Network based distribution of the Abitti Platform

Jari-Matti Mäkelä
Department of Information Technology
University of Turku

***Abstract*—The Abitti platform offers a controlled environment for arranging exams, ranging from small exams local to a school to the annual nation-wide matriculation examination. The current version of the Abitti platform relies on local booting of the operating system image from bootable media. We suggest an alternative, complementary boot process based on network booting, which has the major advantage of a relatively quick and simple setup on supported hardware, with decent performance and usability and without the need to transition to a completely new platform.**

I. Introduction

The Abitti platform[1] offers an interesting controlled environment for arranging exams, ranging from small exams local to a school to the annual nation-wide matriculation examination. There has been debate about the decision of picking a system based on a custom operating system image for the task, instead of a solution run on top of a commonly used commercial operating system, usually provided by the computer vendors. Along with locally run applications, solution based on cloud/web computing and network services have been suggested.

While a custom platform introduces costs and risks related to the maintenance, security, reliability and robustness of a system, we argue that the current solution provides at least two significant advantages. First, the platform is based on a single, centrally maintained version of open source software. With vendor provided and retail software, the market is fragmented with multiple versions of operating systems[6] (and browsers in the case a the suggested cloud/web based platform), which complicates testability. Moreover, enforcing a certain policy with compatible operating systems might result in extra costs as commercial operating system or even hardware updates might be needed to meet the minimum requirements. Recommending certain certified commercial products for a nation wide exam may also be subject to legal challenges such as the antitrust laws[7], but it is also ethically problematic for a government agency to advocate commercial products especially when free alternatives are not only available, but already being used for the task.

The second problem comes from the security concerns. Abitti offers a controlled client-side environment and approaching the same level of control may require multiple monitoring solutions such as rootkits or special monitoring processes with elevated access rights to personal data, which raises a multitude of new technical, but also ethical questions. A cloud/web based solution is also a possible alternative and comes with its own pros and cons, but we argue that transition to such a new approach is a long process and requires a significant amount of software development effort beyond a single Hackabi event. However, even with a cloud/web based solution, a custom Abitti distribution could be beneficial as a client-side solution for offering a controlled web interface to the system. In such a scenario, one of the selling points of Abitti is to be able to shut down and hide existing content stored in the device, in external storage or in nearby networks without actually damaging any personal data.

We make the claim that the current operat-

ing model of the Abitti platform still has use in the coming future revisions of the system, and this paper focuses on incrementally improving the current system from a practical point of view. The main goal of this work is to describe a working exam environment that supports the widely known network booting mode. In the next section, we start with a short introduction to the concepts of network booting, followed by a section describing the configuration required both on the client and server side. Section IV describes our new contributions related to the booting and image handling process and finally in Section VI we draw conclusions.

II. Network booting

The current version of the Abitti platform relies on local booting of the operating system image from bootable media such as USB mass storage[2] (multiple options, USB sticks being one of the most common media) or an optical disc (not officially advertised, but the same boot system applies to optical storage). In theory, the image could also be installed on the hard drive, but this option requires an extra pre-installation phase which may further complicate the process of setting up the system during the examination. The main advantages of pre-installation are anyway moot as each exam instance comes with its own distribution image. At first, the current platform appears as a straightforward and simple idea for implementing the system. However, the current solution with removable boot media faces several issues that complicate each instance of the exam process:

1) Expenses (direct, indirect): hardware, training of personnel, time and other resources for preparing, verifying, and securing the boot media.
2) Technical expertise required: administrators, exam supervisors, preparation of the USB media.
3) Technical limitations: slow USB host/-media (especially cheap USB 2.0 sticks, random access for optical discs), no room on the media for multiple versions of the Abitti system for different hardware, media cannot be removed if the image does not fit in RAM.
4) Technical faults: broken media (connectors, scratches, bad blocks, ...), cloned/missing media (can also pose a security risk).
5) Incompatible hardware: USB type C, no USB ports, no optical drive, not enough ports for both charger and USB media and/or ethernet.
6) Software issues: BIOS (can't access boot menu, don't know how to use, passwords, secure boot, legacy/UEFI boot, no boot option for USB), OS (broken ACPI, incompatible kernel / drivers, incorrect driver configuration), etc.

We suggest an alternative, complementary boot process based on network booting[11] (netboot). The netboot system is a traditional client-server approach to computing in a local area network, but it supports distribution of services to some extent. In this context, we define the workstation executing the Abitti system as client side and the servers associated with the system as server side. The main idea of netboot is to be able to start the operating system on diskless clients without any need for local installation media. This greatly simplifies the burden of maintenance on the client side and naturally affects the expenses as physical media is not needed nor needs to be prepared for the exam. While by default the netboot will not support WiFi directly, preconfigured WiFi bridges can relay ethernet netboot traffic between wireless links, if needed. We only note that while such systems are possible, WiFi may only offer a limited bandwidth, relatively high latency, high packet loss and such might still be impractical for netbooting clients.

Netboot is often fully supported by the workstation's ethernet hardware without any modification, some hardware require an external ethernet device, and some hardware also require a special removable boot media for activating the netboot. As a final fallback, the current approach with local USB media (or optical media) can be used or if even that fails, the machine is unsupported by the system. On the server side, netboot does not have

any special hardware requirements and only requires setting up a small number of services.

The major advantage of netboot is a relatively quick and simple setup on supported hardware - the system BIOS (basic input/output system) can be set up for automatic network booting or as a one time boot option when connected to networks that offer netboot. When used on "normal" networks with no netbooting set up, the machine starts up the local operating system instead. This eliminates the whole class of issues related to pluggable external media.

III. Configuration

A. Client side configuration

Network booting has been commonly supported in many enterprise and consumer computers for a long while due to the minimal cost of including a network boot ROM on the network chip. The biggest threat to a wide scale support is the disappearance of ethernet ports from modern mobile devices as mobile devices rely more on wireless networks and become too thin to house such large legacy connectors.

In the ideal case, the client computer supports network boot and enabling it only requires a tiny amount of configuration (one time boot option at boot time or boot order configuration from the BIOS). Do note that setting network boot as the default boot option may slow down the normal boot time by a few seconds, even when the machine is booted from a local disk since every boot attempt waits for a reply from the network. This slowdown does not affect the energy saving mode known as 'suspend' or 'suspend to RAM', e.g. when closing the laptop lid. The settings also has obvious security implications as we do not want to network boot from every network. We continue with the discussion of the security of network boot options on the client side in Section V.

In the next best case, the machine's BIOS supports netboot with a supported (PCMCIA/USB) ethernet adapter[1] – this could be a common situation with modern notebooks with USB 2/3 but without ethernet ports. The third option is to use the built-in non-netbootable ethernet or external USB ethernet with a gPXE[9] bootloader provided from an USB media. gPXE is an open source (GPL) network bootloader. It provides a direct replacement for proprietary PXE ROMs, with many extra features.

In the worst case, none of the options work and a bootable local USB media needs to be used instead.

If the netboot is supported, the machine will automatically boot via network and present the user with a bootloader prompt, which allows choosing the best operating system configuration. This process is further discussed at the end of the next section. Once the operating system boots, the Abitti platform should launch without further issues.

B. Server configuration

Here we describe a minimal configuration for the server side environment, which is responsible for a) isolating and protecting the temporary network used for the exam process, b) relaying traffic, c) providing address and name services d) monitoring the workstations, e) offering material to the students, and e) providing a netboot environment for the f) student workstations. A full treatment of the network environment is out of the scope of this paper, instead we focus on the required changes for the netboot environment.

DHCP: The netboot process begins with a special DHCP query[8] from the netboot enabled clients (role f) to the DHCP server (role c). This is a standardized process, but involves special conditionals for choosing the right pre-execution environment (PXE) image based on the extracted info (from the query)

[1]According to several network discussion forums[4], [5], [3], USB ethernet does not use a class based driver, thus each driver needs to be separately supported by the BIOS. Suggestions for better supported chipsets include: SMSC7500, LAN9500, LAN7500, AX88772, 88178. The ubiquitous and cheap RTL8153 based NICs from Ebay may not be supported by older BIOS versions.

about the client architecture, e.g. for 32-bit or 64-bit UEFI[15], or for a legacy BIOS client. An example code for choosing the correct images, respectively, with Syslinux[2] as a PXE environment is shown in Algorithm 1. Similar setup is needed for new clients using low-power mobile devices equipped with the ARM processor architecture. Downloading of the image files is delegated to a TFTP[14] server, which can be a separate server. Assuming the naming conventions do not change, the same DHCP server configuration can be used between exams without any need to change it.

TFTP: The next step in kernel configuration is a TFTP server (role e). TFTP is a simple protocol for downloading files in a local area network. TFTP does not support encryption or authentication, which may have security implications. We discuss possible remedies for the TFTP's security limitations in the Section V. Setting up TFTP is a documented standard process and we skip it here. Configuring the Syslinux and an assortment of operating system images is also a documented standard process. An straightforward example configuration for Syslinux is shown in Algorithm 2.

The idea of a boot menu is to provide alternative operating system configurations (e.g. kernel command line parameters) or versions for systems which have issues with the standard distribution. The alternative versions may only use a different kernel & initramfs combination for sidestepping hardware issues, but more customizations can be provided by customizing the full distribution image. We discuss the process of customizing the distribution images in Section IV.

Rest of the boot process: Once an operating system version is chosen for booting, the boot process continues with normal kernel & initramfs stages on the client side. A typical netboot environment does basic initialization and proceeds with a NFS (Network File System) mounted root filesystem. However, we do not make that assumption and do not rely on NFS. The kernel configuration still requires means for distributing the distribution images (the userspace part), which we implement using a custom solution due to an inherent scalability problem with the other approaches in a large scale booting environment such as the exam process. We discuss the further boot process in Section IV.

[2]http://www.syslinux.org/

IV. Optimizing the image distribution

A large problem in a environment with possibly dozens or even hundreds of concurrently booting machines is the network scalability. Schools may rely on legacy hardware, for instance switches designed for 100BASE-T networks. Assume that an Abitti operating system image has the size of 1000 MB. The exam starts with 100 notebooks prepared for netboot. In this example, the systems must transfer 20% of the image to fully start up the desktop environment (the image can be made malleable in such a way by providing it at file level or block level granularity via NFS or NBD[13] (network block device), respectively). With each client downloading a separate image of the system, 20 GB of data needs to be transferred. In an ideal 100BASE-T network with a single server side interface serving the requests, this may take over 30 minutes. However, in practice network protocol overhead and connection issues can double this time. Modern 1000BASE-T networks are approximately 10 times faster, yet still become quite congested with the data. In practice, not all workstations can store the whole image in RAM, which leads to repeated traffic and also more traffic is needed for each application used in the exam.

A. UDPcast

Our solution is to netboot the kernels and initramfs separately for each client, but for the root filesystem image, we use local area network broadcasting. Broadcasting can reduce the $O(n)$ amount of traffic required for n clients into $O(1)$. This task relies on an existing tool called Udp-sender from the UDPcast[3] package. UDPcast is a open source file transfer tool that can send data simultaneously to many destinations on a local area network. We embed the

[3]https://www.udpcast.linux.lu/index.html

Algorithm 1 Choosing the correct bootloader image type (dhcpd).

```
subnet 10.0.0.0 netmask 255.255.255.0 {
        option routers 10.0.0.254;
        range 10.0.0.2 10.0.0.253;

        class "pxeclients" {
                match if substring (option vendor-class-identifier,
                  0, 9) = "PXEClient";
                next-server 10.0.0.1;

                if option arch = 00:06 {
                        filename "pxelinux/bootia32.efi";
                } else if option arch = 00:07 {
                        filename "pxelinux/bootx64.efi";
                } else {
                        filename "pxelinux/pxelinux.0";
                }
        }
}
```

Algorithm 2 PXE menu for choosing operating systems (Syslinux).

```
label os1
    menu label Abitti oletusversio
    kernel kernels/bzImage-abitti
    append initrd=kernels/initramfs-abitti ip=dhcp ...

label os2
    menu label Abitti uusimmilla staging-ajureilla uusille laitteille
    kernel kernels/bzImage-experimental
    append initrd=kernels/initramfs-abitti ip=dhcp ...
```

recipient part of this tool to a custom initramfs image that is responsible for mounting the root filesystem and the sender part to the server that manages the distribution images. In practice, the tool is used to distribute the largest parts of the boot process, the distribution images, using the UDP protocol. Since the protocol does not do as much error checking as TCP, we double check the result with a checksum tool at the client end and repeat the process in case of errors. A more detailed description of the use of the UDPcast tool on the client side continues in the next subsection.

As to why are the kernel and initramfs not broadcasted, modifying the bootloader to do this requires special code and moreover not all of the clients may pick the same kernel+initramfs combination. It is also more likely that the kernel loading would be interleaved with other tasks (people setting up workstations, BIOS boot routine, boot menu) which makes optimizing this stage less useful. Moreover, we can optimize the initramfs and kernel in various ways.

B. Custom initramfs

The initramfs starts with its own init script that contains heuristics for deciding how much

of the distribution can be downloaded to RAM. Downloading parts of the distribution to RAM means that a tmpfs storage is used to permanently store parts of the compressed (read-only) root filesystem until the machine is shut down. Ideally the whole distribution is downloaded (and even uncompressed), which is the case with most recent hardware, but for some machines, only a part or none of the distribution can be stored in RAM. As a fallback, the server must provide the distribution images via NBD shares so that low memory clients can access them on a block by block basis and cache using the general Linux filesystem cache manager. The process is described in a pseudo code form in Algorithm 3.

The exact minimum memory requirements and size of Abitti change between releases so we do not offer the values here. The distribution is split into parts responsible for the basic startup process, exam applications and so forth. Our suggestion is to use squashfs compressed images which are combined on the client side using the mainlined overlayfs virtual filesystem meta-layer. The last layer has to be a tmpfs in order to guarantee that nothing will be written to the local machine or to any network location. Backing up local filesystem state is also possible, but we do not consider it here – we assume that all important state is typed or uploaded to a central remote web service that manages the exam status.

Once the heuristics finish with the calculations, the clients use messaging (e.g. secure MQTT included in the initramfs image) with the server to signal the need for images, their negotiated link capacity and start waiting for UDPcast broadcasts. The server forms groups from the clients once enough clients join a group or enough time has passed since last group formation. These two threshold values require tuning in practice so it is too early to specify them yet. The server can also use the link capacity data for group formation. With the grouping, the server can adjust the traffic appropriately, decreasing the network bandwidth requirements significantly for networks with a large number of clients. For example in the 100BASE-T network with three groups for 100 clients, the boot time requirements for the first 20% of the 1000 MB image might decrease to around one minute, a 97% improvement.

C. Size optimization

First, a few words about the initramfs structure of the suggested netbooting Abitti. We suggest a totally custom initramfs that is based on the musl libc[4] and custom boot scripts. Musl is a very lightweight C library used in embedded distributions such as OpenWRT. It can be used for building statically linked busybox[5] based initramfs images. For instance, all the functionality we describe in this paper (except for the encryption part, which requires cryptsetup tools and more libraries) can be fit in $< 260\ kB$. By comparison, a typical distribution initramfs is several orders of magnitude larger. For instance, a desktop Arch Linux initramfs for kernel 4.8.13-1-ARCH is around 21 MB (or larger with more embedded drivers). Note though, although busybox provides a NBD client, it is not useful for new NBD servers with multiple shares.

We can also delay the loading of many kernel drivers to a later stage and only require a comprehensive list of (ethernet) network drivers and the full network stack for mounting and downloading the distribution images. Since we already do automatic ip configuration in the kernel in the code examples, we might compile all the network drivers in the kernel. While this could have been problematic due to hardware resource conflicts when legacy non-PnP ISA hardware was in use, nowadays it should not lock the boot process. We can embed the initramfs into the Linux bzImage to increase boot speed by decreasing the number of required server requests. Since we want to provide multiple versions of the Linux kernels for different types of clients with problematic hardware, we also need to build custom kernels, but the suggested kernel configurations are dependent on the kernel versions and require extensive knowledge. Although kernel configuration is very exciting and educational, we decided to leave it out from this paper.

[4]https://www.musl-libc.org/
[5]https://www.busybox.net/

Algorithm 3 Decision algorithm for obtaining the distribution images.

```
const distro_memory_req
const[] distro_squashfs_part_sizes

memory = calculate_free()

mqtt_send signal_parameters

if (memory < distro_memory_req)
  fail_cond Koneessa ei ole tarpeeksi muistia, yritetäänkö silti?

memory = memory - distro_memory_req

for part in squashfs_parts {
  if part.size < memory {
    mqtt_send "request" part
    dloaded_part = async_receive part --> mount_local dloaded_part
    memory = memory - part.size
  } else {
    mount_via_nbd part
  }
}
overlay_construct squashfs_parts
switch_root
```

The use of ZRAM[10] has become a common practice on the Android platform and low-memory devices and it is relatively fast with the LZ4 compression. We could enable it in the kernel in low-memory clients to free up more memory for applications.

The overlayfs approach of the distribution images also makes it possible to combine parts of the environment with different desktop systems (e.g. separate desktops for low-memory and high-end machines or power users) at boot time which could make the whole Abitti distribution quite lightweight. One possible scenario is to offer the non-graphical base system as one squashfs image, the desktop as another, and finally the graphical end user applications as possibly as a third layer. We can also use different compression modes for different parts of the distribution. For example, LZ4 is relatively fast, but wastes space, XZ is efficient, but slow even on fast hardware[6].

V. Security

Since the network booting arrangement involves opening new services and distributing network traffic in a shared local area network, there are multiple threat scenarios involved. First, the DHCP protocol is vulnerable to hijacking with rogue servers. We argue that such a threat possibility already exists in the Abitti system if the current platform is deployed in a network with a DHCP service so the netboot does not actually decrease the security in such case.

The TFTP protocol is unencrypted and unauthenticated, but we only use it for reading read-only operating system kernels and bootloaders. It could possibly be hijacked, but we rely on logic on the higher level in the distribution for securing against such attacks.

[6] http://facebook.github.io/zstd/

For the UDPcast part, all traffic is also broadcast to all devices in the network. Due to the open nature of each of the used protocol mentioned so far, intrusion detection systems can easily detect the source of bad traffic and inform the administrators. For the messaging protocol, encryption is used and the keys can be provided within the initramfs. Since these protocols are only used between the server and a client and not between the clients, if the other clients are connected to correctly configured switches, the network will not even route the traffic for possible adversaries unless the switch backbone is hijacked. The same can be said about the NBD traffic provided as a fallback option for low memory workstations.

Assume a rogue DHCP/TFTP server manages to hijack the servers and redirect / offer their own Abitti platform images. We could rely on cryptographic authentication or provide an obfuscated secret within the correct distribution images. The distribution images can further be encrypted with a key obfuscated in the initramfs. We could also make use of secure/trusted booting or TPM[12] hardware, which is further discussed in another Abitti proposal. Overall, figuring out the secret during the exam time frame would require multiple steps, manual work and can turn out impractical even for skilled professionals in ICT security. Especially automated attacks can mitigated with variability provided within the operating system images.

For securing the servers against attacks, the suggested protocols do not support remote procedure calls or provide any way to log in or to modify any files (if configured correctly). Of course, a DoS attack is also easily detectable in a local area network.

Setting the netboot as a default may have security implications if the machine is booted while connected to an insecure, possible rogue network. A simple workaround is to boot the machine pass the netboot prompt before connecting to a network or to revert back to boot from local disk when not participating in exams.

VI. Conclusions

So called “diskless” booting from a network has its up- and downsides. In this paper we have discussed some central issues related to the implementation of a network booting environment with Abitti, both on the client and server side. On the client side only minimal configuration or small modifications are required to make many workstations network bootable. In this case the option can eliminate many of the problems with USB media. In the worst case, the workstation may not support network booting, in which case a local USB mass storage media can be used instead. The network booting can be seen as a supplementary way of distributing the platform, but it cannot replace the old way.

On the server side, the network booting does not necessarily require any hardware changes, but new network services must be configured and distributed to schools that plan to pilot the system. The system requires additional configuration for each exam, but the configuration is centralized unlike with portable USB media.

A broadcast protocol for efficiently distributing the operating system image for groups of clients is proposed. The required services also have some security implications which are discussed. Overall, assuming a significant part of the workstations support network booting, it offers an interesting option for simplifying and improving the setup process required for an exam and can also reduce the expenses inherent to working with physical media such as USB sticks.

References

[1] Abitti - Kohti sähköistä ylioppilaskoetta. http://www.abitti.fi/, 2017.

[2] Abitti - Yleiset ohjeet tietokoneen käynnistämiseksi USB-muistilta. http://www.abitti.fi/fi/ohjeet/tietokoneen-kaynnistaminen-usb-muistilta/usb-yleiset-ohjeet/, 2017.

[3] Microsoft system center - unable to pxe boot from a usb ethernet dongle on a dell xps 13. https://social.technet.microsoft.com/Forums/systemcenter/en-US/8d84b766-34be-4be9-b858-a2f811b184ea/unable-to-pxe-boot-from-a-usb-ethernet-dongle-on-a-dell-xps-13, 2017.

[4] Msfn - usb to ethernet adapters and pxe. http://www.msfn.org/board/topic/162015-usb-to-ethernet-adapters-and-pxe/, 2017.
[5] Plugable - supporting pxe over usb deployment scenarios for tablets and ultrabooks. http://plugable.com/2013/10/27/supporting-pxe-over-usb-deployment-scenarios-for-tablets-and-ultrabooks/, 2017.
[6] Top 7 Desktop OSs from Dec 2015 to Dec 2016 . http://gs.statcounter.com/#desktop-os-ww-monthly-201512-201612, 2017.
[7] Wikipedia: Microsoft litigation - Antitrust . https://en.wikipedia.org/wiki/Microsoft_litigation#Anti-trust, 2017.
[8] Steve Alexander and Ralph Droms. Dhcp options and bootp vendor extensions. 1997.
[9] H Peter Anvin and Marty Connor. x86 network booting: Integrating gpxe and pxelinux. In *Linux Symposium*, page 9. Citeseer, 2008.
[10] Seth Jennings. Transparent memory compression in linux, 2013.
[11] M Johnston. Preboot execution environment (pxe) specification, 1999.
[12] TPM Main Specification Level. Version 1.2, revision 116, 2012.
[13] Marin Lopez and PTA Arturo Garcia Ares. The network block device. *Linux Journal*, 2000(73es):40, 2000.
[14] K Sollins. The tftp protocol (revision 2). 1992.
[15] EFI Unified. Unified extensible firmware interface specification, 2012.