International Journal of Advances in Computer Networks and Its Security– IJCNS
Volume 5: Issue 1 [ISSN : 2250-3757]
Publication Date : 30 April, 2015

DETIboot: A fast, wireless system to install operating systems on students laptops

Carlos Faneca, José Vieira, André Zúquete, João Cardoso

***Abstract*—This work presents a system conceived to deploy temporary Linux systems into an unlimited number of client hosts using a Wi-Fi source station (DETIboot server). The ultimate goal of this system is to provide a simultaneous and just-in-time installation of a custom Linux distribution on several tens of laptops for being used in classes or exams. DETIboot uses a server to endlessly broadcast a custom Linux distribution at maximum Wi-Fi transmission speed, using an ad-hoc network topology to reach all nearby target systems wishing to install it. To deal with packet losses and avoid feedback from the client hosts, we used Fountain Codes. With these codes, client hosts can start at any time the reception and the expected time for completing the download is mainly a function of the number of codewords (wireless frames) effectively received. Field tests were done to evaluate the performance of our system and in average it took around 69 seconds to download a custom Linux image (based on Slax) with a size of 225 MiB.**

***Keywords*—operarating systems, ad-hoc networks, broadcast, fountain codes, LT codes**

I. Introduction

With the improvements in computer technology in the last years, laptops have become powerful enough to satisfy the storage, communication and processing needs of most of the users with the advantage of the mobility, which is not possible with desktops. So, nowadays, personal computers are mostly laptops instead of desktop computers.

Although many students prefer to use their own laptops in university labs, most of the universities still equip them with desktop computers, which is a waste of resources in maintenance and energy consumption. The main reasons to maintain this situation are twofold, i) some software licensing depends on specific machine characteristics (e.g. IP address) and ii) some exams are performed on the desktop computers.

For the exams, teachers have privileged rights over the desktop operating system, giving them the possibility to restrict the access of the students to external sources of information. In general, this is not possible to enforce when student's laptops are used.

To overcome these issues we considered an approach where, in particular scenarios, the operating system of the students' computers is provided by teachers. With this approach, teachers could either (i) deploy licensed software on operating systems that could prevent its irregular exploitation outside the classroom or (ii) deploy hardened operating systems containing only the facilities and tools required to participate in an exam, and nothing else (i.e., following the principle of least privilege [1]).

To deploy the same operating system in many different hardware platforms is by itself a challenge, but the current operating system installation procedures are very powerful on this task and usually handle it transparently to users (i.e., with very few technical questions). On the other hand, many installation problems typically arise when operating systems have to deal with uncommon, external devices and not with the hardware that one would ordinarily find inside laptops.

Another more difficult requirement is that students should not be able to get the operating systems in advance because that would give them the advantage to study it and the possibility to overcome the deployed protection mechanisms. If possible, operating systems should be downloaded to client hosts just-in-time, i.e., immediately before being required, and should not remain in the client host upon being used. This poses a challenging requirement: how can we transfer an operating system installation image to tens or hundreds of machines in a very short time frame (say, a few minutes) to enable this approach to be practical for classes and exams?

DETIboot [2] was designed to solve this problem, by giving to the teachers the opportunity to easily deploy a custom Linux operating system, configured by their own, to run in students laptops. DETIboot uses a server to endlessly broadcast the operating system over an ad-hoc network, using Fountain Codes to deal with packets losses on the Wi-Fi channel and avoid the use of a feedback channel from clients to packet acknowledgment. A generic, first-stage boot operating system was also developed for receiving the transmitted file and boot then from it. Both operating systems run from a RAM disk and at the end they vanish without leaving any trace on the student's laptop. In this paper we

Carlos Faneca
University of Aveiro / IEETA
Portugal

José Vieira
University of Aveiro - Department of Electronics, Telecommunications and Informatics / IEETA
Portugal

André Zúquete
University of Aveiro - Department of Electronics, Telecommunications and Informatics / IEETA
Portugal

João Cardoso
University of Aveiro / IT
Portugal

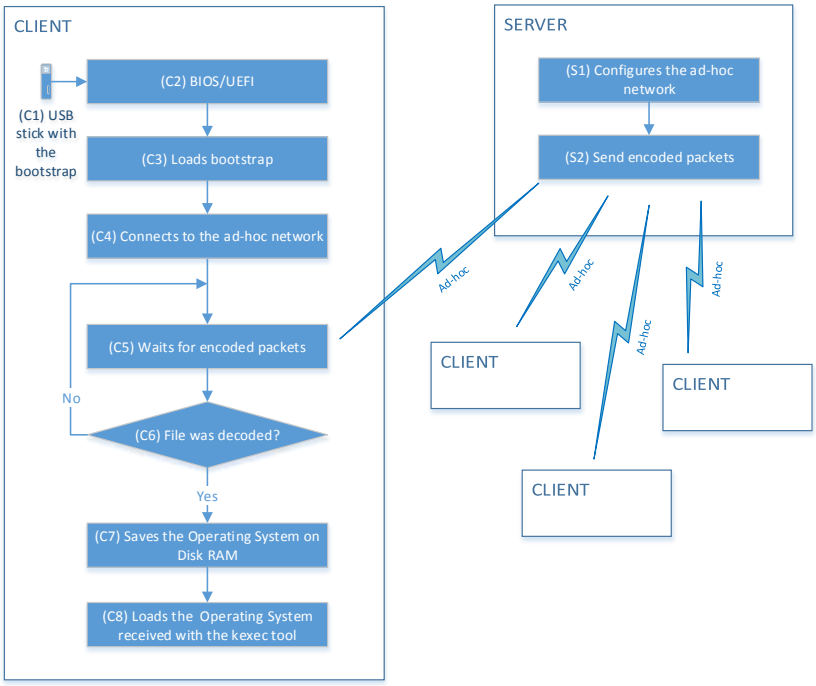

Figure 1. Block diagram of the architecture of the DETIboot system, with the high-level steps carried on by each participant.

provide a high-level description of DETIboot (further details can be found in [2]) and we show practical results achieved with a low cost setup on a typical class room.

II. The DETIboot system

A. *General architecture*

DETIboot is a system that enables a quick, temporary deployment of a Linux live operating system through a Wi-Fi network [2]. The architecture of the DETIboot system is based on a client-server model, where there is a node on the network (server) responsible for transmitting the operating system. The remaining nodes in the network act as clients; they receive the operating system transmitted by the server and boot it after the transfer is completed.

Figure 1 shows a block diagram with the architecture of the DETIboot system and the high-level steps carried on by each participant.

B. *Network*

As Linux images can have several hundreds of megabytes, it is desirable for DETIboot to have a high transmission rate. In order to achieve this, we started by minimizing the requirements and responsibilities of the server.

The use of the Wi-Fi network in structured mode would have meant that the server would act as an Access Point, being responsible for the overall management of the network, such as IP management and client registration. Thus, it was decided to use an ad-hoc network, freeing the server from these tasks. This way, it is only responsible for broadcasting the operating system to all nearby clients. This network mode enables the server to communicate with multiple clients simultaneously and allows clients to enter and leave the network at any time without disturbing the transmission of the server, since there is no need for an explicit network association.

Finally, DETIboot does not even uses IP, since there is no need to route packets from the source (server) to the destination (clients); all intended clients are meant to be within the Wi-Fi radio range. Therefore, DETIboot uses directly broadcast link-layer frames (with an unused EtherType code) to transmit operating system images.

The conventional Wi-Fi broadcast data transmission is held to the minimum rate (1 Mbit/s), in order to maximize compatibility with all system and increase the reception probability (because received frames are not acknowledged). This transmission rate is defined by the “basic rate” and we had to modify it, since our system is intended to broadcast at the maximum possible rate. This modification can be accomplished in a Linux system at the network interfaces’ level. However, when we forced the “basic rate” set to the 802.11g maximum speed (54 Mbit/s), we verified that we could not get it with many Wi-Fi interfaces (and their drivers); instead, they forced a lower maximum speed, nevertheless higher than 1 Mbit/s. This occurs because the standard specification for ad-hoc network mode is fairly free, being the only requirement to not interfere with structured networks. Consequently, each chipset has its own properties according to the manufacturer; it is even possible that the same Wi-Fi interface exhibits different behavior while using the same driver but different versions of Linux.

Table I shows the maximum effective data rate achieved with DETIboot with several PCI and USB Wi-Fi interfaces.

The values presented are measured by the broadcasting application, not by clients.

TABLE I. WI-FI BROADCAST SPEEDS ACHIEVED WITH DIFFERENT PCI AND USB NETWORK DEVICES (MEASURED BY THE TRANSMITTER).

Model	Type	Chipset	Driver	Transmission
MicroNext	USB	---	RTL8192CU	1.6 Mb/s
LMtec	USB	---	RTL8192CU	1.6 Mb/s
MicroNext mini	USB	RLT8176	RTL8192CU	1.6 Mb/s
LMtec 300Mbs	USB	RTL8191s	R8712u	---
Belkin N150	USB	RTL8191s	R8712u	---
Wi-Pi	USB	---	RT2800usb	33 Mb/s
AR5007	PCI	AR5007	---	33 Mb/s
RTL8187b	PCI	RTL8187b	RTL8187b	43 Mb/s
Thomson TG123g	USB	RTL8187b	RTL8187b	43 Mb/s

For the setup that we will evaluate in this paper we chose a small, portable and powerful host for acting as DETIboot server (ODROID-U3). Since it lacks a Wi-Fi interface, we chose to explore it with a Wi-Pi USB interface (intended for the Raspberry Pi). We didn't use the Thomson 123g because it is no longer being produced.

C. *Server*

The role of the DETIboot server can be performed by any Linux-based computer with a Wi-Fi interface since the "basic rate" can be configured for high speed broadcast on an ad-hoc network. Due to its low cost (65 US$), high computing power and excellent portability, we chose the ODROID-U3 system[1] (1.7 GHz Quad-Core ARM processor, 2 GiB of RAM, see Figure 2), running the Xubuntu 13.10 Linux distribution.

In the rest of this section we will describe the tasks performed by the DETIboot server, as shown in Figure 1.

1) (S1) Configures the ad-hoc network

The first operation performed by the server is to configure the ad-hoc network to broadcast the operating system. When creating the ad-hoc network, the server first scans the transmission medium in order to detect surrounding networks and automatically chooses the Wi-Fi channel for the DETIboot system. The channel choice is important, since we want to use a channel that will not interfere with existing nearby Wi-Fi networks (since the server will perform a very aggressive medium occupancy and performs better on a free channel). The server's channel selection policy is to choose a free channel, preferably, surrounded by the less powerful detected channels. Finally, the "basic rate" of the Wi-Fi interface used in the ad-hoc network is set to the maximum possible rate (54 Mbit/s).

[1] www.hardkernel.com

2) (S2) Send packets with codewords

To broadcast the operating system image (a file), a program was developed that receives as input the file name and the output Wi-Fi interface. This program creates codewords from file blocks (symbols) using Fountain Codes [3]–[6] and broadcasts them (encapsulated) in link-layer packets. Codewords are XOR sums of one or more symbols, where the number of symbols and the symbols themselves are randomly chosen.

The DETIboot server transmits random codewords continuous and endlessly, allowing clients to initiate their download at any time, as they only need to receive enough codewords to recover the original file. Furthermore, this type of coding deals well with packet losses without the need for a feedback channel. However, clients need to receive, on average, 5% more codewords than the total of symbols that composes the original file (depending on the settings used for the creation of the codewords).

D. *Client*

DETIboot clients will typically be laptops, which receive, and boot from an operating system transmitted by the DETIboot server. The minimum requisites to be a client of the DETIboot system are: it should have a Wi-Fi card and sufficient RAM to run the entire operating system on it. The size of the RAM depends on the operating system that is being broadcast, but 2 GiB of RAM is the minimum required. For keeping the client laptop unchanged, the image of the received operating system is never stored on disk.

Clients' laptops tend to be a very heterogeneous set of machines with 32 or 64-bit architectures, different processors and instruction sets, not to mention a plethora of peripherals, network cards, keyboards etc. To maximize the compatibility, we used a custom Linux operating system with all possible drivers in the image in order to increase the probability of the client to work without any problems.

For the USB stick with a first-stage bootstrap system we only have to maintain them with the wireless network drivers updated, having the possibility, if needed, to have different USB sticks customized for each laptop model.

Now, we will describe the process performed by a DETIboot client, as shown in Figure 1.

Figure 2. ODROID-U3 as a DETIboot server.

1) (C1) USB stick with the bootstrap system

To use DETIboot, a client has first to boot from an USB stick with a first-stage bootstrap system, that contains only the startup programs and the Wi-Fi drivers required to receive an operating system image from a DETIboot server. Client owners can create their own USB stick from an image available online.

2) (C2) BIOS/UEFI

The bootstrap is prepared in order to perform a boot from a Master Boot record (MBR) or Enhanced Firmware Interface (EFI), depending on the client laptop hardware. The client only has to change the "boot order" in its BIOS or UEFI to boot from the USB stick, or directly enter the "boot from" menu and choose the USB stick.

3) (C3) Loads bootstrap system

In this step, as the name implies, the DETIboot bootstrap system starts, being in charge of: connecting to the ad-hoc network of a DETIboot server; receive codewords and locally reconstruct the operating system file being transmitted; and finally reboot from the image just downloaded.

4) (C4) Connects to the ad-hoc network

The client ad-hoc network configuration consists in joining the ad-hoc network of the DETIboot server. Note that this is just a matter of local configuration, there is no traffic involved.

5) (C5) Waits for codeword packets

After joining the ad-hoc network, the client immediately starts to receive packets with codewords send by the server. These codewords are used to decode other already received codewords, or are decoded with those, or otherwise stored for both actions in the future.

6) (C6) File was decoded?

The received codewords are decoded into their individual symbols using simpler codewords [2] (i.e., composed by less symbols). Recovered symbols are organized to obtain the original image of the operating system.

7) (C7) Saves the Operating System on RAM Disk

The download process ends when the client possesses the complete operating system image. This image is then decompressed to allow booting from it. Then, the downloaded file is removed in order to save space on the clients' RAM.

8) (C8) Loading the received Operating System with kexec

The startup of the operating system is performed by the kexec tool. This uses a homonym Linux system call that enables rebooting a new kernel from the currently running kernel. Essentially, kexec skips the bootloader stage (hardware initialization phase by the system firmware, BIOS or UEFI) and directly loads the new kernel into memory, where it starts executing immediately.

The remaining boot process, after the execution of kexec, is a responsibility of the imported operating system. Thus, it is necessary to modify the operating system to be ready for this boot by kexec, as discussed in the following section.

E. *Preparing an Operating System for DETIboot broadcasting*

When kexec command is executed, it "cleans" the entire RAM, leaving only the kernel and the initial RAM disk (initrd) of the operating system to perform the boot. So, the initrd temporarily loads a file system that decompresses and loads the OS file system that is usually on a ROM location, like CD drive, USB ports, or network. Thus, the OS file system is lost after running kexec because it only "saves" the kernel and initrd.

The solution is to modify the initrd in order to contain the OS file system and modify it so it searches the OS file system within the initrd. Thus, the OS file system is not lost after executing the kexec command because it is loaded into RAM along with the initrd.

Furthermore, the OS file system is modified in order to be as light as possible and contain the customizations needed to perform the exams, namely: user permissions, file permissions, Wi-Fi access control, restricted web access and the class programs and files for the exam.

III. Evaluation

For the system evaluation we performed field tests for measuring the efficiency of our system and analyzed the impact of packet losses, due to interferences and to the distance between the DETIboot server and the client, in the overall file reception time.

A. *Measurement scenario*

For all field tests we chose a realistic scenario, a classroom with 92.25 m^2, in the Department of Electronics Telecommunications and Informatics (University of Aveiro). The university has eduroam Wi-Fi access all over it, thus this classroom is a noisy Wi-Fi environment. We placed the DETIboot server at the teacher's desk and placed 4 equal laptops clients in the places displayed in Figure 3.

As mentioned before, for the DETIboot server we used an ODROID-U3 with a Wi-Pi interface.

For the clients we used four Asus K54HR laptops (i3-2350M CPU @ 2.30 GHz, 4 GiB RAM) without any extra equipment besides an USB stick with the DETIboot bootstrap system.

In this classroom we found many Wi-Fi networks in the environment and we have recorded their SSID as well as an average value for the Received Signal Strength Indicator (RSSI) observed by each client and by the server. The campus-wide "*eduroam"* network was detected in 3 channels (1, 6 and 11) and their RSSI was around -90 dBm, -60 dBm and -75 dBm, respectively. Other temporary networks were also found on channels 1 (-55 dBm) and 9 (-85 dBm).

The DETIboot server broadcast with the BSSID (Basic Service Set IDentifier) "DETIbootWiFi" and chose channel 11 to do so. Near the server we measured an average channel 11 RSSI of -50 dBm; this value decreased as we move away from

TABLE I. MINIMUM, MAXIMUM, AVERAGE AND STANDARD DEVIATION OF THE MEASUREMENTS RELATED WITH THE DECODING OF A 225 MIB OPERATING SYSTEM IMAGE BY THE 4 CLIENT LAPTOPS PLACED IN THE LOCATIONS PRESENTED IN FIGURE 3.

Laptop	lost packets (%)				max RSS (MiB)				elapsed time (s)			
	min	*avg*	*max*	*σ*	*min*	avg	*max*	*σ*	*min*	avg	*max*	*σ*
1	1.3	1.8	2.4	0.4	473.1	544.0	567.6	35.8	60	62.4	68	2.3
2	3.8	9.2	20.5	5.5	476.5	557.2	572.1	23.1	63	69.9	107	11.5
3	4.2	13.1	37.5	8.6	555.0	563.8	574.0	5.2	63	71	103	9.8
4	9.0	15.3	29.5	5.8	476.2	556.1	579.4	22.4	66	73.1	104	9.6

it, measuring over the client's 1, 2, 3 and 4 an average RSSI of -60 dBm, -55 dBm, -75 dBm and -75 dBm, respectively.

On each reception place we measured, for each complete file download and decoding, the percentage of missed packets, several time figures (elapsed time, user CPU time and system CPU time) and the maximum amount of memory used by the decoding process.

B. *Results*

For analyzing the file downloading and decoding processes, we took 15 independent measurements on each of the four places displayed in Figure 3 for a continuous broadcast of an operating system image (Linux Slax) with nearly 225 MiB. The results obtained are presented in Table II, Figure 4 and Figure 5.

The observed percentage of packet losses is small all over the classroom (below 30%, maximum), and increases with the distance from the DETIboot server (ranging on average from 1.8% to 15.3%). As expected, this degradation will affect the time to successfully decode the OS image, ranging on average from 62.4 to 73.1 seconds (c.f. Table II). The distribution of all observed values is presented in Figure 4.

The relationship between the decoding elapsed time and the percentage of packet losses can be observed in Figure 5. The somewhat linear relationship between these two values is highlighted by the computed linear trend line that can be observed in Figure 5.

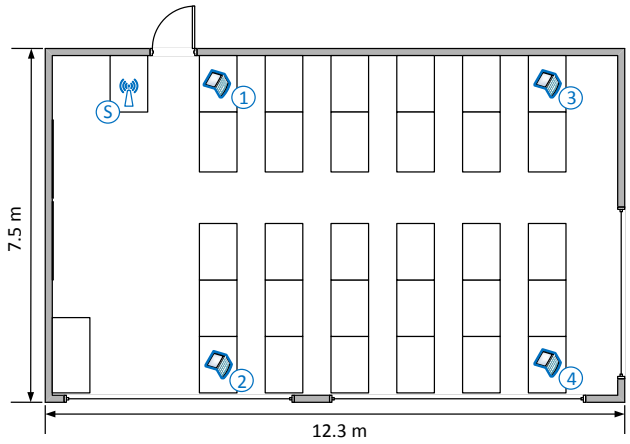

Figure 3. Diagram showing the places where we placed the DETIboot server (S) and the 4 clients in the classroom.

However, the shape of the distribution of decoding times is not only a result of particular loss rates on clients, but also an expected result of Fountain Codes (c.f. Figure 7 in [4]). In other words, higher decoding times are not exclusively due to higher codeword losses, it can happen due to the combination of codewords received. This is a consequence of the random nature of Fountain Codes, which is empirically demonstrated by the test where the client 2 has a packet loss of only 14% but takes 107 seconds to decode the operating system, see Figure 5.

IV. Conclusion

In this work we described an efficient, wireless system to distribute custom Linux images to a large number of laptops in a classroom. This was an engineering work where we have combined different areas of knowledge to achieve an innovative solution to this problem. We have used ad-hoc Wi-Fi networks and a broadcast link-layer protocol with a maximized bit rate, combined with Fountain Codes that allows an efficient broadcast of the data without a feedback channel. We also build a custom boot solution that guarantees a flexible and manageable booting system.

The ultimate motivation behind this project is the realization of exams in a class room using students' laptops, where the teacher must be the administrator of their operating system. Fast and wireless downloading and booting of a custom operating system was the very first step towards this ambitious goal, but many other critical steps need to be given in the future to accomplish our goal. In particular, we need to perform thoroughly tests to find out what students can do to work around the security of this system and fix them. Furthermore, this system requires constant maintenance especially in drivers' updates on the bootstrap and on the custom Linux image to maximize the compatibility with new laptop models.

In section III we evaluated the performance of our system exploring an ODROID-U3 server host. We observed that the percentage of packet losses is relatively small all over the classroom (below 30%, maximum), increasing with the distance from the DETIboot server and ranging in average from 1.8% to 15.3%. A way to decrease packet losses is to change the server Wi-Fi interface for a more powerful one, but nonetheless with Wi-Pi we got great results transferring a Slax Linux image with approximately 225 MiB in 69.1 seconds, on average.

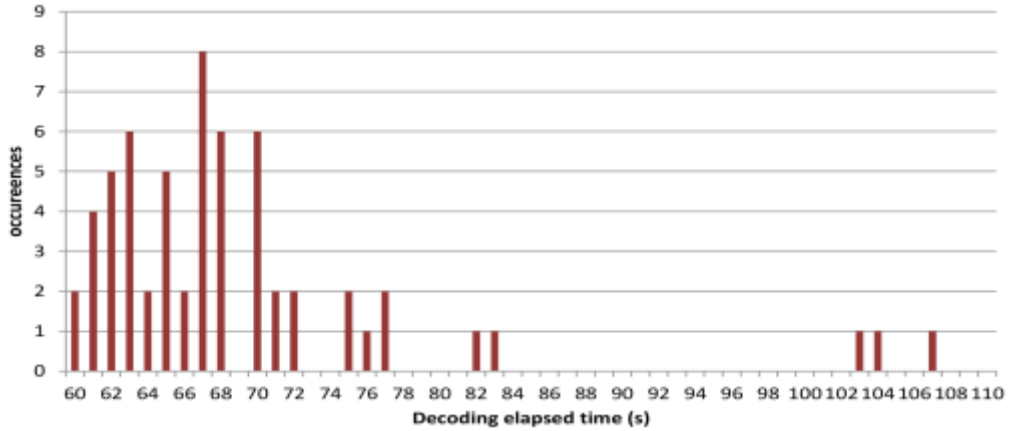

Figure 5. Distribution of the decoding elapsed time considering the observations of all the 4 receiving laptops.

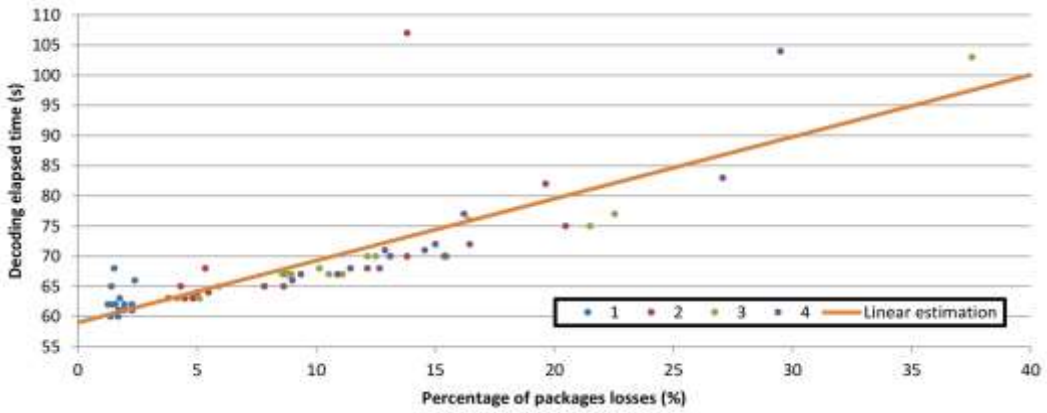

Figure 4. Observed relationship between decoding elapsed time and percentage of packet losses. Dots represent observations in all 4 receiving laptops; the solid line represents a linear trend calculated from those observations.

DETIboot is very attractive to deploy live Linux setups to be used temporarily into a large number of nearby laptops. This is the case, for instance, for giving classes or hands-on tutorials with specially crafted operating system distributions, or to temporarily distribute proprietary software for testing or demonstration. The fact that the system architecture is based on a standalone server station broadcasting through Wi-Fi makes it very convenient to deploy the system anywhere, since it is not dependent of the existing network infrastructure.

Acknowledgment

This research is part of the R\&D project, *CodeStream*, with the reference *PTDC/EEI-TEL/3006/2012* financed by the *FCT - Foundation for Science and Technology*. This Research Unit is funded by National Funds through *FCT - Foundation for Science and Technology*, in the context of the project *PEst-OE/EEI/UI0127/2014*.

References

[1] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[2] J. Cardoso, "DETIboot : distribuição e arranque de sistemas Linux com redes WiFi," University of Aveiro, Aveiro, Portugal, 2013.

[3] L. M. M. M. Byers John W. and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," in *SIGCOMM*, 1998, pp. 56–67.

[4] D. J. C. Mackay, "Fountain Codes," *IEE Proc. - Commun.*, vol. 152, no. 6, pp. 1062–1068, 2005.

[5] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2006.

[6] M. Luby, "LT codes," *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, pp. 271 – 280, 2002.