Teaching networks in the cloud

Augusto Ciuffoletti (augusto@di.unipi.it)
–
Università di Pisa, Dept. of Computer Science
P.le B.Pontecorvo, I-56122 Pisa (ITALY)

***Abstract*—The Web is populated by a growing number of services that provide access to remote IT resources: they are collectively addressed as the *Cloud*. Such incoherent and expanding number of services is investigated to find those that can help the task of teaching, focusing on a challenging case study for which I have a direct experience: a couobrse in *computer networks* with the purpose of giving the students a *hands-on* experience using production-grade techniques.**

The outcome of the case study is that *on-line* services can complement traditional frontal lectures, to enrich the communication between the teacher and the student, and to improve the learning experience. This is a hint for teachers, and characterizes a potential market for developers and providers.

I. Introduction

The availability of distributed services and resources is reshaping the way in which people learns and teaches: the successful diffusion of MOOCs, online tutorials, screen-casts and all sorts of online learning supports corroborates the statement. Not only the organizational aspects of educational institutions are under stress, as shown by the success of the Moodle project, but the tools themselves that the teacher uses to transfer his knowledge and competence to the students are under discussion [3].

In this latter perspective, the Internet provides the teacher with new tools and services, with a modality that has become popular under the term of *cloud computing*. They complement the traditional interactions between the teacher and the students taking place during frontal lectures, to help the teacher to improve the quantity of information conveyed during the lecture, to make it more appealing, to give the student the opportunity to retrieve documentation, to trial expertise, and to communicate outside lecture hours.

Such services are available in the Internet, and require a minimal preparation of the host. The administration of a Laboratory Classroom is thus simplified, and in many cases we may even address a Bring Your Own Device (BYOD) approach. Before embracing this latter option one should take into account that the users may not be proficient in technology, that the computational platform is going to be heterogeneous, and that service costs are an issue. As a matter of fact, most *Cloud Computing* services fit such context, and thus BYOD is of real interest.

The focus of this paper is on a quite specific branch of teaching, that directly interferes with the tools under study: computer networking. Practical activity plays a relevant role in a networking course, since it prepares students to participate into projects involving cutting edge technologies. However, the provision of a networking laboratory entails prohibitive costs, even when supported by a departmental cluster [1], [9], [2]: the adoption of cloud resources may solve the problem and cut the costs.

The intention of this paper is to evaluate the impact of some available cloud related technologies on the organization of the practical activity associated with a course on computer networks [4]. In the closing section three use cases are discussed, with examples of how existing cloud services can effectively improve the learning experience.

II. Requirements and assumptions

Hardware resources are considered first, since it is important to minimize this kind of investment. As a consequence performance limits are an issue, and in a BYOD context their definition becomes fuzzy: the less critical assumption is that students have access to an *entry level* computing device. Connectivity plays a vital role, and it is assumed that it is supported by a WiFi network inside the university campus, characterized by a capacity that is widely variable during the day. It is supplemented by a broadband connection of limited capacity for off-campus activity (i.e. homework).

In a BYOD context resources are *partially defined* since personal devices may run any of the available operating systems in any of the available versions and revisions: a valid solution should be seamlessly portable on any device that can be assimilated with a personal computer. This brings our discussion to consider software resources.

Besides their availability, software tools that need to be installed by the user should exhibit the following features:

- **robustness** – here intended as *short term*: the student must be confident that software tools will behave as expected during the lecture, or during homework, and that they will resist to misuse;
- **inexpensive** – and manufacturers are in fact inclined to provide free plans to diffuse their products;
- **easy to install** – since installation is either left as a homework, therefore unattended, or carried out during a laboratory lecture, in which case it needs to be successful in most cases in a few tens of minutes;
- **easy to use** – but with a graceful learning slope; in fact students are initially not proficient in the technology they are using but, at least in our case of study, they should develop some ability.

Given the requirements, let us see the kinds of task that can be demanded to the Cloud. Two basic aspects are distinguished here: one strictly aimed to provide tools that are useful to exercise the course topics, another for the realization of the

virtual classroom, that expands and reflects the real class-room in the Web domain. In both cases we are going to see that there is a cross-fertilization between the technologies that are used, and the purpose of the course.

As for the first aspect, each student needs to have a network to play with. This statement extends to nearly all levels in the Internet protocols stack: from link level to application level. Even if a specific course may focus on one of these levels, in practice it is impossible to ignore the existence of others. Thus, a teaching approach that ignores the inter-dependencies among networking layers does not adhere to reality, while student's experience should be close to what is found in the real world.

For instance, consider the case of a laboratory activity consisting in the implementation of a TCP client/server pair in a language of choice. A realistic *hands-on* experience requires that the student faces the problem of deploying the software and of launching it on a remote site: both problems have their place in the engineering of a *real world* solution. Using the loop-back interface to see TCP at work hides all the details regarding the network layer, as well as the access to the server to deploy the software, and the student has a superficial and somewhat misleading experience of relevant aspects of network programming.

An answer to the demand of a *realistic networking experience* comes from virtualization techniques. This is one of the driving factors of Cloud Computing: the provision of a resource "on demand" (a key word in cloud computing definition [7]) becomes realistic when it is assembled using raw resources, not when it is already in stock. In addition, as a side effect, the student that uses virtualization techniques to implement a networking laboratory, although shielded from technical details, gets in touch with topics that play a relevant role in information technology.

Once each student has a virtual network to play with, the following step is to connect it to the Internet. This is preliminary to the implementation of the *Virtual Classroom*, a virtual place where the teacher makes available the tools needed to carry out practical activities, and where the students deliver their results to the teacher. It is not a replacement of the physical classroom, in the spirit of [5], but extends communication beyond our senses through the Internet, more like [11].

Consider again the example of the client/server laboratory. In this case the teacher shows lines of code on the blackboard, and the students type them on their PCs: it is not much different if they are copied from a USB stick, or downloaded from a Web page. Indeed, this is not the way in which programmers exchange IT artifacts: an opportunity to practice *real-world* tools [10] is lost.

In the next sections three categories of products are studied that may help in the implementation of a *Virtual Classroom* in our course. To give a practical slant to our discussion, for each case of study one commercial product is indicated as a reference; however, the market is rapidly evolving and new products are going to appear in the near future, and trademarks will change. The case studies are based on the experience gained with two courses recently given at the University of Pisa, in Italy: one specific on hands-on activity given to students of the "Computer Science" course (https://sites.google.com/site/laboratoriodireti/), the other covering both blackboard lectures and hands-on activity on the same subject to the students of the "Humanities Computer Science" course (https://sites.google.com/site/telematicaunipi/).

A. Provisioning of a virtual network

Among the applications that allow the user to virtualize resources, e.g. User Mode Linux, that has been successfully used for purposes similar to those dealt with in this paper [8], VirtualBox (https://www.virtualbox.org/) by Oracle, Virtual PC by Microsoft, to name only those that specifically target the utilization of a single host computer.

These products not only allow the user to virtualize several *guest* computing devices, possibly running a different operating system, on an ordinary *host* computer, but they also allow to virtualize an Ethernet network connecting together the guests and the host, and to bridge guest connections across the network interface of the host. Using such tools it is possible to synthesize a virtual network that is useful for educational purposes.

However, the management of a group of virtual entities to form a network has long been a task requiring specific skills. For instance, realizing a virtual computer with VirtualBox requires the creation of a box with given hardware capabilities, network interfaces included. Next the user imports a hard disk image with the Operating System of choice, or installs one from scratch. Networking requires the definition of a new domain, typically the configuration of a DHCP server, and the link between the virtual Ethernet and the virtual interfaces. Each of the above tasks may exceed student's competence, so that he should blindly follow a "recipe" given by the teacher.

The Mininet is a successful open source tool that simplifies the task of configuring a network of virtual machines running on a single PC [6]. This same operation is needed to set up development test-beds in real production environments, and a relevant effort has been spent in recent years to obtain simple, and reliable, ways to perform that task, which is now called *provisioning*. So that production-grade tools are now available: let us see how one of them — Vagrant (https://docs.vagrantup.com) — can be used for teaching purposes.

Once the hard disk image containing the operating system — a single file sized hundreds of Megabytes — and the configuration script — the *Vagrantfile*, a piece of *Ruby* code that can be easily embedded in a Web page to allow *cut and paste* operation in the classroom — are ready, the creation of a sandbox network is a matter of single command. Most cloud storage providers offer free plans that exceed the size of a hard disk image, and the *Atlas* (https://atlas.hashicorp.com/) service (formerly *VagrantCloud*) simplifies the reference to the hard disk image, independently from the provider used to store the image. Let us see the interplay of the various services, as shown in Figure 1.

A working example is in Table I: it implements two guest virtual machines, both connected to the Internet through a bridge across the network interface of the host PC, and sharing a local Ethernet network with the host PC (see Figure I). It is written in Ruby and it is split into two stanzas, one for each guest virtual machine. Once the teacher has implemented

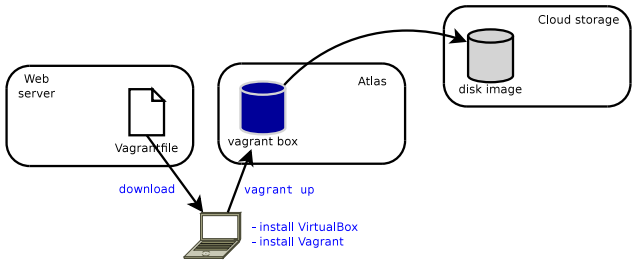

Fig. 1. How several services contribute to the provision of a sandbox network, and the actions of the student to implement the virtual network

```
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.define "pc0" do |pc0|
    pc0.vm.hostname = "pc0"
    pc0.vm.network :private_network, ip: "192.168.5.2"
  end
  config.vm.define "pc1" do |pc1|
    pc1.vm.hostname = "pc1"
    pc1.vm.network :private_network, ip: "192.168.5.3"
  end
  config.vm.box = "labreti/labvm"
  config.ssh.forward_x11 = true
end
```

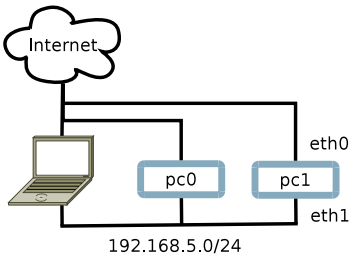

TABLE I. A *Vagrantfile* AND ITS DEPLOYMENT

the solution, the creation of the network sandbox on student's premises consists of installing Vagrant and VirtualBox, copying the *Vagrantfile*, and launching the `vagrant up` command from a terminal.

The minimal example in Table I is functional, and fits the purpose: the two virtual machines are reachable using a remote shell (like `ssh`), using the dedicated IP addresses, just as if they were remote.

Vagrant and VirtualBox conform to the requirements stated in the introduction, since they are distributed for free, and are available for all major operating systems. They are widely diffused, and this is a guarantee of stability and robustness.

B. A Git repository for training coding skills

Coding is one of the activities carried out during a networking lab lecture. Students and teachers manipulate and distribute source code: just like in a production environment, at the core there is a revision control system. However, the data flow is different from that of a traditional development chain, where there is one single repository from which contributors download the code, and occasionally produce an update. A teaching environment is closer to an *open source* programming environments: students download and alter the code distributed by the teacher using a mechanism called *branching*. Each student owns a different branch, that coexists in the same repository of teacher's source, or is recorded in a new repository.

There are several Revision Control System but not all are equally effective: one that is currently receives growing interest is *Git* (http://git-scm.com/) [10], and cloud services exists that are specialized in hosting *Git* repositories.

The distributed nature of *Git* meets our use cases: there is no centralized repository, and branching is implemented effectively, as well as the creation of an entirely new repository. In a typical scenario, the teacher delivers the software, that may be a practical demonstration, but also an incomplete or buggy application to exercises programming skills. The students are asked to interact with the code received from the teacher, which may remain local inside student PC, until when it is shared with the teacher, or with the colleagues, as it would happen in the case of a collaborative project. At that point the virtual classroom is again the place where the sharing takes place, and the student uses the repository hosting service to share his work, since the user of the service can create a new remote repository in his own account.

In our typical scenario, the teacher and the students all have personal accounts, and the teacher *pushes* demos and exercises as individual repositories. The students in their turn *clone* teacher's repositories, and perform their activities on the local copies. Using Git capabilities they can try alternative solutions, restore checkpoints, and share their work with others creating a new repository and *pushing* their solution: the teacher will be granted access to the students repositories when needed.

One company that offers a service that meets our requirements is Atlassian, with the *BitBucket* (https://bitbucket.org/) service. Each user is allowed an unlimited number of projects, that are distinguished in *public* and *private*. Teacher's repositories typically fall in the former category, while students projects are generally private. To take advantage of the free plan offered by *BitBucket* a user cannot share his private repositories with more than four other users, which limits the size of a team to four students plus the teacher.

Given its popularity, *Git* is supported by many programming tools, Integrated Design Environments (IDE) included: for instance, Eclipse has an official plugin for it. Not only exchanging an Eclipse project across a *Git* repository is far easier than sending a source jar by email, but the student experiences the utilization of tools that are effectively used in production environments.

To conclude this section, in Figure 2 the work-flow of a complex activity is shown: a learning assessment consisting in debugging a program. On the left hand the activity of the teacher that writes the correct program as the master branch in a local repository is shown. Next she injects programming errors and records the buggy version as a branch of the repository; this latter is pushed as the master branch in the repository hosted by the public service. A link to this repository is passed to the students together with the assignment. Each student clones locally the repository, and performs the assignment, the last step consisting in the creation of a new repository, shared with the teacher. To assess the performance of the student, the teacher clones the student's repository, and checks the solution.

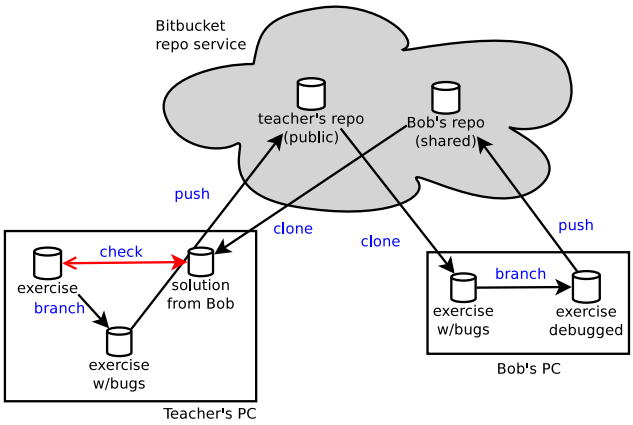

Fig. 2. Using a service hosting a Git repository for learning assessment

```
<html>
<head>
<title> Datagrams computer </title>
</head>
<body>
<h3>Datagrams computer</h3>
<p>
Compute how many datagrams to send a stream (MTU = 576)
<form name="numbytes">
Number of bytes in the stream:
<input type="text" name="nbytes"
       value="" onchange="compute(this.value)">
</form>
</body>
<script>
function compute(bytes) {
packets=Math.ceil(bytes/(576-(20+8)));
alert(bytes+" bytes are encapsulated in "+packets+" packets")
}
</script>
```

TABLE II. AN ACTIVE HTML PAGE WITH A SIMPLE SCRIPT RUNNING ON THE *gear* AT HTTP://DYNWP-ACIUFFOLETTI.RHCLOUD.COM/

C. Offshore experience on a PaaS

A hand-on course about networking must include an *open* Internet experience, outside the boundaries of the campus: the availability of Platform as a Service (PaaS) providers opens the way to this kind of experience.

A PaaS provides the user computing resources configured to support a specific task. Providers offer a range of tools to differentiate and configure the capability of the involved computing resources. The basic options offered by most providers allows carrying out a number of interesting didactic activities, focusing on web services and dynamic web pages, but potentially expanding to other kinds of network functionality. A provider that offers a service fitting the needs of a rich educational experience is *OpenShift* (https://www.openshift.com/)

In *OpenShift* terminology, the computing resource is called *gear*, and roughly corresponds to a server for which the user has a non-privileged access. A *gear* is configured, by default, as a plain Web server, but the user can add further capabilities packaged as *cartridges*. This operation is carried out either through a Web browser, making access to the OpenShift Website using the user's credentials, or through a command line interface. The web interface is probably the most appropriate for our case, since the user — in our case a student — is able to configure a server with a few mouse clicks, and a very limited training and awareness.

One relevant aspect of this operation is that the network bandwidth consumed during the creation and configuration of an *OpenShift* gear has a negligible footprint on the local network: software packages and hard disk images are not downloaded locally — an activity that would deteriorate network performance — but are managed on provider's premises. So that the configuration of the remote server could be the subject of a *hands-on* class, instead of being left as a homework. The educational institution does not have to deploy any additional infrastructure for the virtual machines.

The counterpart for having a configured server in the Internet for free is that the user has no administrative privileges on the *gear*: notably, the user can neither install software packages, nor create Internet sockets (unless those managed through the installed *cartridges*). However, there is a wide range of exercises that can be carried out under such limitations, that have the side effect of protecting the sandbox from misuse or intrusion.

The user populates the *gear* with custom functions that are uploaded using the *Git* version control system: the user replaces the (default) repository recorded in the gear with customized content. This may consist of the engine of a Web service, an active Web page, or the content of a database. The operation consists of cloning the remote repository in the local computer, and later pushing the code on the repository located in the gear. Depending on the application, this operation can be carried out using *Git* terminal commands, or using an integrated development environment.

As it is frequently the case for Web services, *OpenShift* offers a free plan consisting in the provision of a maximum of three *gears* that are publicly accessible: they have a DNS name associated with them. This exceeds what is needed to practice the deployment of a service in the real Internet, with an affordable complexity. More important, the student is able to put in place the workbench with a minimal effort, and makes a *hands-on* experience with the kind of tools that are used in production environments. The simplest activity that can be carried out using a PaaS involves reactive Web pages, an HTML file containing some *Javascript* code: the example in Figure II shows how simple it can be.

However, the *gears* may be equipped with sophisticated software tools: from an educational point of view the most interesting probably are the following, listed according with an increased maturity requested to the student:

- **content management systems** – like Maven, or WordPress, a tool that is often used for training site managers;
- **database systems** – including NoSQL databases that are the first choice for large amounts of data
- **enterprise bus systems** – like JBoss, for an experience in middleware design.

What is notable here is the simplicity with which these sophisticated tools are prepared: in the case of *OpenShift* one click is sufficient to have the *cartridge* installed on the remote machine. It is equally simple to download from a similar gear an example prepared by the teacher: in a matter of minutes the student *clones* teacher's *gear*, and later it may update the

code and *push* the result on its own *gear*.

III. Three examples

In this section three examples are shown of how a virtual classroom that can help during a frontal lecture. Students have their laptops, and they have an Internet connection with a bandwidth that is sufficient for Web browsing. The examples are based on tools and services that have been introduced in the previous section.

The preparation of student's laptops is described in the example, but performed outside class hours: this option saves lecture time and avoids a relevant peak of network traffic, but entails that the operation must not require specific skills.

These are the *first* practical lectures, therefore the most critical ones, of three distinct courses each focusing on one of the networking layers: network, transport, application. The teacher practices some elementary concepts already introduced in previous lectures, and prepares the grounds on which to build other training lectures: simplicity is a requirement since technical details may divert the attention from the topic.

A. Network layer: packet sniffing

This a training lecture that may be part of a *classical* computer networking course. The teacher has already explained the basics of the IP and TCP protocols, and it is now time to see them at work.

The lecture should have been prepared at home by each student: during a preliminary lecture the teacher illustrates the three packages that are going to be used, that later the student installs on his own premises. The sequence of steps is the following:

1) download and install *VirtualBox*
2) download and install *Vagrant*
3) download the *Vagrantfile*
4) launch the command `vagrant up` from a console, and wait.

During the lecture the teacher shows how to access the virtual machine (`vagrant ssh`) and illustrates the virtual network, for instance using basic console commands like `ifconfig`, `ping`, `route`. Using *Wireshark* (which should be already installed on virtual machines) the teacher demonstrates the protocol used by ICMP Ping, illustrates the IP header and how Wireshark can filter based on header content.

Next the teacher introduces the `nc` command, and how it can be used to put in place a simple TCP client/server pair between two VMs. Finally, the students are asked to produce the trace of a three-way handshake.

The interested reader finds the Vagrantfile in Figure I, or in https://sites.google.com/site/tnitcloud/ .

B. Transport layer: Java sockets

The following training lecture might conclude the explanation of the basics of client/server programming. As in the former case, the teacher asks the students to prepare their computers for the activity: besides the tools seen in the previous example, the students are requested to install the *Eclipse* integrated development environment, with the *Egit* extension to manage Git repositories, and to subscribe to the free plan of the *BitBucket* repository service.

The practical lecture starts downloading a TCP client/server pair written in Java: this is obtained cloning a repository indicated by the teacher inside the Eclipse IDE. The Java code may be incomplete or contain bugs, and the students are asked to produce a working version.

To test their products, the students compile the code and save the runnable jars in the sandbox directory containing the *Vagrantfile*, so that they are available in each of the virtual machines in the `/vagrant` directory. For instance, on server side:

```
bob@hislaptop:~/home/sandbox$ vagrant ssh pc1
...
vagrant@pc1:$ java -jar /vagrant/Server.jar
```

The interested reader finds a link to a repository with code to play with at https://sites.google.com/site/tnitcloud/.

C. Application layer: Javascript in a Web page

This lecture can be included as a *hands-on* exercise in a course on the implementation of dynamic Web pages: it does not require a *network intensive* preparation of the device, so it can be presented *on the fly* during the second half of the lecture.

As a first step the teacher invites the students to subscribe for a free plan to OpenShift. Next the teacher demonstrates how to create a *gear*, and indicates the repository containing the code of the dynamic page. During the creation of the *gear* the students indicate this repository as the source of the content. The interested reader can try with the repository indicated at URL https://sites.google.com/site/tnitcloud/. Now each student has a Web server reachable with the browser with the dynamic page installed, and it is possible to inspect the traffic generated by a request, starting the DNS query.

The lecture can be extended *cloning* the content locally (this requires the installation of *Git*) and modifying it. For instance, adding another box for the MTU, or extracting the Java code in a .js file. The modified the code is later pushed on the *gear* with a `Git push` command.

IV. Discussion and conclusions

Cloud technology has the potential to improve the quality of our teaching, reducing at the same time the investment in the computing infrastructure of educational institutions. The paper shows that it is possible to extend a traditional classroom with a *virtual classroom* that the students reach using their own devices. In this sense the virtual classroom extends, and does not replace, the face to face experience of a traditional classroom.

Starting from experience, examples of various facets of teaching of a *Computer Networks* course are shown, highlighting how the virtual classroom takes advantage of *Cloud* technologies. In the examples, the student uses a *private IaaS cloud* for training, finds didactic resources through *public SaaS*s, and experiences Web technologies using a *public PaaS* provision.

There is a strong impact on the quality of the course, that can be split into the following two components.

One is in common with any other e-learning experience, and can be summarized as follows: the classroom does not dissolve when the lecture finishes, but persists as a group supported by shared Web resources. The quality of shared resources improves day by day, and it is a task of the researcher to indicate new directions for this evolution [4]. This paper contributes by adapting to a teaching environment some resources that have been created for a different purpose, and thus encourages the providers to adapt in their turn their offer to an educational environment. For instance creating a commercial offer for a software repository that meet the needs of a virtual classroom. New kinds of shared resources, some of them discussed in this paper, are emerging besides traditional ones: software repositories, on demand platforms and infrastructures have an impact in the future of education, and might be the next frontier for the Moodle.

Another aspect is apparently specific for our use case: for the courses in networking discussed in the paper the tools have been found that bring the students into a realistic experience. This is relevant since the content of the course is strongly technical, and it is desirable that the student has a realistic experience, not mediated by *didactic* expedients. In other words, showing an animation of a three-way handshake is of little purpose compared with capturing and inspecting real packets in a real network. While it is true that the case study and the examples are very specific for the purpose, nonetheless the approach can be extended to any course with a computational flavor. It is often possible to find in the Internet the resources that allow a realistic practical experience, that meet the requirements announced in the introduction: robust, inexpensive, ease to install and use. For instance, consider a course in "technical writing": the *in the cloud* option might envision the use of an on-line word processor, like the Google Documents one. As in the case explored in this paper, the students might use their own equipment, and use a uniform learning platform.

The introduction outlines the requirements for cloud services and network infrastructures. Such an analysis, that is based on experience, is of help for two other categories of ICT professionals: cloud developers and providers on one side, and managers of educational infrastructures on the other.

Cloud developers and providers know that education is a promising market: two examples, besides those extensively analyzed, in this paper are Moodle, that concentrates on the social and management of an educational institution, and Codio, that provides a programming IDE in the cloud and specifically addresses education institutions. The relevant features of a successful educational tool in the cloud have been pointed out: short term robustness, low cost, easy installation and utilization. They are clear indications for the emerging market of educational services.

On the side of *infrastructure management*, the examples given in the paper show that there is a relevant organizational impact of an *in the cloud* education, since it almost entirely relieves the educational institution from the burden of managing computing devices, concentrating instead on the aspects that allow the students and the teachers to use their own devices in the institution premises. The accent is therefore on the network infrastructure, that should allocate bandwidth on a *per classroom* basis, possibly taking into account the network load expected from a given lecture: just as the today teacher asks for a classroom with an electronic blackboard, tomorrow he should be able to ask for a 10 Megabytes/second per student. Similarly, the power supply should be available for each student. Such requirements are technically feasible, and have a moderate economic impact, and their relevance is often underestimated.

Examples have been introduced to show that the tools to implement a *virtual classroom* are available for free: some effort is needed to compose existing services and to adapt their use to the expected competence of the students. In the future, the emergence of products that specifically address teaching is expected, with needs similar to those of *open source* development.

V. Bibliography

References

[1] Ka Ching Chan and M. Martin. An integrated virtual and physical network infrastructure for a networking laboratory. In *Computer Science Education (ICCSE), 2012 7th International Conference on*, pages 1433–1436, July 2012.

[2] F. Gomez-Folgar, R. Valin, A. Garcia-Loureiro, F. Pena, and I. Zablah. Cloud computing for teaching and learning MPI with improved network communications. In *Proceedings of the 1st International Workshop on Cloud Education Environments*, pages 22–27, 2012.

[3] Agustin de La Varga Gonzalez. The importance of "cloud education" at development organizations. In *Proceedings of the 1st International Workshop on Cloud Education Environments*, pages 1–6, 2012.

[4] Keith Jeferry, George Kousiouris, Dimosthenis Kyriazis, Jrn Altmann, Augusto Ciuffoletti, Ilias Maglogiannis, Paolo Nesi, Bojan Suzic, and Zhiming Zhao. Challenges emerging from future cloud application scenarios. *Procedia Computer Science*, 68:227 – 237, 2015. 1st International Conference on Cloud Forward: From Distributed to Complete Computing.

[5] A. Krukowski and I. Kale. Virtual classroom. In *Advanced Learning Technologies, 2001. Proceedings. IEEE International Conference on*, pages 279–282, 2001.

[6] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[7] Peter Mell and Timothy Grance. The NIST definition of cloud computing. Technical Report Special Publication 800-145, US Department of Commerce, October 2011.

[8] Maurizio Pizzonia and Massimo Rimondini. Netkit: Easy emulation of complex networks on inexpensive hardware. In *Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, TridentCom '08, pages 7:1–7:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[9] J. Prieto-Blazquez, J. Arnedo-Moreno, and J. Herrera-Joancomarti. An integrated structure for a virtual networking laboratory. *Industrial Electronics, IEEE Transactions on*, 55(6):2334–2342, June 2008.

[10] Gowtham S. Revision control system (RCS) in computational sciences and engineering curriculum. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, XSEDE '14, pages 76:1–76:3, New York, NY, USA, 2014. ACM.

[11] V. Tam, A. Yi, and E.Y. Lam. Building an interactive simulator on a cloud computing platform to enhance students' understanding of computer systems. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, pages 154–155, July 2013.