Application of Virtualization technology to the study of Quality of Service techniques.

Juan Quiros*, Paulino Ruiz-de-Clavijo†, Alejandro Carrasco‡, Julian Viejo§ and Alejandro Millan¶
*†§¶ Group ID2 (Research and Digital Development)
School of Computer Science (Electronic Technology)
University of Seville, Seville 41012, Spain
www.dte.us.es/id2
‡ Electronic Technology and Industrial Informatics
School of Computer Science (Electronic Technology)
University of Seville, Seville 41012, Spain
www.dte.us.es/id2
*jquiros@dte.us.es, †paulino@dte.us.es, ‡acarrasco@us.es, §julian@dte.us.es, ¶amillan@us.es

***Abstract*—In this article, the teaching of quality of service mechanisms in packet-switched networks is presented. To this end, a methodology based on virtualization technology is introduced. As a result, a truly practical approach is offered to students, in an accessible environment, and from a point of view suitable for the Bachelor's degree in Information Technology - Computer Science.**

***Keywords*—Virtualization, quality of service, qos, linux.**

I. Introduction

This paper introduces the methodology in the practical part of the subject titled "Advanced Technologies in Informatics", in the third year course of the Bachelor's degree in Information Technology - Computer Science, at the University of Seville.

The teaching experience of applying a new laboratory program parallel to the theoretical program, is described. This program not only maintains the coherence with the theoretical program, but also introduces on-trend technologies, such as the virtualization, *cloud computing* and QoS.

The laboratory sessions are incremental. Although each session has specific goals, it also depends on the previous sessions. Alltogether, these sessions constitute a complete network architecture that combines virtual and real resources. Finally, a set of services, such as HTTP and FTP, is included and some QoS mechanisms are applied to assure the proper operation of the system.

This paper is organized as follows. First, in Section 2, the goals and methodology are given. In Section 3, the QoS session is explained and, finally, the results and conclusions obtained are presented in Sections 4 and 5.

II. Goals and methodology

The main aim of the laboratory program is to provide all graduates with sufficient knowledge of information technology infrastructures, required by any public or private organization, while focusing on networking and its security. Students are supposed to have basic knowledge of networking, computer architecture, software engineering and operative systems. Specifically, the laboratory program enables graduates to:

- Deploy network infrastructures at both logical and application layers.
- Design and adopt security policies in computer networks.
- Configure advanced network services.
- Implement traffic classifications using QoS techniques.

In addition to these areas, certain concepts of Unix System administration, virtualization technologies, computer hardware and networks, are introduced throughout the course. This subject therefore provides students width a basic vision of the state of virtualization technology, as well as the deployment of IT infrastructures in flexible environments with few resources.

In order to achieve all these goals, the subject has been divided into four sections. Firstly, basic notions about network security are provided: general concepts, main network attacks, security threats, and perimeter security [1], [2]. Secondly, *Virtual Protect Networks* (VPN) [3] are introduced, which have OpenVPN as the reference implementation. Network traffic control and techniques of Quality of Service (QoS) [4], [5], [6] are explained in the third section, by focusing on managing bandwidth and marking network packets. Finally, some advanced network services, located in the Application layer of the OSI model, are introduced in order to enable students to choose one for in-depth analysis and deploy it in laboratory sessions.

A. Overall description of the laboratory sessions

The laboratory sessions have been designed as a complement to the theoretical part of the subject. Despite their similar structure, they are divided into five sessions that have to be completed in the order provided. The structure of each is similar: an introduction gives students an overall vision about state, methodology and goals of the session, and specific

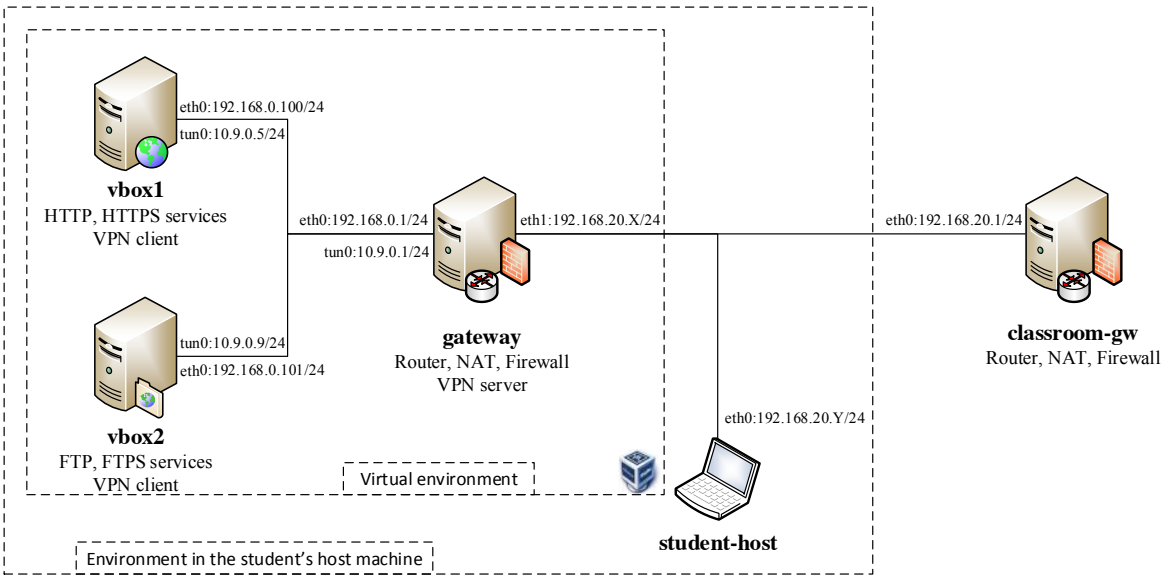

Fig. 1. Diagram of the virtual network.

knowledge of the technologies used in laboratories is provided. Secondly, students familiarize themselves with tools in a step-by-step section. Finally, the session ends with a non-guided section, thereby giving students the opportunity to students go further in depth and to put the acquired knowledge into practice.

All sessions take place in the context of the network diagram shown in Fig. 1, in which there are two computer networks: the network *192.168.0.0/24* is private for each student, but *192.168.20.0/24* is common to everybody. The latter is a virtual network deployed over the physical network of the laboratories. It has internet access through *classroom-gw*, which is configured as a gateway. Two addresses of this network are reserved for each student, one for the virtual *gateway* and another for the host machine,*anfitrión-alumno*. Hence, it is possible to deploy the IT infrastructure required by laboratory sessions, by reusing the current infrastructure and minimizing changes. *Vbox1*, *vbox2* and *gateway* are located on the private network. Regarding the virtualization software, ORACLE VIRTUALBOX [7] has been chosen.

The first session is seen as an introduction. The goal is to deploy the basic virtual IT infrastructure, (see Fig. 1), so that more services can be added in the following sessions. Thus, three virtual machines with the Operative System (OS) UBUNTU SERVER 12.04 [8] are created and configured in accordance with the diagram network.

The next session complements the security concepts of the theoretical part of the subject. Its objective is to implement a firewall in the virtual machine *gateway*, using IPTABLES, a component of the framework NETFILTER, inside the Linux kernel [9], [10]. Therefore, some basic concepts and usage of *iptables* are introduced, while filtering, Network Address Translation (NAT), and log creation are explained in greater detail.

The third session is related to the second theoretical topic. Its purpose is to create and configure a VPN [3] in the virtual network *192.168.0.0/24*. To this end, the interfaces *tun0* are created in the virtual machines (Fig. 1), and the network is built with the software OPENVPN [11].

In the following block, the services HTTP and HTTPS are added to the virtual machine *vbox1*, while FTP and FTPS are added to *vbox2*. Students have to compare, choose, install and configure the necessary software resources in order to provide these services. Therefore, the virtual machine *gateway* should be configured to enable access for any machine in the network *192.168.20.0/24*. These new services are needed in order to be able to complete the next block.

In the last session, the network architecture is completed with traffic control and QoS mechanisms. This practical part coincides with the third theoretical block, and is the most extensive, stating about 40% of all the subject.

Finally, students have to put into practice all the knowledge acquired throughout all the sessions. The practical part of the subject ends with a project in which a new service must be added to the virtual infrastructure, (see Fig. 1). Examples include a network security audit, a load-balancing transparent reverse proxy, installing and configuring IT infrastructure monitoring software, such as NAGIOS [12] and MUNIN [13], and installing, configuring and comparing "file hosting" systems, such as BTSYNC [14] and OWNCLOUD [15].

III. QUALITY OF SERVICE

LINUX offers a wide range of flow-control mechanisms. However, the great number of alternatives, poor documentation, and the fact that more importance is given to the

development and integration of new modules than to usability, make the configuration of these mechanisms complex. For this reason, only those most used in real environments have been selected.

A. Theoretical basis

The LINUX [6] traffic control implementation covers four operations: the conformance, packet reordering, packet matching, and packet dropping. To carry out these operations, each network interface is configured using several policies. In LINUX, the policies can include one or more packet queues, and following custom rules, the packet classifier puts each input packet in a queue. The packet classifier consists of a set of custom rules called filters. Filters include information about the destination queue for each packet when it matches with the rule specification.

All queuing disciplines are attached to a network interface, which is associated to a hierarchical tree. The parent node of the tree is the network interface and the filters attached to this node determine the destination queue for each incoming packet. The classifier system introduces greater complexity when child queues also contain filters. In this scenario, packets hop between child nodes. It is not recommended to attach filters to internal nodes of a network interface.

Certain queuing disciplines include more than one node in the interface queuing tree. These queuing disciplines are made up of sub-nodes where the packet can be queued. Figure. 2 shows a diagram of a possible solution using several queuing disciplines. In order to understand the classifier process, it is necessary to discuss in-depth the three different nodes of the tree: queuing disciplines, classes, and filters.

Classes are incorporated into a queue discipline and are used to classify the packets. For example, if the use of a priority queuing discipline with 5 queues is desired, it is not necessary to attach 5 queuing disciplines to the network interface tree. The priority queuing discipline has the ability to add the child sub-classes required with the priority sets for each sub-class.

The task of disciplines is to change the way data is sent. Their main functions are: accepting, delaying, rescheduling, and dropping packets. Not all existing queue disciplines include child classes, and hence are grouped into two types of queue disciplines: (*classless*) and (*classful*).

On the other hand, the filters are defined as classification rules. In order to ascertain of how rules are applied to packets, it is necesary to know how the LINUX kernel interacts with the class tree. For each packet, the kernel applies only those rules attached to the top node of the network-interface queuing-discipline tree. When the packet matches one rule, the kernel sends it to the queue established in the rule. As mentioned above, child queues can also contain filters but it is a complex solution and is used only in scenarios where the performance is a critical requirement.

Once the queuing disciplines form part of the hierarchical tree, the full tree has two types of nodes: queuing discipline and class. For each tree node, a unique identifier is used to refer to it in filters. The syntax is similar to that used for identification of kernel devices in LINUX, which consists of a pair of *major* and *minor* numbers. The pair of major and minor numbers for a queue discipline is assigned in the configuration by the administrator. The major number identifies the queuing discipline, and the minor number identifies the sub-classes of the queuing discipline, therefore, all sub-classes of one queuing discipline have the same major number.

After having reviewed the main concepts, several queuing policies can now be selected. The Linux kernel supports a wide variety of disciplines, of which some are only experimental and others are deprecated or fail to function correctly. The following are those selected for our purpose:

- **PFIFO and BFIFO:** These are simple single queuing behaviors based on FIFO (*First In First Out*). Both queues are configured with a fixed queue size and differ in units used for configuration: fixed size in packets (PFIFO) or fixed size in bytes (BFIFO).
- **PFIFO_FAST:** Default queuing discipline set for the interfaces. Consists of a FIFO queue divided into three bands. Each band is assigned a priority related with the TOS field of IPv4 packets.
- **SFQ (*Stochastic Fairness queuing*):** This detects flows of packets through analyzing the source and destination fields. This policy reschedules packets and perturbs the flows by approaching a round-robin schedule for flows.
- **TBF (*Token Bucket Filter*):** This is a traffic shaper and never schedules packets. It can be configured with rate limits and burst size.

Regarding *classful* queuing disciplines chosen for laboratories, only two have been selected:

- **PRIO:** Queuing discipline with configurable priority bands which is not a traffic shaper.
- **HTB (*Hierarchy Token Bucket*):** An advanced queuing discipline that organizes its child sub-classes into a hierarchical tree of TBF sub-classes. It is a traffic shaper and shares the flow rate with its child sub-classes, and includes a mechanism to implement priority between its child sub-classes.

In addition to that above, the study and test of other disciplines is proposed to the students. For this purpose, the students test disciplines such as: CHOKE (*CHOose and KEep for responsive flows*), CBQ (*Class-Based queuing*), DRR (*Deficit Round-Robin Scheduler*) and RED (*Random Early Detection*).

B. Configuration tools

There are a wide number of queuing disciplines available in the Linux kernel. To reduce the configuration complexity to the system administrator, all of them have a homogeneous configuration process using a command line tool called TC [6]. In fact, when the command TC is used, there are several common options valid for all disciplines followed by specific options for each discipline.

At a more detailed level, the TC command has three operational modes. Each mode is designed to configure one of three parts of the traffic system: queuing disciplines, classes or filters. An explanation of this tool goes far beyond the scope of this paper, but some examples are shown.

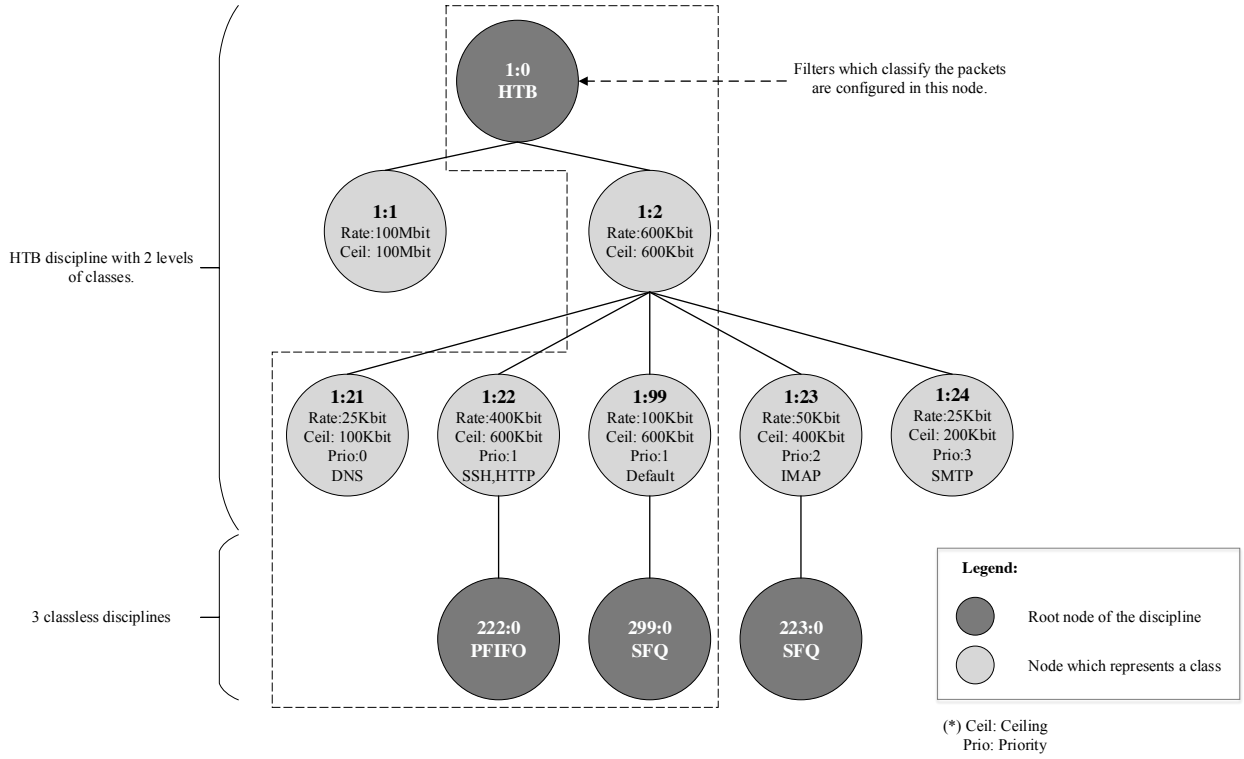

Fig. 2. Example of HTB queuing disciplines example used for DSL connections.

When the configuration of disciplines mode is used, it must indicate the discipline type, the node identifier (major number) and parent node. Code 1 refers to Fig. 2 and shows some examples: HTB at the top of the hierarchy (line 7), SFQ (lines 20-21) and PFIFO (lines 34-35).

The class configuration mode can be used when the queuing discipline is *clasful*. The syntax is similar to that used for disciplines with few differences. For the queuing discipline HTB, shown in Fig. 2, the configuration example (Code 1) is located on lines: 11-12, 18-19, 25-26 and 32-33.

The syntax of the filters is the same for all disciplines. One or more filters can be attached to one node of a *classful* queuing discipline. When several filters are attached to the same node, the filters are organized into a list where each filter has a set priority. A number represents the priority, whereby 1 is the highest priority. It is mandatory to set the filter priority. The filters are applied in order from high to low priority. If several filters have the same priority, then they are applied in sequential order, in the same order as they were added.

The classifier performs two tasks: pattern matching and action taking. The most common action is to put the packet into a hierarchy node. The filter works on OSI level 3; for example, it can test values on the IP header. The classifier is the most complex part of a rule due to the high number of options available in the classifier syntax. The two most used classifiers are: FW, which uses netfilter; and *u32*, which is able to perform in-depth analysis on the packets through masks. Some examples of filters with the U32 classifier are shown in Code 1 on lines 37-46.

C. Laboratory session details

The goal of this session is to put into practice the knowledge acquired on traffic managing and QoS, as well as to combine real and virtual resources in the same infrastructure. Thus, taking the network architecture deployed throughout the laboratory sessions as a starting point, students have to manage the traffic on interface *eth1* of the machine *gateway*, (see Fig. 1). To this end, several QoS policies are applied incrementally, from a really simple configuration to a complex one, as shown in Fig. 2. This set of policies is usually implemented in domestic environments, with the purpose of: controlling the traffic in *ADSL* connections; assuring DNS services; giving HTTP and SSH greater priority, followed by other network protocols; and giving IMAP and SMTP protocols the lowest priority, in that order.

The code necessary to configure the interface *eth1* of the machine *gateway*, in accordance with the highlighted fragment of Fig. 2, is given:

Code 1. Configuration, on interface *eth1* of the machine *gateway*, in accordance with the highlighted fragment in Fig. 2.

```
1 # Remove previous configuration tc qdisc
  del dev eth1 root
3
  #### Tree structure (classes and disciplines) ####
```

```
#HTB configuration for root node.
#Traffic is classified into 1:99 class by default
tc qdisc add dev eth1 root handle 1: htb default 99

#Traffic to internet is classified into 1:2 class
#Maximum bandwidth: 600Kbit.
tc class add dev eth1 parent 1:0 classid 1:2 htb \
    rate 600 ceil 6000 burst 3000

#Traffic by default is classified into 1:99 and
#299:0 classes
#Bandwidth (minimum/maximum): (100/600Kbit).
#In addition, a SFQ discipline is applied.
tc class add dev eth1 parent 1:2 classid 1:99 htb \
    rate 100kbit ceil 600 quantum 5000 prio 1
tc qdisc add dev eth1 parent 1:99 handle 299: sfq \
    perturb 5

#DNS traffic is classified into 1:21 class
#Bandwidth (minimum/maximum): (25/100Kbit).
tc class add dev eth1 parent 1:2 classid 1:21 htb \
    rate 25kbit ceil 100kbit prio 0

#SSH and HTTP are classified into 1:22 and 222:0
#classes, respectively.
#Bandwidth (minimum/maximum): (400/600Kbit).
#In addition, a PFIFO discipline is applied.
tc class add dev eth1 parent 1:2 classid 1:22 htb \
    rate 400kbit ceil 600 prio 1
tc qdisc add dev eth1 parent 1:22 handle 222: pfifo \
    limit 100

#### Classification filters ####
#HTTP and SSH traffic are classified into 1:22 node
tc filter add dev eth1 protocol ip parent 1:0 prio 2\
    u32 match ip sport 80 0xffff flowid 1:22
tc filter add dev eth1 protocol ip parent 1:0 prio 2\
    u32 match ip sport 22 0xffff flowid 1:22

#DNS traffic is classified into 1:21 node
tc filter add dev eth1 protocol ip parent 1:0 prio 2\
    u32 match ip dport 53 0xffff flowid 1:21
```

Finally, when the QoS policies are applied, students can test them using a script provided at the beginning of the session. This script shows statistics about all previously configured nodes and filters, through the TC tool. Fig. 3 shows the output of this script, at a given moment, whose configuration is based on the example of HTB hierarchy of the Fig. 2.

Fig. 3. Terminal output with statistics using the TC tool. The disciplines are surrounded by rectangles and classes by ellipses.

IV. RESULTS

This subject was given for the first time in the academic year 2012/13. Academic results of all students have been collected since that time. These results, and the number of laboratory sessions completed by students are shown in Fig. 4. There are two ways to pass the subject: through normal evaluation, in which students have to pass a theoretical exam and a practical exam, and via continuous evaluation, which follows the methodology described in this paper. In this regard, students have to complete the first three sessions to pass the laboratory part. In 2013, 58.5% of students passed the continuous evaluation, which makes up 76.9% of all students who passed the subject (83.3% of students presented), in comparison with those of 2014, of which 47.6% passed the continuous evaluation, which indicates that 75% of all students

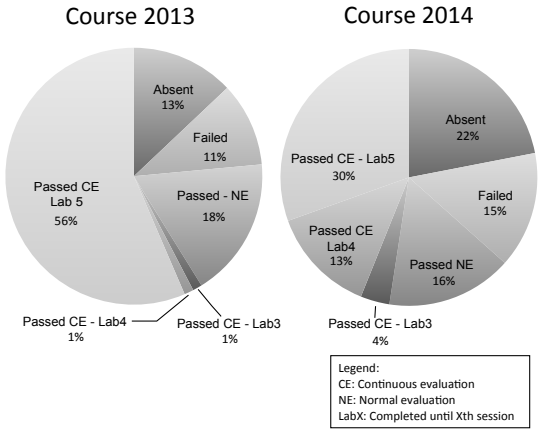

Fig. 4. Results of the academic years 2012/13 and 2013/14.

passed the exams (81.3% of students presented). As a result, the relation between students who passed the exam and those who presented themselves is maintainable. The decrease in the number of students who passed the subject was motivated by the transition towards the Bologna Process: in 2014 a major number of students had moved from old degrees. On average, these students had a higher level in the network area, since they had already passed old degree subjects which gave them an advantage over the rest and, in consequence, they obtained better results.

Regarding the completeness of laboratory sessions, it should be pointed out that, 2013, 56% of all students achieved the last block (QoS), completely or otherwise, in comparison to 30% of all students in 2014. The cause is the same as previously: the introduction of the Bologna Process.

Finally, it is important to mention the greatest difficulties encountered by students. The use of a command-line interface, instead of a graphic interface, is the most common difficulty, and the reason is clear: most students have only ever used graphical interfaces. The second obstacle is the administration of any server and, specifically, of Unix systems.

V. Conclusions

The laboratory program that is introduced in this paper has provided students with a more practical approach to the areas of network architecture, virtualization technologies and QoS, in addition to the theoretical knowledge.

Moreover, one of the main principles throughout all the laboratory sessions has been the reuse of current infrastructure, through environments which combine virtual and real resources. Furthermore, the license of the VirtualBox and LINUX software is free, which has enabled students to work outside the laboratories.

Acknowledgements

This work has been partially supported by the Ministry of Science and Innovation of the Spanish Government under project TEC2011-27936 (HIPERSYS), by the European Regional Development Found (ERDF), and by the Ministry of Education of Spain (FPU grant AP2009-3625).

References

[1] A. Villalon Huerta, "Seguridad en unix y redes," http://www.rediris.es/cert/doc/unixsec/, 2002.

[2] S. Northcutt, L. Zeltser, S. Winters, K. Kent, and R. W. Ritchey, *Inside Network Perimeter Security (2Nd Edition) (Inside)*. Indianapolis, IN, USA: Sams, 2005.

[3] M. Feilner, *OpenVPN: Building and Integrating Virtual Private Networks*. Packt Publishing, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1202604

[4] J. W. Evans and C. Filsfils, *Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[5] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. USA: Addison-Wesley Publishing Company, 2009.

[6] B. Hubert, "Linux advanced routing & traffic," http://www.lartc.org/, 2014.

[7] Oracle, "Virtualbox web page," https://www.virtualbox.org/, 2014.

[8] Canonical, "Ubuntu server web page," http://www.ubuntu.com/server, 2014.

[9] R. Russell, "Linux networking-concepts," http://www.netfilter.org/documentation/, 2001.

[10] M. Rash, *Linux firewalls : attack detection and response with iptables, psad, and fwsnort*. San Francisco: No Starch Press, 2007, index. [Online]. Available: http://opac.inria.fr/record=b1124526

[11] M. Feilner, *Beginning OpenVPN 2.0.9: Build and Integrate Virtual Private Networks Using OpenVPN*, ser. From technologies to solutions. Packt Publishing, Limited, 2009. [Online]. Available: http://books.google.es/books?id=SLlxmAEACAAJ

[12] W. Barth, *Nagios. System and Network Monitoring*. No Starch Press, 2006.

[13] B. t. Brinke, *Instant Munin Plugin Starter*. Packt Publishing, 2013.

[14] B. Inc, "Bittorrent sync user guide," http://btsync.s3-website-us-east-1.amazonaws.com/BitTorrentSyncUserGuide.pdf, 2014.

[15] ownCloud, "owncloud administrators manual," http://doc.owncloud.org/server/6.0/admin_manual/, 2014.