environment with limited bandwidth

In the final and most precise scenario, rule sets used in Sections 5.4.1 and 5.4.2 are combined to gain emulated environment with rate limitation in place. Virtual machines had similar settings to limit their rate as was shown in Table 5.7, but rule set in Table 5.8 was modified to enable NetEm emulation.

However, modifications were quite simple, as it only required to change pfifo qdiscs in child classes to NetEm. As in Section 5.4.1, mobile peers were configured to experience 1% packet loss and 400 ms delay with 100 ms jitter. SDS and original data source had better conditions, as they only experienced 0,2% loss and 40 ms delay with 26 ms jitter. Queue limits were set to 1000 packets in both, which was adequate as each VM had its own queue as in Section 5.4.2 was explained. The rest of the settings in Table 5.8 were not touched.

From Figure 5.7 the traffic rates from the mobile environment test runs can be seen. This time the rate limitation also in ingress traffic is more visible. Other than that, the traffic rates seem quite the same as in Figure 5.6, as could be expected when the rate limits were the same. The real difference was in average packet losses and delays. Ping results from one peer can be seen in Table 5.9.

Table 5.9: Ping results with 200 byte packets, 1 pkt/s for 1h.

Scenario	packet loss	RTT min (ms)	RTT avg (ms)	RTT max (ms)	std dev (ms)
Normal run, peer	0,94%	183,79	411,26	1144,00	58,70
Normal run, SDS	0,28%	8,31	49,90	449,22	30,17
Retransmission, peer	1,31%	177,03	457,59	4188,72	267,64
Retransmission, SDS	0,28%	16,23	57,92	2125,96	97,03

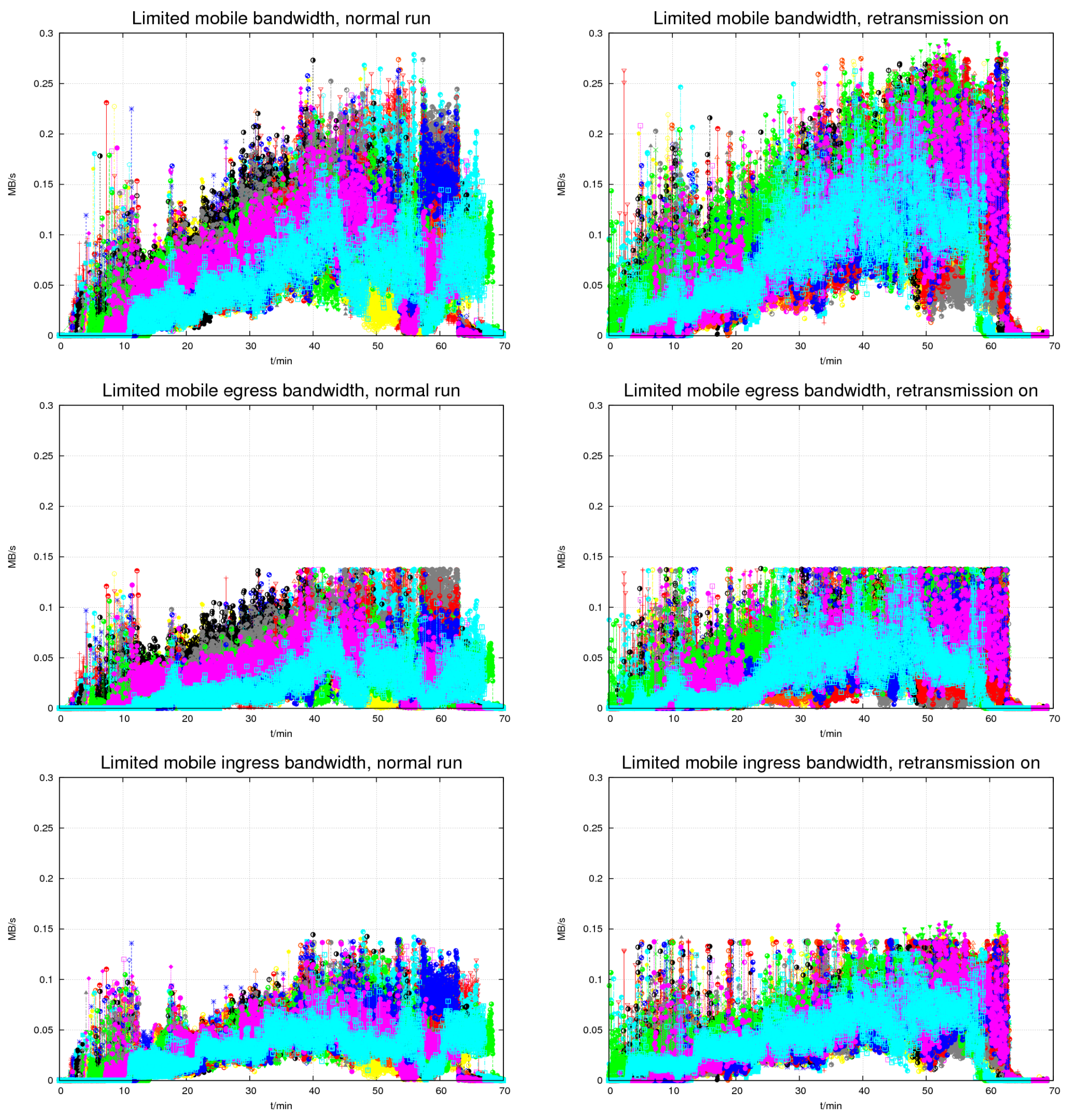


Figure 5.7: Full mobile environment results at virtual machines.

The results in Table 5.9 are not fully reliable, as the sample was small, only 3600 packets. However, as ping was run on the side of RTP2P peers at the VM, all of the packets were treated equally, as they all used the same queue. Therefore, ping gives a rough estimation what the peers experienced.

Data in Table 5.9 indicates, that rate limitation has increased the average delay approximately 10 ms from the emulated values. The maximum delays have been much greater than in earlier tests because of queuing at either VM, WANemulator, or both. Now there were more packets in the queues at WANemulator than in Section 5.4.2, because the packets were queued to achieve emulated delay. Still, the limits of the queues were not reached. This was verified from Tc statistics, where the packet loss rates were closer to the configured values, than ping results in Table 5.9 indicate.

Peer count statistics could not be gathered from SDS in this case either, but the SSH script was used again to gather the number of RTP2P instances at VMs. The script revealed, that this time retransmission run had performed better. It had lost the same 20 peers than in Section 5.4.2, while without retransmission 45 peers had quit the service. As with all of the test runs before this, the results have uncertainties due to the lack of automatic reconnectioning of RTP2P.

5.5 Summary

This chapter focused on running RTP2P on the built environment. The environment was setup to correspond a cellular network environment with realistic packet losses, delays, and jitters. Also the bandwidth of the peers was limited, so that the scenario would be as close to a real world situation as possible. Virtualized 160 mobile peers were joining the streaming overlay, which was run by the original data source of the stream and Service Discovery Server that were on a wired landline network.

Although not all of the peers remained in the overlay in the tests, it was important to notice that RTP2P can operate at a realistic rate limited network, where packet losses and network delays are occurring. The delays some packets encountered are too long for real-time streaming, which most likely affects the quality of the user experience. Unfortunately, due to the lack of time, the user experience was not researched.

Without modifications to the RTP2P source code, it is difficult to have better test runs than the ones received in this chapter. The randomness of the peer departs from the overlay hinder the test results, so that it is troublesome to make conclusions of a single peer behavior.

Also, the automated peer joins do not correspond to the real P2P situation with joining and leaving peers, which is also known as peer churning. RTP2P has a prototype of a random peer churning algorithm embedded, but as this behavior is artificial and caused unstable results, it was not used.

The results presented in this chapter are selected among a few test runs, not all of them performed as good as the ones shown here. There were less peers in the overlay at the end of some runs. One of the runs did not even finish, as for some reason the original data source stopped streaming the content.

However, these issues cannot be seen as a result of emulation, as similar behavior was experienced already before any emulation was used. Therefore, the results received in this chapter should not be generalized, they only prove, that RTP2P can handle realistic packet losses and delays with jitter to some extend.

6 Discussion

The process of building a network environment for RTP2P is now done. The environment structure is presented in Chapter 3 and the capabilities of the environment are explained in Chapter 4. Chapter 5 focused on some experimental test runs with RTP2P. Reasons for the decisions made have been presented, but some things could have been done differently. A brief discussion about these matters follows. At the end of this chapter an evaluation of the thesis in general is done.

6.1 About the network environment

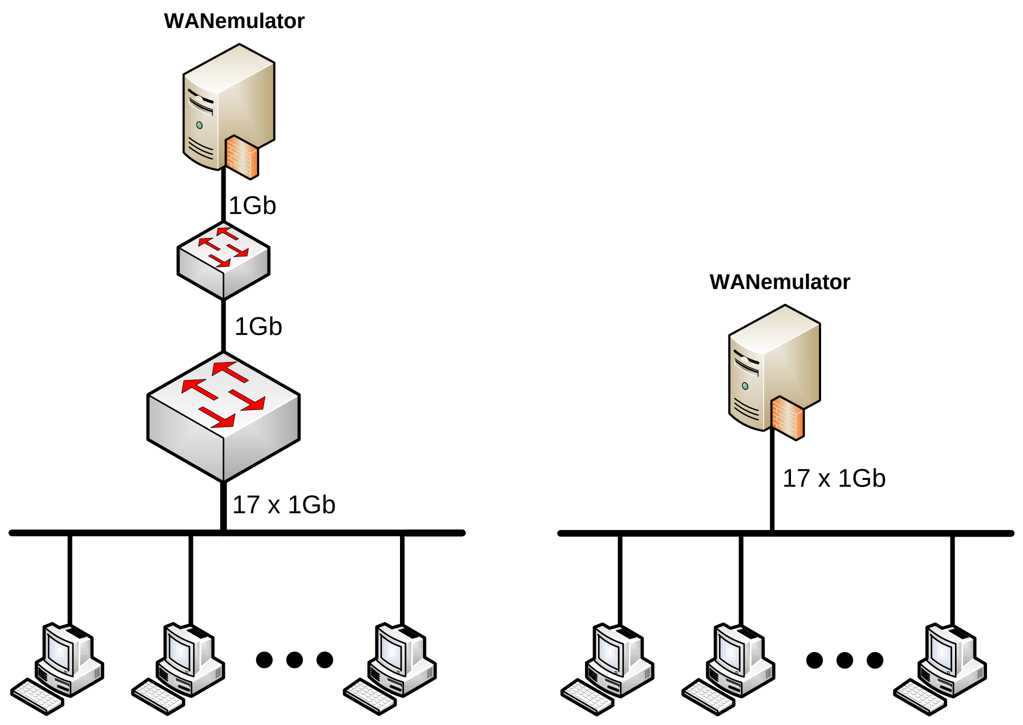
Proving the environment reliable was troublesome. The randomness of RTP2P application made it useless for the environment capability testing, as it was impossible to find out whether the source of irregular behavior was the RTP2P software or the environment. Traffic generators had to be used instead.

Without complex configuration, the traffic generators cannot be used to produce the random traffic typical of P2P environment. Therefore, the tests performed with traffic generators do not fully correspond to the situation of normal RTP2P run. The capability limits of the environment stated at the end of Chapter 4 might not hold with RTP2P.

However, the testing phase of the environment was important as it helped the author of this thesis to fully comprehend the operating principle of the environment. This was the point when the architectural defect of the environment was discovered.

A centralized emulation node, WANemulator, was used. NetEm performed the emulation at WANemulator in router mode. Another option would have been using NetEm in bridged mode, which would have made the WANemulator fully transparent at IP level. There are benefits and disadvantages on using Layer 2 device as discussed in [81], but the main reason not to use bridged WANemulator was the lack of necessary equipment.

To gain actual advantage of bridging, WANemulator should have been brought closer to the host computers. Figure 6.1 illustrates the issue. In Figure 6.1(a) all of the traffic is passed through a single 1 Gbps link to WANemulator, which becomes a bottleneck in a high speed scenario. The issue is explained better in Chapter 4. In Figure 6.1(b) each of the 17 PCs are connected separately via their own 1 Gbps link to the WANemulator, which removes the bottleneck from the network.



(a) Current setup (b) Desirable setup

Figure 6.1: Comparison of the setups.

However, this would require 17 physical interfaces on WANemulator, and maybe an 18th one for remote administration. If the bridging is done with the virtualized WANemulator running on VMware in the current setup, the same bottleneck remains as WANemulator still physically exists behind the switches. Virtualized interfaces do not help if they are connected to the same physical link. Nevertheless, the lack of sufficient hardware forced the author of this thesis to use the current setup.

In the early stage of the network environment construction, a separate PC was used to host the WANemulator, but its capacity was found inadequate. Therefore, a powerful server was harnessed to run the WANemulator. This lead to the unfortunate situation with only one link to and from the WANemulator. Thus, the routing table trick to force all of the environment traffic through the WANemulator was necessary. As can be seen from the results, this had a deteriorative effect on the performance. Some minor enhancements could be achievable by tuning the queue lengths or using different qdiscs at WANemulator, but the physical limitation of one network interface remains.

Another solution would have been building a distributed testbed as in [61]. Their testbed was build on FreeBSD using Dummynet and process level virtualization to gain scalability and lower overhead. They virtualized the network identity of the P2P processes by binding each process to a different alias IP address. To achieve this, they had to intercept some network system calls by modifying FreeBSD C library. They pointed out, that the virtualization of a full operating system is not necessary for P2P applications. Traffic shaping in their P2PLab is performed at the end nodes, so no single emulation node is necessary. Also in [33] it was found out, that introducing a centralized emulation node to a network does not get support from the users, which justifies the decentralized approach.

The centralized approach used in this thesis simplified the administration of the environment, and all the same emulation options are possible as with the distributed model. Besides, the virtual machines at the edges of the network have the same Tc utility installed as the WANemulator does. Tc at the VMs was used to gain realistic limited bandwidths for peers. This in turn reduced the burden the WANemulator has to handle, as the maximum traffic a VM can send was limited. Actually, even the emulation itself could be done at the edge nodes as NetEm is bundled with Tc. This was not tested, as the configuration of the distributed approach was seen too demanding.

The decision between Linux and FreeBSD was more intuitive than scientifically justified. They are both UNIX-based operating systems, so their differences are minor. The peers had to be run at Linux as RTP2P is developed on it, but the WANemulator could have been built on FreeBSD as well. By choosing Linux as a platform for the WANemulator ensured that only one operating system had to be learned.

Full-virtualization might be an overkill in performance sense, but it was the only feasible option for this thesis. The need to separate the testing environment from physical PC configurations made full-virtualization necessary. It ensured easier maintenance and faster setup of the environment. More details on the matter can be found from Chapter 3. IP aliasing might have been possible, but modifications needed to bind IP address to each P2P process was seen too demanding.

The choice to use virtualization was correct in a maintenance sense, but it caused some performance challenges discovered in Chapter 4. The issues with VirtualBox's internal bridging were surprising, but a greater issue was the high CPU usage of the virtual machines. Guest OS kernels were compiled with the default 250 Hz tick rate, which caused a lot of CPU usage at the host PC when RTP2P was running. By changing the rate to 100 Hz should make the virtual machines less resource demanding. With a lighter burden to the CPU, more peers could be run on the environment.

Full-virtualization causes extra overhead, but it differs on different virtualization products. One choice worth trying might be to convert the VirtualBox images to VMware format and try the free VMware Player in order to lower the overhead. This might remove the performance issue with the internal bridging as well.

Security issues were not given much thought in this thesis. No firewalling was done and the peers operated on a full connectivity scenario. This might not be the case in a real world network, where peers might reside behind NAT devices and firewalls. The encapsulation of the experiments into a separate user account on the host PCs, and running the test inside a virtual machine on this account, could be categorized as a security method. This was seen as adequate security matter in this thesis.

Even though the environment built in this thesis can be setup rapidly from a scratch, it is not very scalable in a maintenance sense. Manual configurations to the bash scripts and Dnsmasq configuration file are needed if more virtual machines and peers are added to the environment. If more automated setup is required, for example Emulab might be an option. It has well documented practices on what needs to be done to setup an emulated networking environment on powerful servers. Anyone can join if certain prerequisities are met.

During the literature survey it became evident that for a large-scale testing, local networking environments are hardly used. Most of the emulators surveyed were old, discontinued projects. Simulation is still performed locally to prove the concept of a new application valid, but with local emulation the behavior of the Internet cannot be properly modeled. This is especially true for complex networks such as P2P. Some preliminary tests can be done locally to emulate packet losses and delays for a link, but for a proper wide-scale experiment, controlled and well-established distributed testbeds like Emulab or PlanetLad should be used.

Even better solution would be releasing the software to the end-users, as some P2P software developers have done. This solves the resource issue as the real users contribute their own hardware for the test. This removes the need for multiplexed peers on one computer, as was done in this thesis. Multiplexing is an artificial technique, which might cause some performance issues because of the peers sharing the resources of one computer. By releasing the software, natural peer churning is also achieved, the mathematical models can only predict the user behavior to some extend.

Naturally, releasing is not a suitable option for software in too early stages of development. P2P application should also be designed from the beginning to be released as a product and monitored during its actual usage. RTP2P does not fulfill either of these demands, so releasing it was not an option. Besides, the patenting issues made releasing RTP2P impossible as well. The same applies to the distributed testbeds, neither could they be used. Therefore, a local test environment had to be built for this thesis.

6.2 Evaluation of the work done

The research plan for this thesis was wide from the beginning. It was unsure how a P2P application like RTP2P should be tested, and there were many features in RTP2P that needed measurement data in a realistic networking environment. The plan had to be revisited during the work several times. Unfortunately, this lead to an omission of many interesting tests about RTP2P features. To keep the effort reasonable for a thesis work, the thesis was delimited to focus only on building the environment. The RTP2P tests in Chapter 5 were run only to see that the created network environment is realistic for testing P2P software.

At the beginning of the project, it took quite a while to get familiar with RTP2P, and then the literature study was performed during the summer months. Many different emulators were trialed, which took some time as well. When NetEm and VirtualBox were chosen as the base for the environment, the work was just about to start.

Plenty of time and effort was spent on writing and modifying the bash scripts to build the environment. As there were hundreds of multiplexed peers, controlling them proved to be demanding. The same applies to data analyzing, as the peers produced megabytes of log files from where the essential information had to be found. Some bash scripts were prepared for starting and controlling the test run as well as to collect logs from the peers.

After the environment was built, the capability tests explained in Chapter 4 were done. This took some time as well, even though the tests were kept as simple as possible. The sheer amount of necessary test runs was large, after which the data had to be understood. This applies to the test runs with RTP2P, too. Hundreds of test runs with RTP2P were performed during the thesis work.

Many things that were listed in the research plan at the start of the thesis work were not completed. For some of them, the instabilities in RTP2P software can be blamed, but the rest of them just had to be skipped due to many unpredictable issues encountered on the way. RTP2P is functional software, which would benefit from the automatic reconnection feature and some debugging properties. Without reconnection it is difficult to perform automated testing on large overlays. Also, the behavior of users is problematic to emulate in an automated scenario, which would require more consideration.

The built network environment is suitable for testing P2P streaming applications, even though it uses tools designed for traditional client-server communication. Actually, P2P environment is nothing more than a mesh of client-server connections, so such tools can well be used. The mesh is where the complexity of P2P environment is concealed.

During the thesis work, a considerable amount of time was spent on figuring out how to control the number of different peers in the overlay. A centralized emulation node easened the task, but it has its downsides that are explained earlier in Section 6.1. Even with the emulation node, the manual configuration necessary when the tests were setup and run, was tremendous. It should be considered to use some of the existing P2P testbeds, such as

PlanetLab or Emulab, in the future tests. The testbeds automate the configuration, leaving more time for the actual performance evaluation.

Considering the constraints, it can be concluded that this thesis has fulfilled the most important objectives in the research plan. A quickly deployable testing environment for P2P streaming applications has been built and its capabilities have been found. The environment was used to run experimental tests for RTP2P application. RTP2P seemed to be operating correctly in an environment having mobile networking conditions.

7 CONCLUSIONS

In this thesis, a network environment suitable for testing P2P streaming applications was built. In the beginning of the thesis, an extensive literature survey was made to find the best solution for testing P2P streaming applications. Then, a network link emulator, NetEm, was chosen to be used at a centralized emulation node, WANemulator, which produced realistic packet losses, delays, and jitters to the network.

VirtualBox was used to keep the teaching laboratory, where the environment was built, still suitable for teaching during the testing period. With the help of virtualization, the testing environment could be encapsulated inside a few virtual images, that were easily deployable and configurable to the laboratory. The images also reduced the maintenance effort, as the testing environment could be kept intact from the underlying operating system upgrades. Both the WANemulator and the VirtualBox virtual machines are operating in a Linux environment.

The environment was proved to be reliable with the chosen streamed content and 160 peers. Depending on the content encoding and the WANemulator's settings, a bigger overlay should be achievable, but this was not tested. In the end of this thesis, a P2P streaming application, RTP2P, was tested in the environment. The network metrics were chosen to correspond mobile conditions, in which the application performed well. The tests with RTP2P were not extensive, their purpose was only to prove the environment suitable for testing with P2P streaming applications.

The environment has certain limitations, that were discovered while it was built. The bridge implementation in VirtualBox was not capable to the traffic rates its documentation promised. Another issue was the surprisingly high virtualization overhead, that limited the amount of virtual machines at the host computers. In addition to these, the centralized emulation architecture, where the WANemulator has only one network interface, acts as a bottleneck. Due to the lack of resources, a routed solution had to be used, which in combination with centralized scenario decreases the performance of the environment.

Regardless of the issues discovered in the assembly of the environment, it is suitable for testing P2P streaming applications, if the traffic limits are respected. For more reliable results with P2P streaming applications, bigger overlays are desirable than can be achieved with the environment in its current state. However, the environment can be used in decentralized mode as well, because NetEm is installed onto the virtual machines as well. This should increase the number of peers, that the environment can handle.

In general, emulated local network environments are seldom used for testing these days. However, a local environment was a necessity for this thesis, because of the patenting issues related to the RTP2P application that was tested in the environment. For more scalable and automated testing environment, some of the distributed network testbeds, such as PlanetLab or Emulab, should be considered. Such testbeds correspond better with the networking situations similar to the Internet, too.

Besides the distributed mode tests in the built environment, also user experience in an emulated scenario should be evaluated. Some of the delays, that the RTP2P application encountered, were too long to be feasible for a real-time media streaming. It would be interesting to see how these affect the video and audio quality at peers. In addition, more tests with RTP2P should be performed to see how it operates in varying networking conditions. Also, to get rid of the VirtualBox issues, the VDI images could be converted to VMDK format and be tested with VMware Player.

Even though not all of the questions in the research plan were answered, a reliable testing environment for P2P streaming applications was achieved. The rest of the questions can be researched later on, as the network environment is ready for more thorough testing with RTP2P and other applications, if it is seen necessary in the future.

REFERENCES

[1] J. Peltotalo, J. Harju, M. Saukko, L. Vaatamoinen, I. Bouazizi, I.D.D. Curcio, and J. van Gassel. A Real-Time Peer-to-Peer Streaming System for Mobile Networking Environment. In *INFOCOM Workshops 2009, IEEE*, pages 1–7, April 2009.

[2] A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill Higher Education, 1999.

[3] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A survey of peer-to-peer network simulators. In *Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK*, 2006.

[4] L. Nussbaum and O. Richard. A Comparative Study of Network Link Emulators. In *12 Communications and Networking Simulation Symposium (CNS'09)*, San Diego, USA, 2009.

[5] M. Wang and B. Li. Network Coding in Live Peer-to-Peer Streaming. *Multimedia, IEEE Transactions on*, 9(8):1554–1567, Dec. 2007.

[6] C. Feng and B. Li. On large-scale peer-to-peer streaming systems with network coding. In *MM '08: Proceeding of the 16th ACM international conference on Multimedia*, pages 269–278, New York, NY, USA, 2008. ACM.

[7] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking (TON)*, 9(4):392–403, 2001.

[8] 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.

[9] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo. Planetsim: A new overlay network simulation framework. *Software Engineering and Middleware, SEM 2004, Linz, Austria*, 3437:123–137, 2005.

[10] The Network Simulator - ns-2. [WWW]. [Cited 18.8.2009]. Available at: `http://www.isi.edu/nsnam/ns/`.

[11] M. Jelasity, A. Montresor, GP. Jesi, and S. Voulgaris. The Peersim Simulator. [WWW]. [Cited 13.9.2009]. Available at: `http://peersim.sf.net`.

[12] PlanetSim project. [WWW]. [Cited 13.9.2009]. Available at: `http://projects-deim.urv.cat/trac/planetsim/`.

[13] J. Pujol-Ahulló, P. García-López, M. Sànchez-Artigas, and M. Arrufat-Arias. An extensible simulation tool for overlay networks and services. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2072–2076, New York, NY, USA, 2009. ACM.

[14] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured peer-to-peer overlays. *Lecture Notes in Computer Science, Peer-to-Peer Systems II*, 2735:33–44, 2003.

[15] R. Baumann and U. Fiedler. RplTrc: A Tool for Emulating Real Network Dynamics for Performance Evaluation. In *Telecommunications, 2007. ConTel 2007. 9th International Conference on*, pages 219–226, June 2007.

[16] S. Hemminger. Network emulation with NetEm. In *Linux Conf Au*, 2005.

[17] Modelnet. [WWW]. [Cited 28.9.2009]. Available at: `https://modelnet.sysnet.ucsd.edu/`.

[18] Emulab: Total network testbed. [WWW]. [Cited 3.10.2009]. Available at: `http://www.emulab.net/`.

[19] PlanetLab: An open platform for developing, deploying and accessing planetary scale services. [WWW]. [Cited 3.10.2009]. Available at: `http://www.planet-lab.org/`.

[20] Dummynet home page. [WWW]. [Cited 28.9.2009]. Available at: `http://info.iet.unipi.it/~luigi/dummynet/`.

[21] NIST Net. [WWW]. [Cited 25.9.2009]. Available at: http://www-x.antd.nist.gov/nistnet/.

[22] WANem: The Wide Area Network emulator. [WWW]. [Cited 30.9.2009]. Available at: http://wanem.sourceforge.net/.

[23] The Linux Foundation - Net:NetEm. [WWW]. [Cited 1.10.2009]. Available at: http://www.linuxfoundation.org/en/Net:Netem.

[24] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 255–270, New York, NY, USA, 2002. ACM.

[25] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, 2002.

[26] K. Yocum, K. Walsh, A. Vahdat, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGCOMM Comput. Commun. Rev.*, 32(3):28–28, 2002.

[27] X. Liu and A.A. Chien. Realistic Large-Scale Online Network Simulation. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 31, Washington, DC, USA, 2004. IEEE Computer Society.

[28] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the Emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

[29] J. Chen, D. Gupta, K.V. Vishwanath, A.C. Snoeren, and A. Vahdat. Routing in an Internet-scale network emulator. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings. The*

IEEE Computer Society's 12th Annual International Symposium on, pages 275–283, Oct. 2004.

[30] Modelnet testbed. [WWW]. [Cited 28.9.2009]. Available at: `http://www.csee.usf.edu/~guerrerc/tbed_modelnet.htm`.

[31] Some answers on Modelnet. [WWW]. [Cited 28.9.2009]. Available at: `http://www.ics.uci.edu/~mayur/model-net-details.html`.

[32] S. Guruprasad, R. Ricci, and J. Lepreau. Integrated network experimentation using simulation and emulation. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 204–212, Feb. 2005.

[33] M. Carbone and L. Rizzo. Adding emulation to Planetlab nodes. Technical report, Dip. di Ingegneria dell'Informazione, Universita' di Pisa, Italy, June 2009.

[34] Xen hypervizor. [WWW]. [Cited 4.10.2009]. Available at: `http://www.xen.org/`.

[35] VMware Business Infrastructure Virtualization. [WWW]. [Cited 4.10.2009]. Available at: `http://www.vmware.com/`.

[36] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[37] T. Roscoe. The PlanetLab Platform. *Lecture Notes in Computer Science, Peer-to-Peer Systems and Applications*, 3485:567–581, 2005.

[38] Linux-Vserver. [WWW]. [Cited 17.10.2009]. Available at: `http://linux-vserver.org/`.

[39] VNET: PlanetLab Virtualized Network Access. [WWW]. [Cited 17.10.2009]. Available at: `http://www.planet-lab.org/doc/vnet`.

[40] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[41] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: realistic and controlled network experimentation. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–14, New York, NY, USA, 2006. ACM.

[42] L. Peterson, A. Bavier, M.E. Fiuczynski, and S. Muir. Experiences building PlanetLab. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 351–366, Berkeley, CA, USA, 2006. USENIX Association.

[43] MyPLCUserGuide - PlanetLab. [WWW]. [Cited 17.10.2009]. Available at: `https://svn.planet-lab.org/wiki/MyPLCUserGuide`.

[44] M. Carbone and L. Rizzo. Dummynet Revisited. Technical report, Dipartimento di Ingegneria dell'Informazione, Università di Pisa, May 2009.

[45] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.

[46] M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126, 2003.

[47] L. Nussbaum. NISTNet on recent Linux kernels (2.6.26+). [WWW]. [Cited 11.10.2009]. Available at: `http://www.loria.fr/~lnussbau/#nistnet`.

[48] NIST Net Next Generation. [WWW]. [Cited 1.10.2009]. Available at: `http://nistnet.sourceforge.net/`.

[49] RplTrc: A Tool for Emulating Real Network Dynamics for Performance Evaluation.

[WWW]. [Cited 25.9.2009]. Available at: `http://rpltrc.hypert.net/`.

[50] RFC3549. Linux Netlink as an IP Services Protocol. [WWW]. [Cited 1.10.2009]. Available at: `http://www.ietf.org/rfc/rfc3549.txt`.

[51] A. Keller. Trace Control for Netem. [WWW]. [Cited 2.10.2009]. Available at: `http://tcn.hypert.net/tcn.pdf`.

[52] MasterShaper - Network traffic under control. [WWW]. [Cited 1.10.2009]. Available at: `http://www.mastershaper.org/`.

[53] J. Fabini, P. Reichl, C. Egger, M. Happenhofer, M. Hirschbichler, and L. Wallentin. Generic access network emulation for NGN testbeds. In *TridentCom '08: Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[54] Linux IMQ - Intermediate Queueing Device. [WWW]. [Cited 11.10.2009]. Available at: `http://www.linuximq.net/`.

[55] T. Gleixner and D. Niehaus. Hrtimers and beyond: Transforming the linux time subsystems. In *Proceedings of the Ottawa Linux Symposium, Ottawa, Ontario, Canada*, 2006.

[56] TCN: Trace Control for Netem. [WWW]. [Cited 25.9.2009]. Available at: `http://tcn.hypert.net/`.

[57] A. Keller. Manual: tc Packet Filtering and netem. [WWW]. [Cited 2.10.2009]. Available at: `http://tcn.hypert.net/tcmanual.pdf`, July 2006.

[58] The Netem Archives. [WWW]. [Cited 2.10.2009]. Available at: `https://lists.linux-foundation.org/pipermail/netem/`.

[59] Iproute2. [WWW]. [Cited 1.10.2009]. Available at: `http://www.linuxfoundation.org/en/Net:Iproute2`.

[60] Linux Advanced Routing and Traffic Control. [WWW]. [Cited 1.10.2009]. Available at: `http://lartc.org/`.

[61] L. Nussbaum and O. Richard. Lightweight emulation to study peer-to-peer systems. *Concurr. Comput. : Pract. Exper.*, 20(6):735–749, 2008.

[62] F.L. Camargos, G. Girdard, and B. des Ligneris. Virtualization of Linux Servers: a comparative study. In *Proc. of the 2008 Linux Symposium*, volume 1, pages 63–76, 2008.

[63] QEMU: open source processor emulator. [WWW]. [Cited 25.10.2009]. Available at: `http://www.qemu.org/`.

[64] Kernel Based Virtual Machine. [WWW]. [Cited 16.10.2009]. Available at: `http://www.linux-kvm.org/page/Main_Page`.

[65] VirtualBox. [WWW]. [Cited 16.10.2009]. Available at: `http://www.virtualbox.org`.

[66] OpenVZ Wiki. [WWW]. [Cited 26.10.2009]. Available at: `http://wiki.openvz.org/Main_Page`.

[67] G.J. Popek and R.P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[68] The netfilter.org project. [WWW]. [Cited 16.10.2009]. Available at: `http://www.netfilter.org/`.

[69] Dnsmasq - a DNS forwarder for NAT firewalls. [WWW]. [Cited 16.10.2009]. Available at: `http://www.thekelleys.org.uk/dnsmasq/doc.html`.

[70] Sun VirtualBox User Manual. [WWW]. [Cited 16.10.2009]. Available at: `http://www.virtualbox.org/manual/UserManual.html`.

[71] Wireshark: network protocol analyzer. [WWW]. [Cited 22.10.2009]. Available at: `http://www.wireshark.org/`.

[72] Bandwidth monitor NG. [WWW]. [Cited 22.10.2009]. Available at: http://www.gropp.org/?id=projects&sub=bwm-ng.

[73] D-ITG, Distributed Internet Traffic Generator. [WWW]. [Cited 23.10.2009]. Available at: http://www.grid.unina.it/software/ITG/.

[74] Iperf. [WWW]. [Cited 23.10.2009]. Available at: http://sourceforge.net/projects/iperf/.

[75] D.I. Wolinsky, A. Agrawal, P.O. Boykin, J.R. Davis, A. Ganguly, V. Paramygin, Y.P. Sheng, and R.J. Figueiredo. On the Design of Virtual Machine Sandboxes for Distributed Computing in Wide-area Overlays of Virtual Workstations. In *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*, pages 8–8, Nov. 2006.

[76] VMware Knowledge Base: Article 1428. [WWW]. [Cited 12.11.2009]. Available at: http://kb.vmware.com/kb/1428.

[77] A Practical Guide to Linux Traffic Control. [WWW]. [Cited 12.11.2009]. Available at: http://blog.edseek.com/~jasonb/articles/traffic_shaping/.

[78] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, 2009.

[79] The PingER project. [WWW]. [Cited 14.11.2009]. Available at: http://www-iepm.slac.stanford.edu/pinger/.

[80] M. Družijanić T. Blajić, D. Nogulić. Latency Improvements in 3G Long Term Evolution. [WWW]. [Cited 3.12.2009]. Available at: http://www.ericsson.com/hr/about/events/archieve/2007/mipro_2007/mipro_1137.pdf.

[81] The Linux Foundation - Net: Bridge. [WWW]. [Cited 2.10.2009]. Available at: http://www.linuxfoundation.org/en/Net:Bridge.