ABSTRACT

DESIGN AND IMPLEMENTATION OF THE uLab
REMOTE HARDWARE DESIGN LABORATORY

Timothy Raymond Pearson, M.S.
Department of Electrical Engineering
Northern Illinois University, 2013
Reza Hashemian, Director

This thesis details the design, construction, and operation of a new type of remote hardware design laboratory, based on Free and Open Source Software (FOSS). The need for such a remote laboratory is examined, and the new laboratory is put into historical context within the computing field. Prior and current remote laboratories are compared and contrasted with the new laboratory, and the advantages and disadvantages of each are discussed. Detailed information on the design of each of the laboratory's three main components is provided, along with the general reasoning that produced each component's specific architecture. Integration with test equipment and laboratory hardware is discussed, and scalability and reliability concerns are addressed throughout, with methods given to mitigate potential risks. Finally, a brief discussion on continuance of this open-source laboratory project is given, along with a statement highlighting the final results of this research project.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

MAY 2013

DESIGN AND IMPLEMENTATION OF

THE uLab REMOTE HARDWARE

DESIGN LABORATORY

BY

TIMOTHY RAYMOND PEARSON

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFULLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

Thesis Director:
Reza Hashemian

ACKNOWLEDGEMENTS

Many individuals have contributed resources that have made this thesis possible, and I am grateful for their assistance. Special thanks go to Dr. Reza Hashemian and Northern Illinois University for their significant investment in the server cluster on which this laboratory runs. I also would like to thank those who have put up with my loud, power-hungry server farm at my residence over the years; this work would not have been possible without their understanding and patience. My editor, Janette, who has volunteered long hours ensuring that I obey the rules of the English language, no matter how arcane they may seem, deserves honorable mention as well. Finally, my gratitude goes to the thousands upon thousands of Free and Open Source Software (FOSS) developers and academics who have, over time, built a collection of open-source software and unencumbered documentation solid and comprehensive enough to support complex systems, such as this laboratory. I stand upon the shoulders of giants and can see farther[1]–and accomplish much–due to their prior work and their willingness to share not only their work but also any fundamental knowledge they gained.

1. With apologies to Sir Isaac Newton

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

PREFACE viii

Chapter

1. INTRODUCTION 1

Hardware Design Laboratories 1

Motivation of Research 1

2. BACKGROUND 4

Origins of Remote Computing 4

The Personal Computer Revolution 6

Return to Centralized Services 7

The Rise of Free and Open Source Software (FOSS) 10

Advantages of Linux and FOSS 13

Overview of Field Programmable Gate Arrays 14

Past and Present Remote Laboratories 15

3. uLab DESIGN AND IMPLEMENTATION 19

Design Goals and Laboratory Components 19

Infrastructure 20

Centralized User Management and Authentication 20

Chapter Page

DNS and DHCP 27
Persistent Data Storage and Diskless Nodes 28
Configuration and Session Management Database 30
Public Network Interface 31
Final Design 31
Scalability and Reliability 32
Terminal Services 33
Desktop Environment 36
Software Packages 39
User Data Storage 40
Final Design 41
Scalability and Reliability 42
uLab Remote Hardware Access System 43
Authentication, Authorization, and Encryption 44
Client Design 46
Server Design 47
Protocol Design 49
Test Equipment Interface 53
FPGA Viewer Part 56
FPGA Programmer Part 59
Final Design 62

Chapter Page

Scalability and Reliability 63

Miscellaneous Services 64

4. DISCUSSION AND SUMMARY 66

Future Work 67

5. REFERENCES 74

LIST OF TABLES

Table Page

1. Comparison of Remote Laboratory Features 17

LIST OF FIGURES

Figure Page

1. Major components of a generic uLab system 21
2. Realm controller service interdependency 25
3. Two of the realm management utilities utilized within the uLab system 26
4. Architecture of the terminal services component 37
5. A typical uLab FPGA development workspace 48
6. Hardware access protocol encapsulation for transmission over unsecured networks 51
7. The uLab oscilloscope frontend part, showing a high-voltage transient and the use of a zoom region 54
8. FPGA debug module internal structure 58
9. Single Spartan 6 hardware development pod 61
10. Internal structure of the uLab installation at Northern Illinois University 68
11. uLab installation at Northern Illinois University 70

PREFACE

Many years ago, my eyes were opened to the wide, new world of FPGAs via a public tour of Northern Illinois University's hardware design laboratory, a world I previously thought to be inaccessible to a mere student of integrated circuit design. Laboratories provide students of all disciplines irreplaceable, hands-on interaction with devices and concepts that, until experienced personally, are simply abstractions in a textbook–things not to be touched by mere mortals. Understanding this, I have worked for many years to extend the transformative laboratory experience to students both young and old, utilizing modern technology where possible and inventing the rest where needed. In an era of fast computer simulations and Facebook, hands-on interaction with real hardware has fallen by the wayside, considered an inconvenience by many or, as one student put it, “the domain of technicians and not engineers.” My belief is that engineers must straddle two worlds, both theoretical and practical, in an effort to create meaningful new technologies that will better people's lives for years after their introduction. One cannot properly innovate without a reasonable understanding of a particular technology from many different perspectives; only after a particular technology has been observed, probed, and characterized can our intuition take us to the next levels of form and function.

The remote laboratory described herein is my contribution to making real hardware and design tools available in a convenient manner to all who are interested, regardless of location or current degree of skill. To ensure that readers are able to clearly understand the underlying

technology and factors driving the final architecture, I have organized this thesis in such a way as to present the overall design goals of the laboratory, the available technologies at the time of this writing, and the reasoning behind the design decisions that were made to advance the stated goals. Less emphasis has been placed on the particular form of the laboratory as installed at Northern Illinois University; while important to provide access to the hardware design resources at the institution, and as an initial proof of concept, this particular implementation does not allow the remote laboratory software to reach its full potential, which would require additional time and resources. Nevertheless, this implementation is the first of its kind and will likely expand in the future as interest and support grows.

The reader will note a thread of reliance on open systems throughout this work. This is one of the most important contributions of this laboratory system to the state of the art; prior to this work, the only major remote laboratory systems to utilize open source from the hardware to the interface were severely lacking in features and capability. Unwilling to compromise on either principles of freedom or functionality, I designed and implemented this laboratory system to provide a modern, full-featured, open-source remote hardware design laboratory. The overall system has been pretentiously dubbed the Universal Laboratory, or more conveniently, uLab; this name is partially a pun on Debian (which claims to be the “Universal Operating System”) and partially a reference to my goal of providing universal access to the hardware and software needed to continue the advancement of science and engineering. For the good of mankind, the information and tools required to dream up the next technological breakthrough must never be restricted to a small subset of the population; history shows that true innovation tends to come from unlikely places, and even with the breakneck pace of modern innovation, the smallest

players are sometimes the most influential. We, therefore, may ignore both history and the potential contributions of a single individual only at our own great peril.

It is my hope that this laboratory, someday, and in some small way, may help a student somewhere to realize his or her full potential, and that mankind may benefit from the subsequent work of that individual. All who are involved in scientific research or the support thereof have inherited this common responsibility: to share what we learn and to ensure that future generations may build upon our work, just as we have built upon the work of our predecessors.

INTRODUCTION

Hardware Design Laboratories

A typical hardware design laboratory consists of several workstations and associated hardware in an access-controlled room with rigidly scheduled laboratory dates and times. This laboratory model inherently presents several drawbacks. One of the largest issues with this type of laboratory is the low average utilization ratio; there normally are large portions of each day that the laboratory is nearly or completely idle with no student access permitted. Another drawback is a relatively short window for laboratory sessions, during which the students are more focused on completing particular assignments within the allotted time frame than they are on learning vital concepts via semi-structured, hands-on interaction with design tools and hardware. Additionally, in many institutions, there are insufficient resources available to handle simultaneous usage by all students within a particular laboratory period; this forces multiple students to be assigned to a given workstation and further removes each student from hands-on interaction with the design software and physical hardware.

Motivation of Research

The primary goal of this research is to obviate many of the issues associated with a traditional hardware design laboratory, as well as to expand the ability of students to engage in

hands-on learning with a combination of physical hardware and industry-standard design tools. Secondary goals include cost effectiveness, ease of access, and expandability of the laboratory system to accommodate new technologies and custom laboratory equipment. Free and Open Source Software (FOSS) will play a large role in achieving these goals, and also will help to ensure reliability and scalability of the resultant design. To avoid software compatibility issues, and to ensure that a wide variety of students are able to access the laboratory regardless of the type of computing device they own, a full laboratory workspace will be made available to any network-enabled device that supports the Remote Desktop Protocol (RDP), which is now a de facto standard. This customizable workspace will present a wide array of engineering design and simulation software, as well as the software required for physical hardware access, to the student via a familiar and powerful Windows, Icons, Menus, and Pointer (WIMP)-based desktop interface.

From an institutional perspective, this type of laboratory provides several distinct advantages. By owning a central, homogeneous system that runs on server-grade hardware, maintenance is simplified and downtime is reduced. Adding new hardware for a specific laboratory becomes a simple matter of installing the additional resources in a proper location with network access, and informing the central system of the new hardware's location and purpose. It is no longer necessary to install copies of engineering design software on the general purpose computers in a computer laboratory; students who are enrolled in hardware design courses simply may connect to the laboratory if they wish to utilize the associated design software. This keeps general purpose computing laboratory maintenance to a minimum; in some cases, it even may be possible to replace the general-purpose computers found in such

laboratories with low-power thin clients connected to one or more central laboratory systems. Another significant advantage of the new laboratory is that the students no longer have direct, physical access to expensive and scarce engineering hardware, minimizing the risk of damage. This new laboratory system can, therefore, be used to allow controlled, safe access to hardware that may have been previously unavailable to most students due to damage and/or safety concerns.

BACKGROUND

Origins of Remote Computing

Remote computing traces its origins to a time when general-purpose computers were expensive and relatively rare. These early computers were also large, power-hungry machines, requiring highly specialized building infrastructure to keep them operational [1]. While the earliest forms of these machines utilized non-interactive, batch-oriented, input/output methods, such as punched cards and teletypewriters, need for truly interactive access was growing. The earliest dumb terminals, also known as "Glass TTYs," became available in the late 1960s, in the form of such machines as the Datapoint 3300 [2]. It would be several years before RAM became inexpensive enough to allow the first intelligent terminals to be manufactured. These terminals allowed less bandwidth to be used to show the same information that a dumb terminal could display; this was due to the presence of a local processor that was capable of performing certain limited display operations without direct, low-level involvement of the remote computer [3]. These types of systems can be considered the distant ancestors of modern terminal services, which allow a user to use a Graphical User Interface (GUI), executing entirely on a remote machine through the use of an appropriate thin client device [4].

A second critical development occurred even earlier, in 1957, with the introduction of the concept of time sharing. Time sharing was first introduced as a concept by Bob Bemer in *Automatic Control* [5], and would be successfully implemented on an IBM 704 later that year

[6]. The concept of time sharing exploits the fact that an individual user often presents "bursty" demand to a given machine. This is because most interactive use patterns, such as word processing, leave the machine idle between commands or even keystrokes, yielding a low average utilization of the CPU. By allowing multiple users to utilize the same hardware, the average utilization of the CPU increases with little noticeable effect in the machine's interactivity, thus making maximum use of a scarce resource. A limit is eventually reached when the average utilization of the CPU nears 100%; however, this merely sets an upper boundary on the maximum number of simultaneous users and does not detract from the desirable efficiency increase gained from implementation of the time-sharing concept.

By leveraging both the time-sharing concept and the existence of relatively inexpensive terminals, new uses were found for computers that otherwise would have been impractical or impossible due to the expense of a typical mainframe [7]. Over time, additional advantages of this model were discovered, including high reliability, high resource efficiency, and relative ease of maintenance; most of these advantages were inherited from the mainframes typically used to drive this type of system, and others were inherited from the well-known administrative and control advantages of centralized systems [8]. The concepts underlying both time sharing and terminals remain in use to the present day, although in different forms and implementations. One of the most popular forms of this model, as of this writing, is the Internet itself; many large websites are served by expensive, dedicated server farms, analogous to mainframe computers, while the client-side Web browser takes on the role previously played by intelligent terminals [9] [10]. A lesser known, although important, implementation of this model is generally called terminal services. In this system, a large number of graphical terminals are connected to one or

more central servers, with all application processing being done on the remote server. Given sufficiently high link speed, a user of the terminal services should see little to no difference as compared to usage of a local machine [11].

The Personal Computer Revolution

As faster and less expensive microprocessors were developed, the advantages of time-sharing systems became less pronounced, while the disadvantages of the systems remained profound. Time-sharing systems were considered somewhat unreliable because fluctuations in the instantaneous number of users on such systems could drastically affect performance at any given time [12]. A typical time-sharing system operated under the control of a single organization, and, as such, presented several opportunities for abuse. Central control of a computer system guarantees that the owner of such a system can deny access to any user for any reason at any time. The central system provides a single point of failure and is not usually a guaranteed service; therefore, its continued existence cannot be relied upon for any significant period of time. On a more sinister note, a centralized system also can be used to hold users' data hostage, possibly forcing users to pay an arbitrary fee for continued access to their own data.

Given these significant drawbacks, it is not surprising that the Personal Computer (PC) revolution occurred as the price of minicomputers and workstations dropped low enough to allow an average technically-minded individual to own his or her own computer. This, in turn, encouraged a culture of unprecedented, widespread, low-level machine knowledge and freedom to tinker; in this culture hackers wrote programs for the machines they owned and shared these

programs among members of various dedicated technical groups. One of the most widely known and used products of this unique culture is Linux itself, originally developed by Linus Torvalds on a custom-built 386 PC in order to address limitations he saw in MINIX [13]. Due to, in no small part, individuals such as Torvalds, the years leading up to the introduction of the World Wide Web saw explosive growth in new software and in expansion of tasks to which computers could be applied. For the first time, the WIMP GUI was made available to users at home through products such as the X Windowing System [14] and Microsoft Windows [15]; this, in turn, fueled another wave of innovation as people found new uses for their computers. Interestingly, for many years after widespread adoption of the GUI, existing communication links remained insufficient, primarily due to a lack of both bandwidth and availability, to support the existence of anything resembling the old mainframe/terminal model.

Return to Centralized Services

The first foundations for the return to the mainframe/client model were laid with the introduction of the World Wide Web. Originally conceived as a means for anyone to be able to author and edit shared documents across the world [16], the first implementation of HTTP only allowed read-only access to documents when used outside of the commercially unsuccessful NeXTSTEP operating system [17] [18]. Over time, as communication links improved and broadband became available, people became more reliant on Internet access to a relatively small number of major information sources, thus bringing centralized systems back into mainstream usage for the first time since the introduction of the PC. After the introduction of the cellular

telephone, and subsequent development into the now ubiquitous smartphone, consumers have embraced a return to the centralized architecture of the 1970s for many of the same reasons that drove acceptance of the original time-sharing systems. Many consumers simply want a lightweight, low-power device that, in exchange for a service fee, allows communication with other users and access to various sources of information. Much like the mainframes of yesteryear, these sources of information are large, dedicated server farms that would be impossible to carry around in daily life [19]. Earlier, these consumers would have been forced to purchase a PC to access these information and communication resources; now they are able to obtain the same level of service via their smartphone. As a result of these changes, we fully have entered a "cloud"-oriented, post-PC era, one in which the service model originally introduced with mainframes and terminals is dominant once again.

Post-PC devices, while offering clear advantages in terms of human communication and ready access to several types of information, nevertheless inherently possess several severe drawbacks. Many post-PC devices, such as cellular telephones, rely on software which, under United States copyright law, is licensed property [20], owned and controlled by the original manufacturer, a software development firm, or, in some cases, even the carrier from whom a person has purchased a telephone. Because most telephones, tablets, and other consumer devices contain firmware which prevents the usage of unapproved software [21], under United States law the consumer cannot legally use the device for any purposes other than those expressly permitted by the device's legal owners and any active Copyright Office rules [22]. This has the effect of drastically increasing barriers to entry for anyone who may wish to engage in hands-on learning of application programming. If left unchecked, this unfortunate effect would tend to centralize

the power to develop new applications into the hands of a relatively small number of individuals, ensuring that new generations of students would no longer have the advantage of being freely able to tinker with and experiment upon their own general-purpose computing devices.

Fortunately, the cost of general-purpose computing hardware is still low enough to be affordable by most individuals. This has allowed the open-source community to continue development of new software in much the same manner as during the PC's golden age, albeit at a slower pace than before, and also has served to offset many of the high barriers of entry into a typical smartphone ecosystem. More recent developments include the repurposing of semiconductors that were originally designed for smartphones in order to create a new breed of inexpensive, general-purpose computers; developments like this create a climate in which future generations will be able to, once again, learn about machine internals and gain direct, hands-on experience with programming tools and hardware interfaces. One of the most prominent examples of this type of computer is the Raspberry Pi [23], which, incidentally, plays a significant role in driving down the cost of the new laboratory system.

Another result of the proliferation of post-PC devices is the push for post-WIMP GUIs, primarily due to the fact that post-PC computers lack traditional input devices, such as a keyboard and mouse. The differences between GUI types are substantial, and selection of the appropriate GUI type is critical to ensure that the user can attain maximum productivity for a given class of work. The WIMP GUI is a somewhat rigid paradigm that forces the user to be in control of the machine at all times; that is, the machine generally will not perform any action that was not previously commanded. Additionally, strong emphasis is placed on hierarchical file systems, interface permanence, application consistency, and overall worker productivity [24].

By contrast, the post-WIMP GUI is designed primarily for information retrieval and interpersonal communication, focusing strongly on searches for existing documents, consumption of content, and brief but frequent communication with other individuals; therefore, special emphasis is placed on computer prediction of the user's current activity, and proactive assistance of the user in that task. Much of this technology is essential for smartphones and tablets due to the low input/output bandwidth typically available between machine and user; however, its usefulness and practicality on larger devices is currently a subject of intense debate [25].

The Rise of Free and Open Source Software (FOSS)

Free and open source software plays a vital role in scientific and computing research [26], enabling the creation of novel systems with state-of-the-art technology [27]; therefore, it is useful to understand some of the history and culture that has shaped the open-source movement into the form we rely on today. Early programmable computers were created with heavy influence from professors working at universities; much of the design knowledge and computer software for these machines was, therefore, shared openly with other researchers. A prominent example of this paradigm was IBM's ACP airline reservation system. Written for the IBM/360 mainframe, the source code was available for anyone to modify, improve, and share with other interested parties [28]. Only later, once machine architecture had stabilized and relatively few computer types were commercially available, would closed-source computer software such as UNIX and VAX/VMX become practical.

The free software movement, as we know it today, traces its origins to the MIT Artificial Intelligence Laboratory and the collaborative culture that existed in its hacker community before that community's demise in the early 1980s [29]. Around this time, new computers were being released with proprietary operating systems that could not legally be studied or modified by the end user. As a result, a distinction now needed to be made between the older, traditional, open software and the new, closed, licensed software; “free” and “open-source” became meaningful terms at this point. Due, in part, to the refusal of some people, including Richard Stallman, to relinquish their freedom of software development and modification, the open-source software community was slowly restarted with the primary goal of creating a full-featured, open-source, UNIX-compatible operating system. Interestingly, while Stallman set out to create a complete UNIX-compatible operating system, the GNU project would, instead, become primarily known for its userland tools, including the GNU Compiler Collection (GCC) used by many developers and projects worldwide. To this date, the GNU project has not produced a GNU/HURD kernel that is competitive with any major proprietary or open-source offerings [30]; however, it has continued to produce and maintain core development tools and libraries.

Like the GNU project, Linux was born out of dissatisfaction with closed-source, or otherwise restricted, operating systems. Linus Torvalds was a student when he first wrote a small terminal emulator to overcome limitations in the restricted MINIX system that came with his Intel 386 (i386) computer [13]. Later, this terminal emulator, which only ran on similar i386 machines, grew into a rudimentary POSIX-compliant kernel. In and of itself, this was not particularly noteworthy, but what happened afterward introduced the world to a new development model that was originally ridiculed as doomed to fail. Torvalds released the first

versions of Linux under a license that allowed modification if and only if the changes also were made public, and he completely forbade sale of Linux in any form. After Linux was modified to support swap to disk, its popularity began to increase, and Torvalds had to rethink his original license. He was aware of the work of Stallman, and also aware of the other open-source developers on which he relied; therefore, he decided to release Linux under the GNU General Public License (GPL) [31]. As Linux's popularity grew, more and more people sent in patches to add features and to fix bugs; therefore, despite its initial ridicule, this basic development method has grown into a worldwide success. Torvalds' original fears of IP theft never materialized; by virtue of its large and active development community, anyone who attempts to fork Linux in secret is rapidly left behind as the kernel continues to evolve. In many respects, the open-source concept and development process parallels the best traditions of academia; the original ideas and culture that allowed the development of the first computers have, therefore, come back into widespread use decades later.

Once the foundations of an open-source, POSIX-compliant operating system had been laid down, many other open-source projects found new footing and prospered. Linux became, and still is, the most popular platform for servers [32] [33]; within the past decade it also has become usable in the desktop space. Since the foundation of the GNU project, numerous important open-source projects have flourished. There is insufficient space to do them justice, so I only will spotlight a few critical developments. MIT was responsible for the initial development of the now ubiquitous X Window System, of which some variant is in use on almost every graphics-enabled, POSIX-compliant desktop or server system worldwide. In turn, two major desktop environments and their associated applications were built on top of the X

Window System, namely Gnome and KDE. On the server side, the Apache HTTP server was developed for use on Linux and on other POSIX-compliant systems, and has remained the world's most popular Web server for the past 17 years [34]. In light of this remarkable success, open-source software is now widely considered a serious contender to commercial, closed-source software [35].

Advantages of Linux and FOSS

Advantages of Linux and other free and open-source software include very low barriers to entry, a wide variety of high-quality libraries and graphical toolkits with which to build new applications, standards compliance, and true multi-user support. In general, only an inexpensive PC with an Internet connection is required to install Linux and to gain access to a wide array of open-source development tools. Linux also supports many different desktop environments; a Linux machine can, therefore, be customized to optimize specific tasks, whereas commercial operating systems generally provide a single, one-size-fits-all desktop environment. Unlike many commercial operating systems, Linux is fundamentally a multi-user system, supporting multiple independent users on a single machine and providing a security framework designed to keep user accounts separate and secure. In general, unless a user has been granted administrative privileges or an exploit has been found, it is impossible for a single user to bring down or fundamentally corrupt a Linux system; this is an attractive characteristic when considering a system for deployment in a collegiate environment.

Overview of Field Programmable Gate Arrays

The initial version of this remote laboratory primarily targets remote development with Field Programmable Gate Arrays (FPGAs); therefore, a brief look at the history of the FPGA follows. The FPGA was originally invented by Ross Freeman [36], a founding member of Xilinx [37], to allow customers to define custom, on-chip logic without requiring design and manufacture of an expensive Application Specific Integrated Circuit (ASIC). At the time, transistors were an expensive, on-chip resource, and major semiconductor corporations, such as Zilog, were unwilling to commit resources to a project that could end up as a commercial failure. Freeman left Zilog, along with Xilinx co-founder Bernard Vonderschmitt, to pursue his idea through their start-up, Xilinx, funded primarily by venture capital [38]. Xilinx shipped their first FPGA, the XC2064, in 1985, leveraging common workstation platforms to support the basic design tools needed for programming this new type of device [39] [40] [41].

The fact that the entire development toolchain for all major FPGA manufacturers has been completely closed-source from the beginning has led to a number of problems and inefficiencies in a typical FPGA development cycle; these start with random failures inside the synthesis tools and end with the requirement that an expensive, proprietary programmer be utilized to interface with the FPGA device over the Joint Test Action Group (JTAG) interface. Furthermore, the closed-source tools are typically compiled for Intel 386-compatible platforms only, severely restricting the use of new, power-efficient development platforms and also preventing the emergence of self-modifying hardware designs; although such designs have been

studied in theory, none have been reduced to practice due to this limitation of the toolchain.

Modern FPGAs can be viewed as a blank integrated circuit containing many different specialized components; these components can be dynamically wired together utilizing a configurable interconnect fabric. For example, a typical Xilinx device includes, at minimum, clock generation modules, dedicated clock routing buses, small RAM blocks, and I/O buffers. Newer devices include more complex modules, such as SERializer/DESerializer (SERDES) blocks, to enable high-speed interchip communication. Fundamentally, though, the basic operation of an FPGA has not changed much since its invention. The FPGA still contains large quantities of configurable discrete logic within its fabric, grouped together into slices that are then used by the manufacturer-provided toolchain to implement designs described in a standard Hardware Definition Language (HDL). The FPGA remains a popular alternative to custom ASIC designs because, in most cases, the slight benefits in unit cost and performance of a custom ASIC do not justify the high cost and initial risk associated with such a design. These devices also are ideal for an educational setting, allowing a student to create a custom integrated circuit design and actually test that design, instantly, on real hardware with negligible cost to either the student or the educational institution. FPGAs, therefore, enable hands-on learning in an area that was previously inaccessible to students, and also largely supplant the prior methods of wiring new designs by hand from discrete logic gates.

Past and Present Remote Laboratories

Several attempts have been made over the years to create a remote laboratory

environment that can supplant the existing physical laboratories present in most universities. Many remote laboratories focus on simulation only, or on point-and-click guidance through a specified, simple experiment. Additionally, many of these laboratories do not offer terminal services, and those that do are heavily reliant on Microsoft Windows and other closed-source software for their operation. Due, in part, to the various shortcomings in each remote laboratory system, none has seen widespread adoption in education. It is the author's belief that in order for any new technology to be widely adopted, and especially if it is to supplant an existing technology, the new technology must provide all the features originally present in the existing technology, as well as contribute at least one new and useful feature that justifies the migration. Another method to force widespread adoption historically has been deprecation and removal of the older technology; this is problematic because it also has made users of such products justifiably reluctant to invest in current systems produced by the same source, especially if the current system is considered largely inferior to the older system.

Table 1 provides an overview and comparison of the various types of remote laboratories in existence today, including the new remote laboratory system detailed in this document (hereinafter referred to as the Universal Remote Laboratory or uLab). As can be seen, the new remote laboratory is the only full-featured system to make extensive use of open-source software; this allows the laboratory to be customized to the particular needs of a given institution and also ensures that there is some degree of permanence built into the overall system. Additionally, the new laboratory is the only laboratory that provides an unscripted, reconfigurable workspace to the user; instead of simply altering one or two input variables and observing a response, the user is given complete control over a desktop-based development

session, a reconfigurable hardware device, and any test equipment attached to that device, thereby closely approximating a typical physical laboratory environment. This unscripted environment inherently includes the ability to use the equipment in unusual and novel ways, or even to fail to produce a working experiment; these real-world traits are not present in most guided online laboratories.

Table 1: Comparison of Remote Laboratory Features

	uLab	RemoteFPGA [42]	SolarLab [43]	MIT iLab [44] [45]	TCAD [46]	VISIR [47]	TINI-based [48]
User access control	X	X	X	X	X	X	
Pluggable backend server access control	X			X		X	
Encrypted, authenticated client-server communication	X						
Encrypted, authenticated inter-server communication	X						
Single sign-on (SSO)	X						
Terminal services	X	X			X		
Integration with existing course management systems							
Access from new forms of computing devices	X		X	X	X	X	X
Web-based interaction			X	X	X	X	X
Rich GUI widget I/O (not camera based)	X	X	X	X	X		X
Modular design	X			X			

Legend:
X Supported
/ Technically possible with minor changes but not yet implemented
An empty box indicates that the feature is unsupported without major changes to software, hardware, or both.
Please note that only those features mentioned in the referenced sources are included in this table.

(continued on following page)

Table 1: Comparison of Remote Laboratory Features

(continued from previous page)

	uLab	RemoteFPGA [42]	SolarLab [43]	MIT iLab [44] [45]	TCAD [46]	VISIR [47]	TINI-based [48]
Reconfigurable hardware connections					X	X	
Integral surveys and/or learning assessment	/		X	X			
HCI method	WIMP	WIMP	WIMP	WIMP	WIMP	WIMP	WIMP
Primary client/server programming language(s)	C C++	Visual Basic	LabView PHP	Java LabView	LabView	LabView	Java MATLAB
Remote development tools	X	X		X			
Remote simulation tools	X	X		X	X		X
Access to physical hardware	X	X	X	X	X	X	X
Exclusive, full access to physical hardware	X	X	X	X	X	X	X
Inexpensive, low-power network to hardware interface	X						X
Unscripted experimentation environment	X	X		X		X	
Laboratory software available as open-source	X	X	X	X		X	X
Storage for user design files	X	X					
Integral capture and storage of experimental results	X						

Legend:
X Supported
/ Technically possible with minor changes but not yet implemented
An empty box indicates that the feature is unsupported without major changes to software, hardware, or both.
Please note that only those features mentioned in the referenced sources are included in this table.

uLab DESIGN AND IMPLEMENTATION

Design Goals and Laboratory Components

The new remote laboratory system was designed, from the ground up, with scalability, reliability, and security in mind. Single points of failure were determined, and, where possible, eliminated through the use of redundant systems. Where this elimination was not feasible, largely due to a lack of underlying network protocol support and global concurrency issues, highly reliable systems were utilized. By transparently distributing session resources across multiple systems, both reliability and scalability were enhanced; this is because the central servers were no longer directly involved with execution of design tools or access to physical hardware. If a non-central server were to fail or crash, a hot spare can be placed into service immediately, thus minimizing disruption to users of the system. Multiple non-central servers, hereinafter referred to as nodes, also could be installed into one remote laboratory system; this would provide nearly linear scalability of the maximum number of concurrent sessions as long as bandwidth limits were not approached.

The new laboratory system is comprised of three main parts: fundamental infrastructure, terminal services, and laboratory workspaces. Without proper infrastructure, the laboratory would degenerate into a loose collection of relatively isolated software with a corresponding increase in difficulty of use and management. Therefore, this discussion will describe infrastructure design goals and their practical implementation, and then progress, in a similar

manner, to the dependent services that comprise the remotely accessible portion of the laboratory. For reference, a simplified, generic laboratory design that uses all three components is shown in Figure 1.

Infrastructure

Because this system will be exposed to the Internet and also may be exposed to unsecured public networks for inter-server communication, user management and encryption play a large role in the design of the laboratory. Design goals include single sign-on (SSO) capability; centralized, point-and-click user management; varying levels of access to laboratory workspaces and software; authentication of users and servers; encryption of all communications; and the ability to safely place equipment interface servers wherever physically convenient. While several open-source solutions exist that can handle centralized user management, authentication, or authorization, including NIS [49], Samba AD [50], Kerberos [51] [52], and OpenLDAP [53], none of the existing solutions met all of the design criteria on their own. Therefore, a custom solution was formed through the assembly of NTP [54], Heimdal Kerberos, OpenLDAP, GSSAPI [55], and SASL [56], as well as the creation of new GUI and Command Line Interface (CLI) tools for realm and user management.

Centralized User Management and Authentication

Kerberos, a key component of the custom solution, is primarily a secure authentication

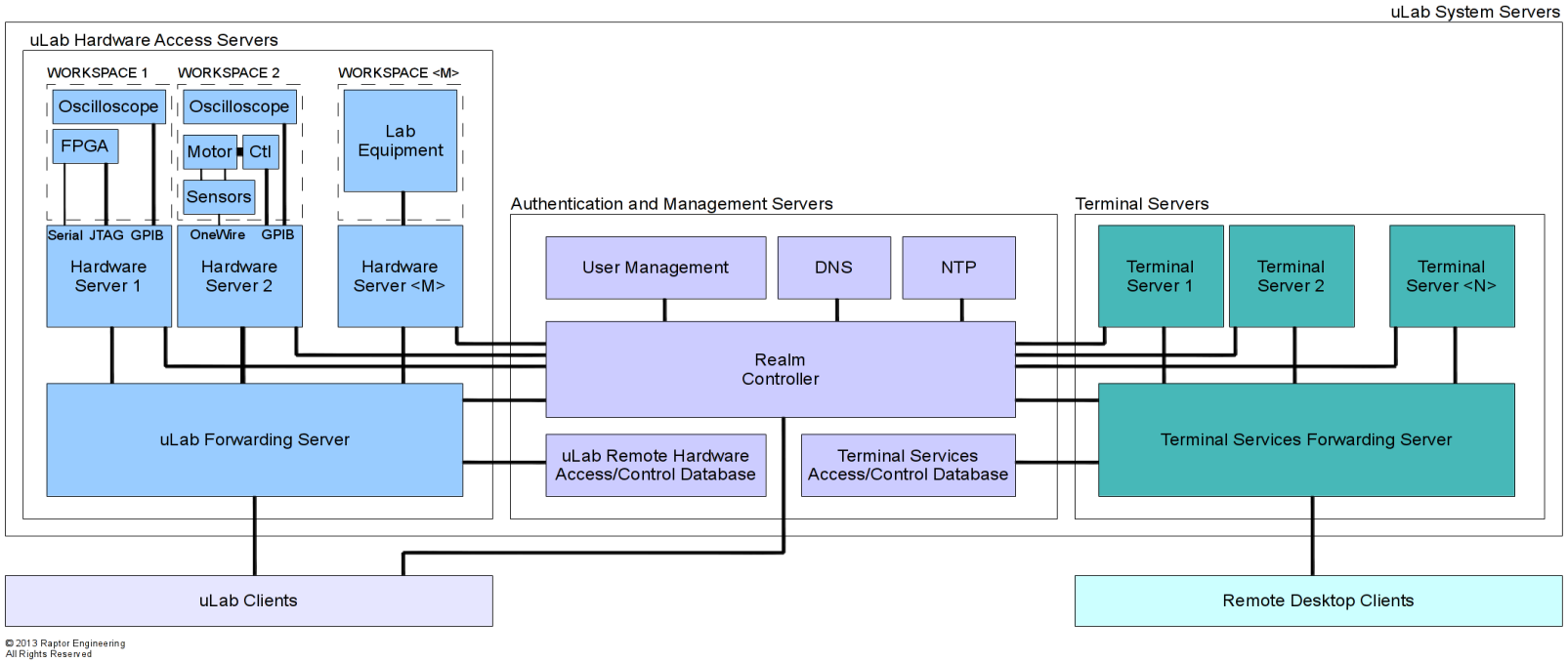

Figure 1: Major components of a generic uLab system

protocol, working through the use of "tickets" to cryptographically grant access to remote services. Each ticket is granted to a user from a central server, on request, via a passwordless exchange; at minimum, this ticket guarantees that the ticket holder is who he or she claims to be on the network. Generally, this initial ticket is given in the form of a "ticket-granting ticket," which allows the user to request access to specific services without reentry of the logon password. Once the user has received a service access ticket, all further communication and authentication for that service is performed solely between the service provider and the client. Tickets are mandated to expire after a certain duration to prevent a wide range of attacks; this expiration interval is configurable on the central Kerberos server(s). Expired tickets may be renewed through communication with the central Kerberos server(s), up to a maximum number of times specified by the configuration on the central Kerberos server(s). After all renewals have been used, the user must re-authenticate to Kerberos and receive a new ticket-granting ticket to continue using services.

Holders of Kerberos tickets are able to establish encrypted communication with remote machines after initial authentication. This allows the entire exchange, from authentication to connection termination, to take place within a secure channel, effectively guaranteeing that eavesdropping attempts or data injection attacks are not possible if an attacker only has access to the communications network. Additionally, since authentication is an integral part of establishing the encrypted connection, all servers and clients "know" who they are communicating with at all times. Kerberos groups all machines with access to a specific Kerberos server or set of servers into Kerberos realms, with the name of each realm consisting of an uppercase version of the machines' Domain Name System (DNS) suffix. For example, if a

Kerberos server's full DNS name was server1.my.domain.edu, the Kerberos realm would be MY.DOMAIN.EDU.

Stand-alone Kerberos has a number of drawbacks that mandate its use with other technologies like the Network Time Protocol (NTP) and Lightweight Directory Access Protocol (LDAP). The cryptographic exchange that occurs during authentication is extremely sensitive to clock skew, and will fail if the clocks on the two machines involved are offset from each other by more than a few minutes. This mandates the use of a technology such as the Network Time Protocol; when such an NTP server is set up on the primary Kerberos server, all configured clients automatically can keep their clocks in nearly perfect (<1ms jitter over most links) synchronization with the Kerberos system. In turn, the NTP server is configured to keep its clock in synchronization with a global clock reference, such as the NTP pool, or a lower-stratum (higher precision), atomic-clock-derived reference, such as a local GPS receiver.

Another significant drawback of most stand-alone Kerberos installations is the difficulty of managing users. Kerberos was designed to store information only directly related to authentication; therefore, it does not provide any authorization or user-management related information, such as POSIX group membership, POSIX user ID, home directory, or even the user's full name. Without this additional information, Kerebros authentication is nearly useless because the systems and servers using such authentication will not generally know what permissions to grant each authenticated user. Therefore, a separate LDAP server is required to store this additional information and make it available to Kerberos-enabled systems on request.

A typical LDAP server, such as OpenLDAP, is essentially a network-enabled wrapper around a very small, lightweight database. Its sole purpose is to store and transmit data, on

request, over the network. On its own, an LDAP server provides basic access methods, such as password-based login over a (possibly encrypted) network link, but the LDAP server cannot natively provide the robust user authentication, ticketing, or encrypted links characteristic of the Kerberos system. Furthermore, a stand-alone LDAP server maintains its own user database, which will tend to drift out of of sync with more advanced user management methods, such as Kerberos or NIS. Clearly, some combination of the two services is ideal, and many corporations and individuals have combined the two with varying degrees of success.

Kerberos can be configured to store its authentication information on an LDAP server; however, this still does not solve the problem of LDAP maintaining a separate user database. Therefore, a third piece of software is required, namely the Simple Authentication and Security Layer (SASL). OpenLDAP can be configured to use SASL to authenticate users; SASL, in turn, uses Kerberos to authenticate users passed to it by OpenLDAP, as illustrated in Figure 2. As might be obvious at this point, there is a circular dependency loop present for non-local access when this system is operating normally. The presence of this loop, when combined with OpenLDAP and Kerberos' arcane and poorly documented configuration files, makes the creation of a working server utilizing all three technologies very difficult. Thus, for the uLab system, new open-source tools were created to automatically handle this tedious and error-prone task. The new tools also allow easy, point-and-click access to post-setup maintenance functions, such as LDAP root certificate updates and administrative password changes; two of the new tools are shown in Figure 3.

On the local machine hosting Kerberos and LDAP, the dependency loop is purposefully broken at one point; Kerberos communicates to OpenLDAP through a local domain socket

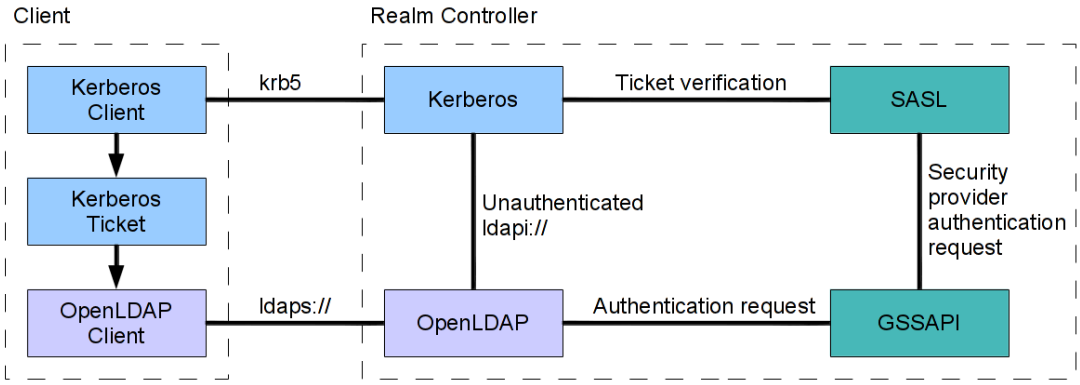

Figure 2: Realm controller service interdependency

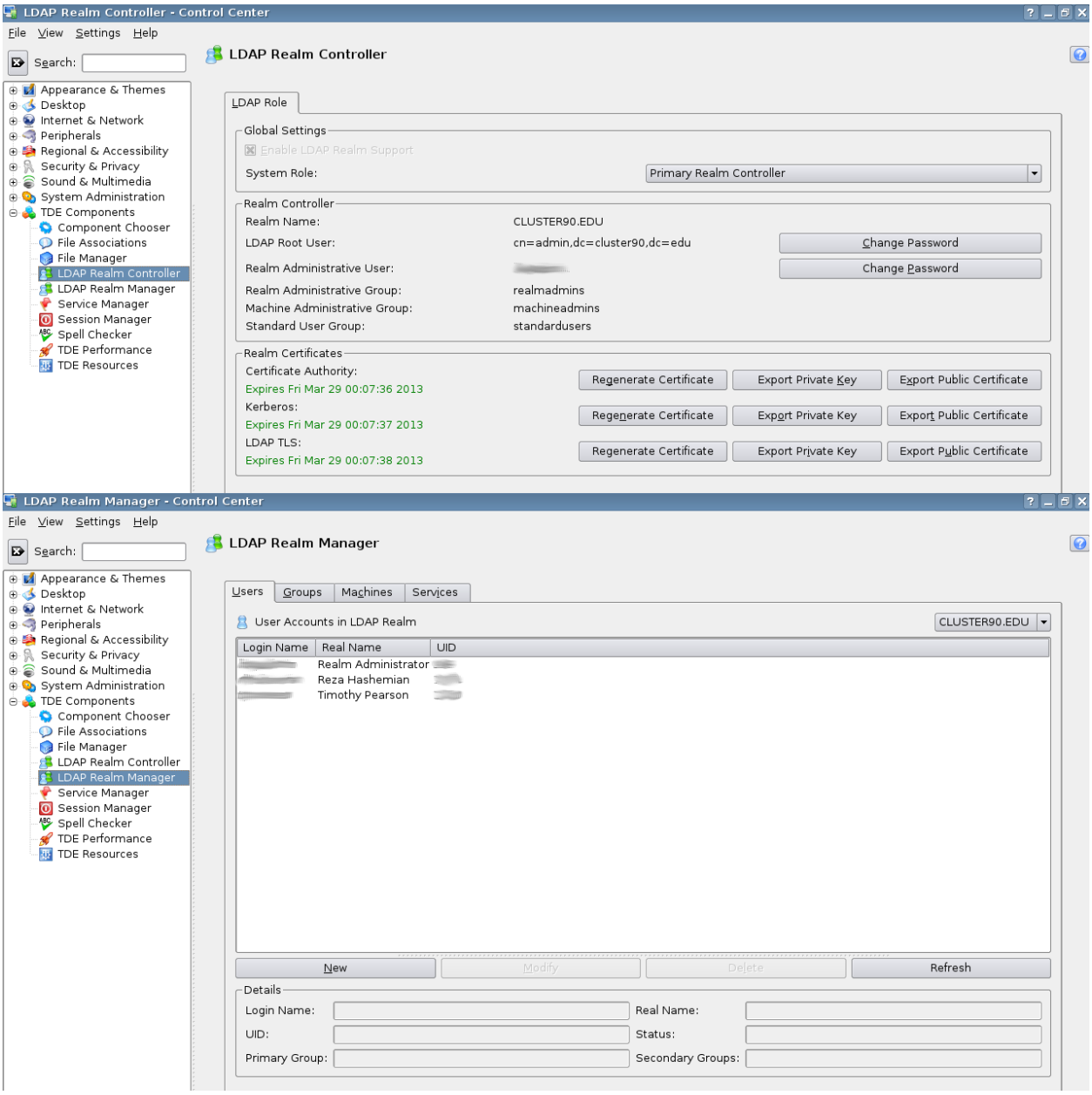

Figure 3: Two of the realm management utilities utilized within the uLab system

(ldapi://), bypassing OpenLDAP's authentication protocols entirely. Therefore, it is important to both keep the realm controllers in a secure physical location and to have no other services of any type running on the realm controllers. The realm controllers are the largest single point of failure and security concern of the entire system; if even one realm controller is breached, an attacker could gain full access to every machine in the realm. Similarly, if only one realm controller is present, its failure will cause all machines in the realm to reject all non-cached/non-local login and service access requests, effectively halting normal operation of the entire realm. LDAP and Kerberos both contain mechanisms to create backup secondary realm controllers that allow the realm to continue normal operations if the primary realm controller fails; however, alterations of the user directory generally are not possible unless a primary realm controller is present and active.

DNS and DHCP

A critically important and often overlooked Kerberos requirement is proper forward and reverse name resolution of all servers and client machines within the realm. This is accomplished in the new remote laboratory through the use of the Berkley Internet Name Daemon (BIND) DNS server [57], containing correct entries in both the forward and reverse DNS files for every server within the cluster. In a larger network with hundreds of workstations and possibly even Bring Your Own Device (BYOD) support, BIND can interface with the ISC Dynamic Host Configuration Protocol (DHCP) server [58] to automatically create and remove the appropriate DNS entries on DHCP lease allocation/expiry. For the service to work, the DNS name of the Kerberos realm must be resolvable from all member machines within the realm. If

the Kerberos realm's DNS name is not resolvable, Kerberos will fail to work because it will be unable to locate the appropriate servers from which to request tickets. Similarly, if a client is not properly entered into the DNS system, the server will refuse to generate tickets for that unknown client.

Persistent Data Storage and Diskless Nodes

Because the remote laboratory provides users with access to a desktop and various design and simulation tools, some form of user data storage is needed. To simplify uLab maintenance and to limit potential points of failure, a single, large, redundant disk array is provided for this purpose. This disk array also contains, on a separate partition, the operating systems and data files of all nodes within the cluster; each node is booted via the Preboot Execution Environment (PXE) [59] and the Network File System (NFS) [60], and contains no local permanent storage devices. This architecture allows a failed node to be swapped out simply by replacing the entire server and changing its persistent DHCP lease to reflect the new server's MAC address. No data copying or setup is involved; this allows a server to be swapped out (or a spare machine brought online) within a matter of minutes, limiting downtime and any related disruption of services.

The central disk array is shared within the cluster over NFS v3; this is a basic file sharing protocol that does not allow encryption or authentication. It is assumed that the internal network within the remote laboratory cluster will be physically protected against malicious devices; if for any reason this is not possible, then diskless clients should not be used. If it becomes desirable to share the disk array outside of the remote laboratory cluster, NFS v4 should be used because

NFS v4 supports Kerberos-based authentication and encryption. Currently, the uLab system uses NFS v3 and permits no direct external access to the disk array.

Each diskless node is not capable of immediately starting the Linux kernel from its internal BIOS. While projects such as CoreBoot [61] exist that would make this feasible in some situations, the extreme difficulty of replacing the provided BIOS with a custom bootloader makes this impractical for all but the largest server farms. Instead, the Linux kernel must be loaded from an intermediate bootloader, which is, in turn, loaded from the network card's network boot system, and, thus, from the provided BIOS itself. The intermediate bootloader used by uLab is PXE; this loader requires both a DHCP and a Trivial File Transfer Protocol (TFTP) server [62] to function. On boot, the network card's firmware uses DHCP to acquire an IP address, and also to locate the TFTP server on which the network bootstrap program is located. On acquisition, the firmware downloads the network bootstrap program (in this case, the PXE bootloader) into RAM and executes it. PXE, in turn, queries the TFTP server for valid configuration files; these files are searched, in order, from most specific (MAC address) to least specific (IP address, then a default file). Each configuration file contains, at minimum, the filename of the Linux kernel and associated Initial Ram Disk (initrd) image to download and boot. Normally, the kernel command line also is specified, which allows the root device to be set to the NFS server and path containing the system files for the particular node. Upon successful download, PXE boots the kernel; the kernel, in turn, locates the system files on the specified NFS server and mounts that directory as the system root, thus completing the initial stages of the boot process. From this point, boot proceeds normally, as it would on any other Linux system, with sequential service startup leading to boot completion.

As with any centralized solution, there is a bottleneck inherent in the communication links connecting the servers to the disk array. To ensure that sufficient bandwidth is available at all times, a dedicated 10Gbps Infiniband [63] system was installed solely to transfer data between the terminal service nodes and the server containing the disk array. Core system files, Kerberos packets, and RDP data are transferred over a standard 1Gbps Ethernet (IEEE 802.3) system [64] that links all of the disparate nodes together. Security is ensured by denying users direct access to the networking hardware, forcing all requests for data to go through the kernel's built-in security protocols. As mentioned earlier, if this is not possible, then diskless clients should not be used; the infrastructure required to support them would present a security risk when used outside of a dedicated, physically secured environment.

Configuration and Session Management Database

Many of uLab's remote laboratory services use MySQL [65] to determine the privilege level of authenticated users, for statistical tracking, and for service configuration; therefore, the central server also provides a MySQL database server instance to the internal network. This instance is strictly off-limits to users of the system, but is accessible from the services running on the various remote laboratory nodes. In general, the databases are intended to be directly modified only by a qualified administrator during installation of a new service and by the various laboratory services themselves. Similarly, the statistics gathered are designed to be used by an appropriate data mining tool, not obtained directly in human-readable format. To maintain modularity and security, each of the two remote laboratory components stores its information in

a separate, self-contained database within the MySQL server, and accesses the appropriate database through its dedicated machine user account.

Public Network Interface

A final interface is needed between the remote laboratory cluster and a public network or Internet. This interface should consist of a router with a firewall configured to forward only a few select, needed ports onto the public network, thus keeping the internal network separate and secure. Ideally, this firewall would be a stand-alone system that does not run the same base operating system as any of the machines in the remote laboratory cluster; this prevents a single exploit from being used to take down the entire system, as would be possible in a homogeneous environment. Furthermore, this router should be able to track bandwidth used, hacking attempts, and, as the last single point of failure, should be constructed from a high-reliability system. pfSense [66], a router built on the Berkley Software Distribution (BSD) and with all of its functionality accessible through a web-based point-and-click GUI, meets these needs in the uLab system. pfSense also provides several optional features, such as a built-in Intrusion Detection System (IDS) and the ability to block abusive clients.

Final Design

The final infrastructure design for the uLab system uses three separate, high-availability servers; a Gigabit Ethernet (GbE) switch; and an Infiniband.10Gbps switch. One server is

dedicated to pfSense (router001.cluster90.edu); this machine accepts a high-speed public Internet connection and routes it to a dedicated network port on the master server (master001.cluster90.edu). The master server runs Debian Wheezy; it also contains a large Extended File System v4 (ext4) disk array, two GbE network ports, and a 10Gbps Infiniband interface card. This machine hosts DNS, DHCP, TFTP, NFS, MySQL, and Apache HTTP services for the entire cluster, and is a member of the internal CLUSTER90.EDU Kerberos realm. In this particular instance, the master server also hosts a copy of the Debian package archives over HTTP for access by machines within the remote laboratory cluster; this makes software management both faster and more reliable. The final server (ldap001.cluster90.edu) is a dedicated Kerberos realm controller, hosting LDAP, Kerberos, and SASL services for use by the entire CLUSTER90.EDU Kerberos realm.

Scalability and Reliability

LDAP, Kerberos, and MySQL are known to reliably scale to thousands of clients with ease, require little bandwidth, and support inter-machine replication; therefore, I will treat access to those services as essentially unlimited. However, both the centralized disk array and router present challenges to scalability. Some of the scalability and reliability concerns of the disk array can be mitigated through the use of a clustering file system, such as Oracle Cluster File System v2 (OCFS2) [67] or the Z File System (ZFS) [68]; however, this can lead to a performance loss, depending primarily on the speed of the interconnect between the replicated file servers. If diskless clients are in use, they should be allocated in racks with a dedicated disk

array for their system files placed within each rack, with only the global user data store utilizing the clustered file system. In a similar fashion, limitations of a single central router can be worked around through the use of several routers, with clients being randomly routed to a specific router on initial connection through a round-robin DNS scheme. Therefore, limits on scalability should not be apparent within the fundamental architecture until the aggregate network bandwidth between servers is nearly saturated.

Terminal Services

Unlike most competing remote laboratory solutions, the uLab system does not use a Web-based interface to any of its services, opting instead to provide a traditional, feature-rich, WIMP-based GUI in order to provide an experience that is as close to real-world work as possible. As such, it is important to provide access to a full desktop environment in which the design tools and remote laboratory GUI can be utilized effectively and efficiently. For this reason, terminal services are provided to the students over Microsoft's industry-standard Remote Desktop Protocol (RDP) [69]. Normal terminal services over RDP, such as those provided by Microsoft or the open-source FreeRDP project [70], are not easily scalable due to the fact that each client must connect to a given server. Providing additional terminal servers, while alleviating overcrowding on existing servers, brings additional challenges in the form of session consistency, server management, and active session resumption after disconnection.

Design goals for the terminal services component of uLab include ease of use, single sign-on, persistent sessions, persistent user data, efficient use of bandwidth, access to shared data

files (such as course information and digital drop boxes), and usage of a standard protocol that most network-enabled computing devices can utilize. Additionally, the terminal services should provide a wide range of appropriate hardware design and simulation tools to allow users to select the best tool for a task, instead of being forced to use a limited or incorrect tool simply because it was the only workable tool available on the remote system. The hardware used for the terminal services nodes should be powerful, with multiple CPUs and a large amount of RAM installed to support multiple users running complex simulation or synthesis tasks simultaneously.

The RDP protocol fundamentally provides the ability for input devices, such as a mouse and keyboard, to transmit input actions over a network to a terminal server for processing. Similarly, it also provides the ability for the terminal server to send back the current remote screen as a series of compressed images, transmitted only when an item or items change on the remote screen. At one time, various portions of the basic RDP protocol were covered under various patents; however, most (if not all) of the relevant patents have expired in the United States. If there is any concern of patent violation in the intended application (i.e., commercial use instead of computing research in areas where any RDP-related patents still are valid), then alternative, but less widely supported, equivalent protocols, such as the X Display Manager Control Protocol (XDMCP) [71] or Virtual Network Computing (VNC) [72], with a separate PulseAudio [73] service, should be used instead of RDP. Encryption is supported by RDP, but is relatively weak by modern standards; there also is no way to increase the nominal encryption level of the terminal services when used with most existing remote desktop clients. When using RDP or a similar protocol, all computations, input processing, and display actions are performed on the remote server, reducing the client to the role of a dumb terminal. This characteristic

allows almost any computing device to connect to the system and provide a basic level of functionality; only a network-enabled computing device and its associated keyboard, mouse, and display are required to interact with the remote terminal server.

To provide redundancy and scalability to uLab's terminal services, the xrdp server from the FreeRDP project was modified to support the concept of a central forwarding server. This forwarding server is responsible solely for initial authentication and selection of an appropriate RDP backend server; each backend server is then responsible for hosting individual terminal sessions. This architecture alleviates the scalability and reliability concerns typically associated with a single terminal server by allowing multiple RDP backend servers to be utilized transparently, even though the system only presents a single terminal services address to its users. Cross-session consistency is ensured through the use of the central disk array, which stores all user data including each users' desktop configuration files. Additionally, each login is recorded in a dedicated database with all of the pertinent information required to reestablish a connection to the backend server, if needed. This information includes the username, backend server, X11 display number, and Process Identifier (PID) of the window manager in control of that session. Upon termination of the window manager, the associated session information is removed from the database, and all daemons and processes related to that session are terminated.

Communication between the RDP forwarding server and the RDP backend servers is accomplished over three separate channels. Control of the RDP backend servers is handled via passwordless secure shell (SSH) commands; the commands originate on the forwarding server in response to login and termination requests, and are sent to the appropriate backend server during session startup and teardown. RDP video and input device actions are handled via an xrdp

session stream, with the forwarding server acting as a simple router, ensuring that each stream is routed from the appropriate RDP backend server to its attached remote client. Audio is handled in a third PulseAudio stream. Each terminal session executes an independent PulseAudio server; this server then captures all sounds generated within the active session, and transmits the resultant audio stream to the RDP forwarding daemon for subsequent transmission to the client. While portions of this architecture were present as legacy code within the xrdp project, the original intent of that code appears to be separation of a single forwarder and single backend server. A significant amount of work on the xrdp codebase was required to make the aforementioned design work reliably and to incorporate much-needed features, such as remotely commanded termination of active sessions and tracking of active-connected versus active-disconnected sessions. The final design of this component is shown in Figure 4.

Desktop Environment

Linux provides a wide variety of desktop environments to choose from; therefore, selection of an appropriate environment that will both enable complex engineering tasks and function well over RDP is critical. The two most widely available desktops, as of this writing, are KDE [74] v4.x and Gnome [75] v3.x; however, neither of these is suitable for use over RDP for a number of reasons. Both rely heavily on OpenGL and raw CPU power for “eye candy” (graphical effects that primarily serve to entertain the user); because the terminal servers do not contain graphics cards to enable accelerated OpenGL, severe performance penalties would be incurred by its continual use. Although KDE contains an OpenGL-free compatibility mode, it is

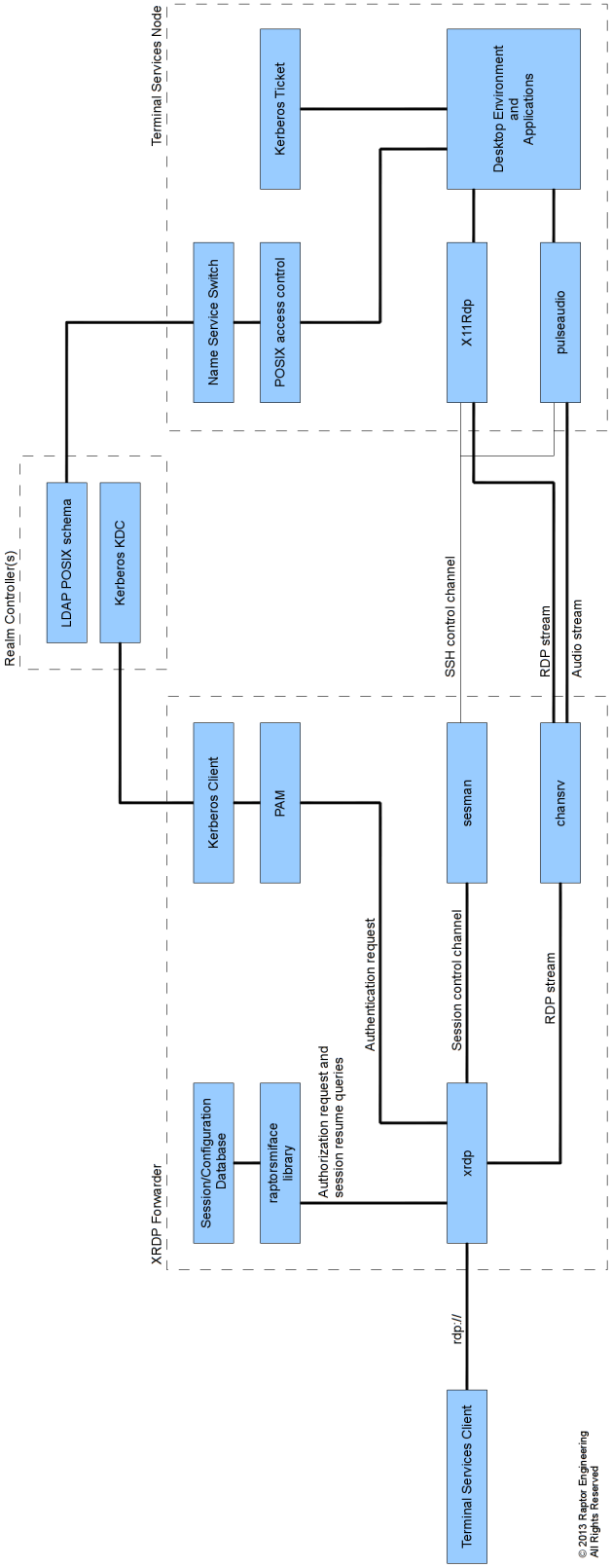

Figure 4: Architecture of the terminal services component

built on top of the Qt v4.x toolkit, which does not perform well over remote desktop links. Furthermore, both desktops are built around a concept generally referred to as the “semantic desktop”; this concept primarily is designed for personal information management and retrieval. The search and indexing tools on which the semantic desktop is built also require a fair amount of CPU power and memory to function properly, and the desktop environment will not function correctly without them. Supporting the maximum number of users on a given terminal services node requires that all sources of replicated bloat, that is, any unneeded CPU and memory usage caused by a single user session, must be carefully controlled. In the author's opinion, neither KDE nor Gnome is a suitable candidate for use with the new remote laboratory system, since both inherently use non-trivial amounts of CPU and memory per session for functions that do not enhance the overall laboratory experience.

Fortunately, there are several less popular, non-semantic, WIMP desktops from which to choose, including LXDE [76], XFCE [77], Cinnamon [78], and TDE [79]. All are reasonable choices for inclusion in a remote laboratory; however, TDE was chosen due to both its unique feature set and the author's familiarity with the TDE desktop environment. In the author's opinion, LXDE and XFCE are too light on features to be chosen if another reasonable alternative is present; also, they, together with Cinnamon, suffer from a somewhat complex, confusing programming style and set of Application Programming Interfaces (APIs). Because all three desktops are based on the Gimp Toolkit (GTK) [80], programs written in the competing Qt [81] toolkit do not integrate well with any of these desktop environments, and vice versa. Therefore, providing a consistent user experience practically mandates the use of the native programming toolkit of the desktop environment in use; where workarounds do exist for other desktops, they

generally consist of complex and largely unsupported pieces of software. By contrast, TDE is built on the older Qt v3.x toolkit, which, despite its age, not only works efficiently over RDP and XDMCP links but also presents a reasonably powerful programming style and set of APIs to the developer, thus making it the best choice for the uLab system.

Software Packages

The desktop environment in use is only one consideration in providing a full-featured laboratory workspace for the student. Careful attention must be given to the applications that are made available to the user; for a computer engineering laboratory to be successful, a wide variety of design tools must be installed. Broad categories include: software development tools, hardware development tools, hardware simulation tools, office and graphics tools, and remote-hardware access tools. In the uLab system, the first category is populated with FOSS programs, such as Kdevelop [82], Eclipse [83], gcc [84], and similar utilities, with no need to resort to proprietary or closed-source tools of any type. Hardware development tools are a different matter, as discussed earlier; in particular, FPGA design tools, such as Xilinx's ISE, are only available as closed-source bundles from the FPGA's manufacturer. However, a few notable exceptions to this general rule exist, such as gEDA [85] and LibreCAD [86]. The new remote laboratory also includes closed-source freeware, such as LASI, an integrated circuit design package. Hardware simulation tools include KPicoSim, a FOSS Picoblaze simulator, and LTSpice, an excellent closed-source SPICE-based circuit simulator. Office and graphical tools are provided through the comprehensive FOSS LibreOffice suite, with advanced graphics

handled through the inclusion of GIMP [87], another excellent FOSS program. Finally, a means of accessing the hardware of the remote laboratory must be provided; in the uLab system, this is handled via the inclusion of the uLab remote client, as detailed in a later section.

User Data Storage

All of the tools listed above require read-write access to a significant amount of permanent storage, as does the desktop environment itself. Following UNIX tradition, each user is given a home directory, which resides on the central disk array. Each user has full read-write access to this directory, and is expected to properly organize his or her files within this space. It may be advantageous to provide each user with read-only or write-once access to certain other paths, with the former being useful for provision of a central course documents area, and the latter being useful as a digital drop box that relies on the file creation date to timestamp submission of the dropped file. Furthermore, a collaborative area also could be provided via the use of POSIX groups; by adding each user to course-specific groups, then providing a set of directories with group read-write permissions for each, students who are members of particular classes could view and modify each other's files in course-related directories. This feature would be useful in a course, such as Introduction to Engineering, which includes development of collaborative ability in its desired outcomes.

Final Design

The final design of the uLab system adds one diskless terminal server node (node001.cluster90.edu) to the cluster, and installs the RDP forwarder on the central server (master001.cluster90.edu). In addition, the following software packages are installed because of this particular laboratory's focus on computer engineering and FPGA-based hardware design:

- TDE
- QtOctave
- WXMaxima
- KPicoSim
- LTSpice
- gEDA
- LASI
- Xilinx ISE
- Xilinx EDK
- LibreOffice
- GIMP
- uLab Client

Scalability and Reliability

Scalability in this portion of the system primarily is limited by the RDP forwarder itself. The scalability may be increased by utilizing multiple RDP forwarder servers, with a specific forwarder being selected on initial user connection through round-robin DNS or a similar load-balancing scheme. All other aspects of the terminal services component should scale linearly with the number of available terminal service nodes. From a reliability standpoint, the RDP forwarder servers present single points of failure with the potential to disconnect large numbers of students if one or more of the servers were to fail. Although the students' sessions would continue to run on the RDP backend nodes, students would need to reconnect through a different RDP forwarder in order to reestablish a connection to their sessions. This process should be handled transparently by the load balancing scheme in use; however, there would be an undesirable, although transient, interruption of the users' sessions upon RDP forwarder failure. Failure of a single terminal service node is less catastrophic on a global level, although more serious on a local level. If a terminal service node were to fail, all connected users would have their sessions immediately terminated, potentially losing data in the process. In many respects, this would have the same user-visible effects as if someone had pulled the power cord on a stand-alone desktop system. However, unlike an XRDP forwarder failure, a terminal service node failure only would affect a small number of users. Therefore, it is recommended that the XRDP forwarder utilize high-availability hardware, whereas the XRDP backend servers may utilize less reliable–although much more powerful–hardware that is suitable for the computationally demanding loads encountered.

uLab Remote Hardware Access System

The third and final portion of the new remote laboratory system is the software that enables direct access to laboratory hardware. Because the students will be using this portion of the remote laboratory as a substitute for direct, hands-on access to laboratory hardware, careful thought was given to the design of this component. Additionally, because a wide range of laboratory hardware is available, hardware-specific servers and client components were utilized where possible. The overarching concerns of scalability, reliability, and security are no less relevant here than elsewhere within the system; as such, the fundamental architecture of the hardware-access system is designed to reflect these goals.

Design goals for the uLab remote hardware-access system include modularity, intuitive GUI controls, user access control, encrypted links, scalability, reliability, and low cost. The system allows third parties to easily develop new hardware interface servers and related GUI clients, thereby allowing new hardware devices to be provided with appropriate GUI interfaces within the hardware-access software. Another major goal of the new hardware-access system is to break the observed reliance on closed-source software, such as LabView, and, therefore, to provide a complete open-source solution from the uLab client GUI all the way down to the physical hardware devices within the laboratory. A final goal is to utilize inexpensive, open-source, hardware-interface computers instead of the previously utilized proprietary hardware-interface cards and machines. The hardware-interface computers usually are one of the most expensive components of the laboratory, along with the laboratory hardware itself, simply because of the number of interface computers required to support any significant number of

simultaneous users of the remote laboratory. Usage of inexpensive, open-source hardware for the hardware-interface computers is, therefore, highly desirable, especially in education where peak simultaneous users likely will be relatively high when compared with the average number of simultaneous users.

Authentication, Authorization, and Encryption

To accomplish the stated design goals, the hardware access system utilizes Kerberos extensively. A central arbiter daemon provides a uLab Kerberos service, and each client (including hardware-interface servers) must present a valid Kerberos ticket on initial connection. Upon successful Kerberos authentication, the communications channel immediately is switched to encrypted mode for security purposes. The lowest levels of this functionality are broken out into a new library (tdekrb) that provides an easy-to-use, frame-based data transfer method to higher-level applications while transparently handling Kerberos-based authentication and channel encryption. The use of Kerberos tickets in this application ensures that the hardware-access client can utilize the credentials provided on initial login, thus avoiding the need for the user to re-enter login credentials when starting the hardware-access client. If the user does not log on via the provided terminal services, then he or she will need to request a valid Kerberos ticket before being allowed to use the remote-hardware services.

The central arbiter utilizes a MySQL database to store its configuration information. As such, it also reads a small configuration file on initial startup to obtain database login credentials and its Kerberos service name; the former is required because MySQL lacks Kerberos integration

at the time of this writing. The preferred solution is for the arbiter to authenticate to MySQL through GSSAPI, and this should be implemented once support is available in either MySQL or the newer MariaDB packages. The configuration database contains information on each available workspace, including type, hosts and services, the client part associated with each service, workspace access control, and active workspace reservations. The central arbiter checks each authenticated, incoming service-access request against the permissions database to ensure that only authenticated users are allowed to access the hardware-access daemons for which they have permission. Upon detection of a request to access a disallowed resource, an authorization failure message is sent and the connection is terminated immediately; this happens without establishment of a connection to the requested hardware access daemon. This process effectively prevents an anonymous Distributed Denial of Service (DDoS) attack against the hardware-access servers themselves.

Hardware-access servers utilize the same authentication methods and encrypted links as the hardware-access client; therefore, the hardware-access servers can be placed safely away from the remote laboratory cluster if desired. The central arbiter utilizes a persistent, non-expiring Kerberos ticket to identify itself to each hardware-access server; this prevents a rogue or malicious arbiter from utilizing hardware resources to which it has not been granted access. If a hardware-access server becomes compromised, the damage would be limited to any directly connected hardware and/or the hardware-access server itself, presenting, in the worst case, an effective Denial of Service (DoS) attack against a single laboratory workspace. This design allows inexpensive hardware-interface servers to be placed directly on site, for example where bulky or sensitive equipment is present; in extreme cases, this could allow laboratory hardware

located on the other side of the world to be safely and securely utilized by users of a given remote laboratory cluster.

Client Design

The hardware-access client is primarily a container application into which GUI client "parts" can be inserted. The container provides an MDI container window and status bar; the latter may be changed by the active client part to present informative status messages to the user. Additionally, toolbars and menus are provided from which installed GUI parts may be launched as desired. These toolbar buttons and menu items automatically change, based on the type of remote workspace the user has selected; for example, an FPGA development workspace might show the FPGA programmer and FPGA viewer parts, while a process-control workspace might show sensor plotter and PLC programmer parts. Each part communicates with the central arbiter, utilizing a rigidly defined protocol. If the client part is authorized to use the requested hardware on initial connection, then the central arbiter contacts the appropriate hardware-access server and routes the client connection to that server. This architecture allows multiple, identical workspaces to be provided within a given laboratory with the system, not the user, deciding which particular workspace will be utilized for a given connection; this type of architecture provides redundancy and masks from the end user any potential hardware-access server failures.

The client MDI container handles initial connection to the central arbiter, and on initial connection receives a list of available workspace types to which the user has been granted access. It presents this list to the user; then, after the user selects a workspace type, the client

MDI container requests a workspace reservation, matching the selected type, from the central arbiter. If all workstations of that type are in use, the central arbiter will respond with a busy code, and the client will prompt the user to try again later. Otherwise, a reservation for a specific workspace of the selected type is entered into the system, and will remain valid until the initial connection to the arbiter has been terminated, either through a client-initiated disconnection or through the action of a laboratory manager. The central arbiter keeps track of these reservations and, on establishment of a connection by a client part, will ensure that the reserved workspace receives the client part's connection request. The client MDI container is designed to take a single command line parameter that specifies the DNS address of the central arbiter of the cluster; this can be used to hide the implementation details from the user and present an "instant-on" hardware-access interface when used with existing Kerberos tickets from the initial login. If this command line parameter is missing, the client will prompt for the address of the central arbiter to which it should connect and authenticate. All of these details are hidden from the client parts, simplifying development of new client parts and ensuring that system security is maintained at all times. An active session with multiple active client parts is shown in Figure 5.

Server Design

Each client part connects to a specific hardware-access daemon that runs on a hardware-access server. In turn, each hardware-access daemon is assigned a specific port on a given host; this architecture allows one workspace to be assembled from either one server running one or more hardware-access daemons, or from multiple hardware-access servers, each running one or

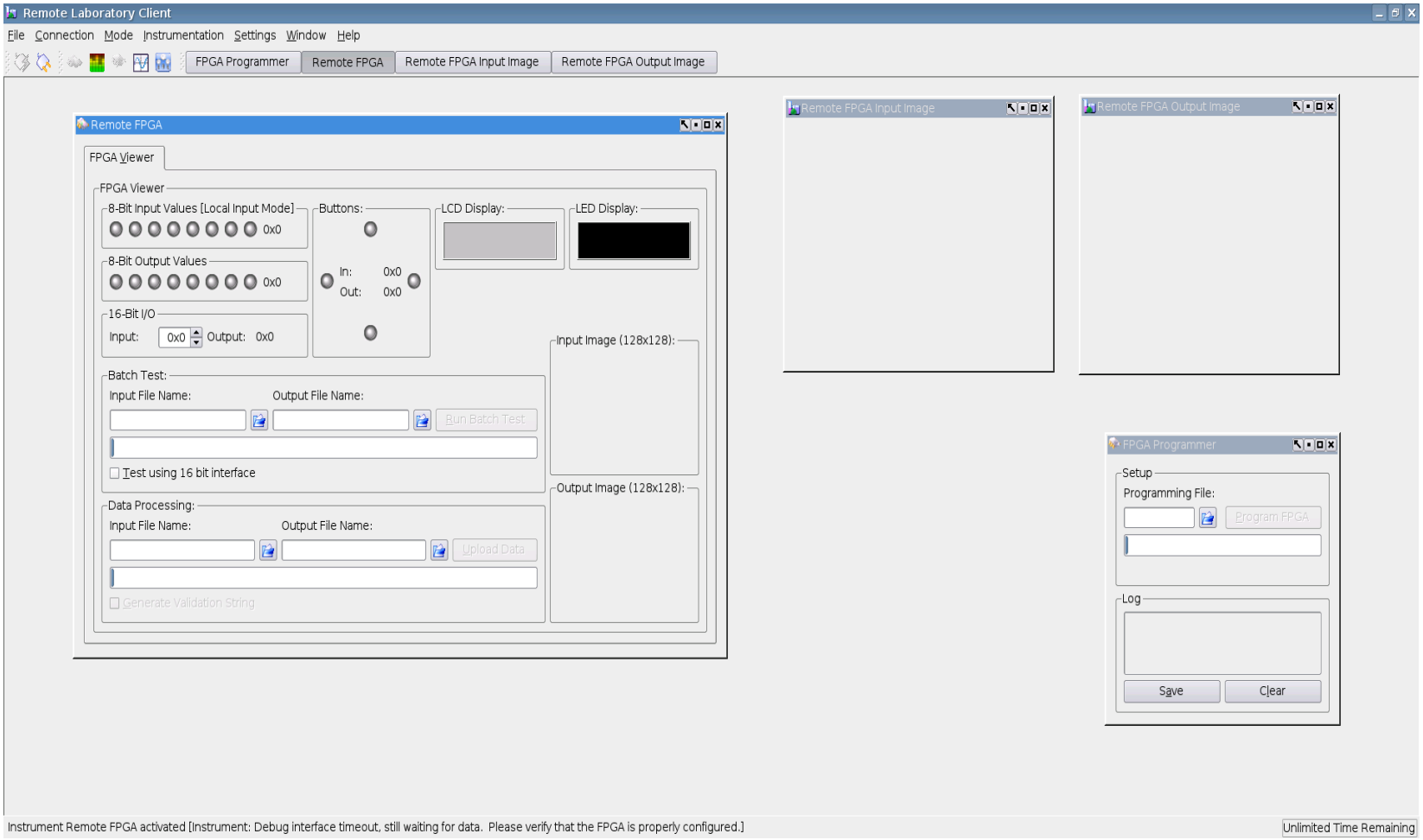

Figure 5: A typical uLab FPGA development workspace

more access daemons. Each access daemon opens a Kerberized server socket and expects connections to be made from the central arbiter; if the provided Kerberos credentials do not match the known arbiter credentials, the connection is immediately terminated. As with the central arbiter, each hardware-access daemon follows a strict protocol for identification and connection setup; after this process is completed, arbitrary data may be transmitted between the client part and the hardware access daemon until the connection is terminated.

Protocol Design

The protocol between the arbiter and either clients or hardware-access servers is rigidly defined to ensure that clients and the arbiter are able to quickly establish and verify functionality of new connections. Although the authoritative protocol definition is provided in a document within the Git [88] tree of the uLab system, this section will give an overview of the protocol and the design decisions that influenced its final form. It was realized early on that some form of frame-based transfer would be needed, as most experiments produce somewhat packeted data, such as related groups of instrument readings, that cannot be processed reliably or displayed until the entire data chunk has been received. Furthermore, usage of the encryption system provided by SASL strips away any packet-length information that normally would be present if the underlying TCP/IP protocol were directly utilized; therefore, the decision was made to sacrifice a single bit of every byte to control characters, such as the end of frame indicator, through the use of Base64 encoding. In practice, given the relatively small bandwidth of the hardware-access data streams as compared with other services, such as the terminal servers, the 12.5% bandwidth

loss to this encoding scheme is insignificant. Using frame-based transfer ensures that all the requisite data arrives before the client part begins to process it, increasing reliability and eliminating a major source of network-related deadlocks. Usage of this protocol is mandatory because the central arbiter decrypts all incoming streams and re-encrypts them using its own keys; this process slightly enhances security by not allowing the clients to access cryptographic material related to the hardware-access servers, and vice versa.

In an effort to make development of new client parts and hardware-access daemons as easy as possible, the TQt-specific TQDataStream class was utilized extensively throughout the hardware-access system. This class enables transmission and reception of TQt objects, such as TQString and TQByteArray, with a single line of C++ code, enhancing code readability and simplifying development. If access is needed from a different toolkit, the data structures sent and received by the TQDataStream class are well documented, and the appropriate interface code could be written as needed. The binary stream input and output of each TQDataStream class is connected with the tdekrb library, which handles encryption, framing, and raw network input/output, as detailed above and illustrated in Figure 6. Usage of this class allows network operations to be treated as a simple passing of objects between the server and client, thereby hiding much of the underlying complexity of the hardware-access network protocol from the application developer. While usage of a TQDataStream-based protocol is strongly encouraged, its usage is not mandated past initial connection establishment between a client part and its associated hardware-access daemon. If desired, any binary protocol may be substituted once an initial connection has been fully established; it also may be utilized without interference from the arbiter until that particular connection is terminated for any reason.

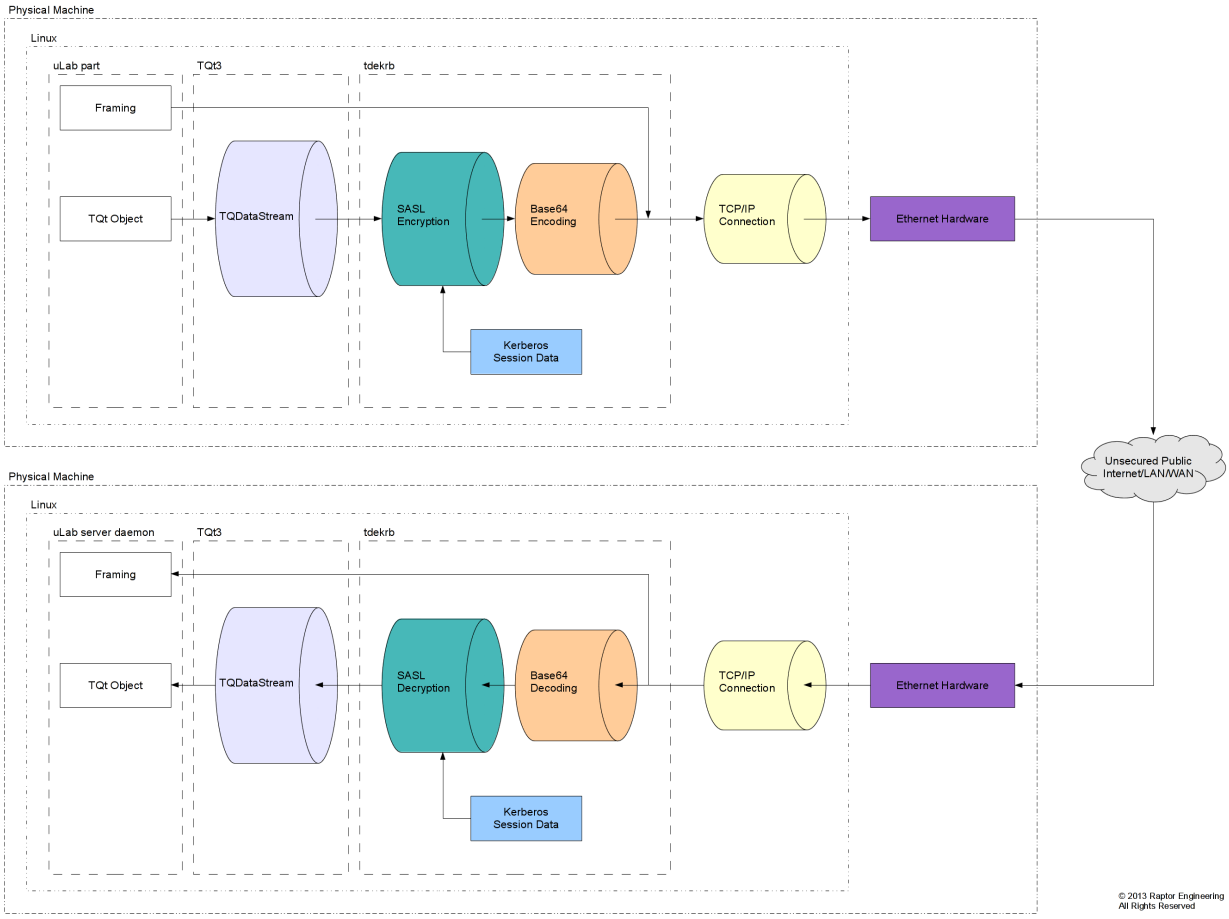

Figure 6: Hardware access protocol encapsulation for transmission over unsecured networks

The hardware-access system currently includes several client parts and hardware-access daemons that will find use in most electrical engineering laboratories. A GPIB-based oscilloscope and spectrum analyzer part is provided, along with an I2C sensor plotter part. All three parts allow export of captured data to external files for later analysis, and import of external data files for viewing. FPGA access is included via two parts, one for programming the FPGA and one for interacting with the design after it has been loaded into the laboratory hardware. On the management end, two parts have been provided: one to manage authenticated user access permissions, and one to view and control both active terminal service and active workspace users. The latter part, in particular, allows a laboratory manager to set session timeouts and even disconnect users that have, for example, been idle for too long or who might not be using laboratory resources in accordance with an institution-specific acceptable use policy. Following UNIX tradition, each part has been designed to do one thing and to do it well. This separation of duties not only allows the maintainer of each part to stay within his or her area(s) of expertise, but also has the effect of dividing loosely related functions into separate GUI windows. For example, the FPGA programmer is separate from the FPGA viewer, allowing the programmer to be minimized or obscured when testing the FPGA, and vice versa. Alternatively, on large screens, both may be visible at the same time, with the user choosing where each part should be located on his or her screen for maximum usability and efficiency. By following this UNIX tradition, the user is granted more control over how he or she sets up and uses his or her workspace.

Test Equipment Interface

The GPIB interface part, unlike its distant predecessor utilized in the RemoteFPGA system [42], handles all display and processing of the raw instrument data on the client end. This increases responsiveness and enables real-time operation; unlike the older system, which captured raw screen shots of the instrument displays, the uLab system essentially comprises a complete instrumentation front end, similar to the interfaces that have been integrated with stand-alone test equipment since the beginning of the digital test equipment era. This allows the backend test equipment in use to be fully abstracted from the user; aside from various hardware-driven specifications, such as number of traces and bandwidth, each major type of test equipment added to the system will present the same generic graphical interface to the user. This also enables the possibility of using "headless" test equipment, such as some of the more recent PC-based oscilloscopes that do not contain a display or physical keypad, in the laboratory to reduce overall cost. In addition, it also enables the repurposing of specialized hardware for more general purposes without the added complexity this often brings; for example, the spectrum analyzer server included in the uLab software package interfaces with an Agilent CDMA test set, allowing use of the spectrum analyzer functionality buried within that specialized equipment without requiring the user to first understand how to operate the basic functions of the test set. The oscilloscope part is shown in Figure 7; this image was taken while using the author's four-channel, GPIB-capable Tektronix oscilloscope to capture and analyze a high-voltage transient.

It should be noted that the general principles discussed herein are not only applicable to GPIB. The author had a need to access GPIB-based test equipment at this time of this writing,

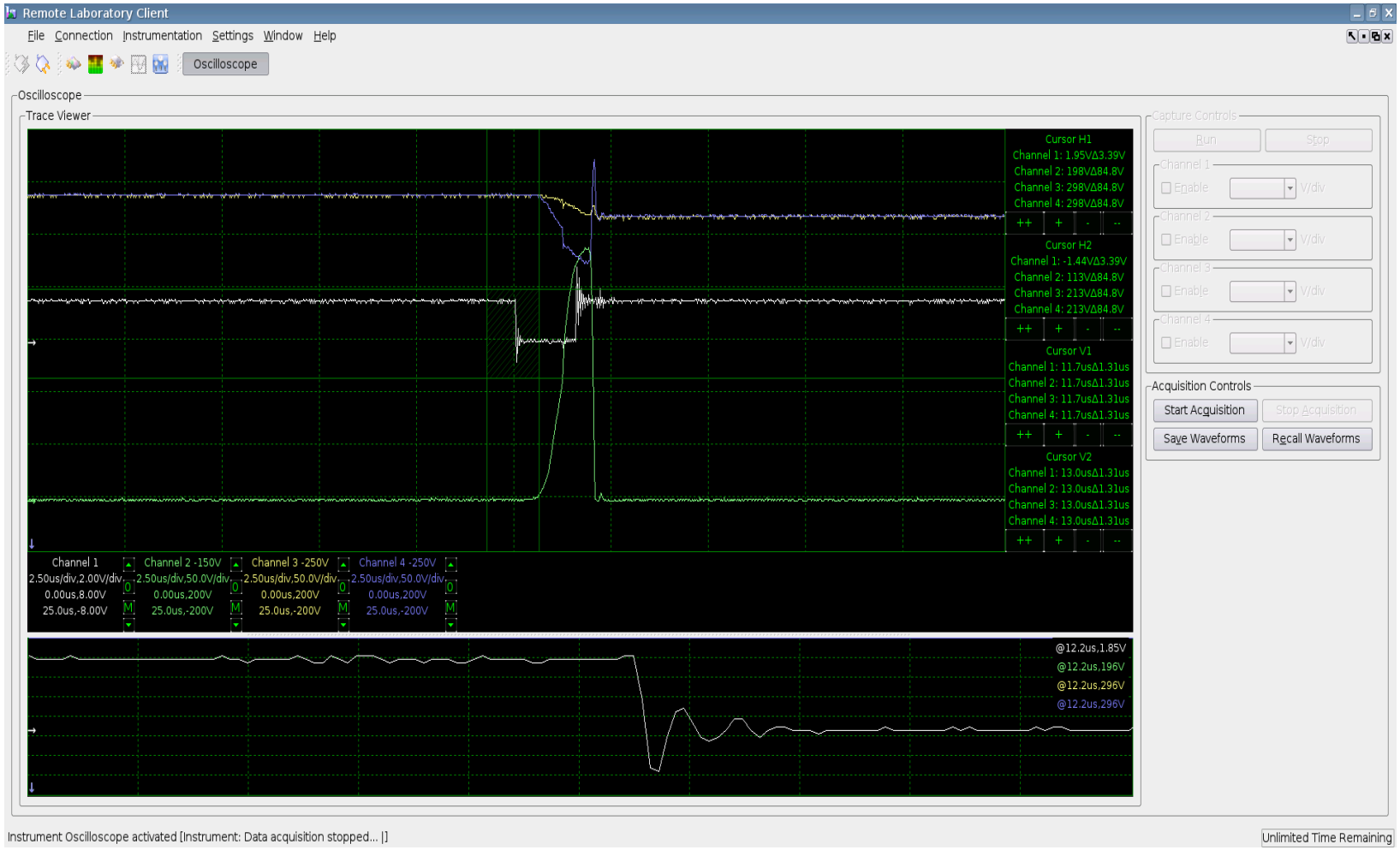

Figure 7: The uLab oscilloscope frontend part, showing a high-voltage transient and the use of a zoom region

hence the inclusion of GPIB-access daemons in the uLab software, but the principles documented above are applicable to newer interfaces as well. As long as the manufacturer provides a programming reference manual that is not protected by a non-disclosure agreement (NDA) or similar legal instrument, new backend server daemons can be written to interface with test equipment over almost any hardware interface. In fact, the provided GUI client parts may be used with new backend daemons utilizing a non-GPIB hardware interface. Special effort was made to ensure that the test equipment client parts use a generic protocol that should be applicable to all test equipment within a particular class; therefore, for example, the oscilloscope client part and protocol should be usable with any type of oscilloscope backend interface daemon. GPIB interfaces were implemented primarily because of the low cost of GPIB-enabled test equipment, the fact that a GPIB interface is all that typically is required to obtain adequate functionality from oscilloscopes and other signal analyzers, and the fact that most GPIB-enabled equipment manufacturers freely provide protocol documentation for those instruments. Newer equipment interfaces, such as USB, tend to be far more restricted and poorly documented, if publicly documented at all; this may be part of an effort to force the acquisition of expensive interface software licenses, simple oversight on the part of equipment manufacturers, or some combination of both. Regardless of motives, this effectively prevents usage of new, low-end test equipment from certain manufacturers in a laboratory of this type. Fortunately VXI-11, a relatively new, publicly documented industry standard [89], is coming into use by major corporations, such as Tektronix, LeCroy, and Agilent. Additionally, an open-source Linux library supporting this protocol is available for use [90]. VXI-11 is effectively a full, proper replacement for GPIB, utilizing Ethernet connectivity and providing a fully documented

command-based interface to the test equipment. The author recommends that only GPIB, VXI-11, or similarly fully open and documented equipment be utilized in new remote access laboratories, as this will not only ensure reliable operation of the resultant laboratory, but also will send a strong message to test equipment manufacturers regarding the continuing need for open-protocol access to test equipment.

FPGA Viewer Part

Because this particular laboratory will be used primarily for FPGA design, the architecture of the FPGA viewer part will be discussed. The FPGA viewer is essentially a full rewrite and extension of the original FPGA remote access solution created by the author and deployed in 2009 at NIU [42]. It uses a hardware abstraction module, contained within the students' FPGA design, that interacts with a given set of signals. This abstraction module then transmits current signal levels to the hardware-access server via a standard serial link, while allowing a second set of signals to be controlled by the hardware-access server via the same serial link. The hardware-access server forwards this basic serial communication protocol over the network to the client part, which then interprets the data and displays the current status of the virtual lights, switches, and displays. The abstraction module also includes the ability to read and write to either internal block RAM or external SRAM; when interfaced with the client part, this allows block-data-based algorithms, such as image processing, to be implemented on the FPGA, then to be tested easily using the FPGA viewer client part. A final feature, implemented within the client part itself, is the ability to run a batch test and record the results. This batch-test

feature takes a list of inputs from a simple text file, sequentially applies them to the 16-bit data bus, and records the results in a second text file. The hardware abstraction module itself did not need alteration to work with the newly rewritten hardware-access daemon and client part; therefore, it is the only portion of the original remote access solution still in use as of this writing. The internal architecture of this FPGA debug module is shown in Figure 8.

While the FPGA interface could be implemented with a second, dedicated FPGA, or possibly even utilizing JTAG to directly read and set FPGA pin states, the described solution contains several benefits when compared to these other alternatives. A second, dedicated host FPGA would both add expense to the system and require a rich interconnect to the target in order to be useful. Even more problematic, JTAG pin commands are not standardized across FPGA manufacturers, or even FPGA types, and are often considered trade-secret or proprietary information. By contrast, the described access solution allows abstraction of many different FPGA types because it does not utilize any external pins to link with the user's hardware design; stated another way, the user can create a generic FPGA hardware design and rely on the hardware abstraction module for all input/output. This access method also allows the creation of virtual hardware resources; for example, the provided module emulates both an LCD display and a 7-segment LED display, allowing the student to learn about both display types and, in fact, to design hardware that is capable of driving them, without requiring a physical 7-segment or LCD display to be attached to the FPGA hardware. The results of driving the provided interface signals are displayed in real time on the FPGA viewer client part, as if the student were observing a physical display on a development board in the laboratory.

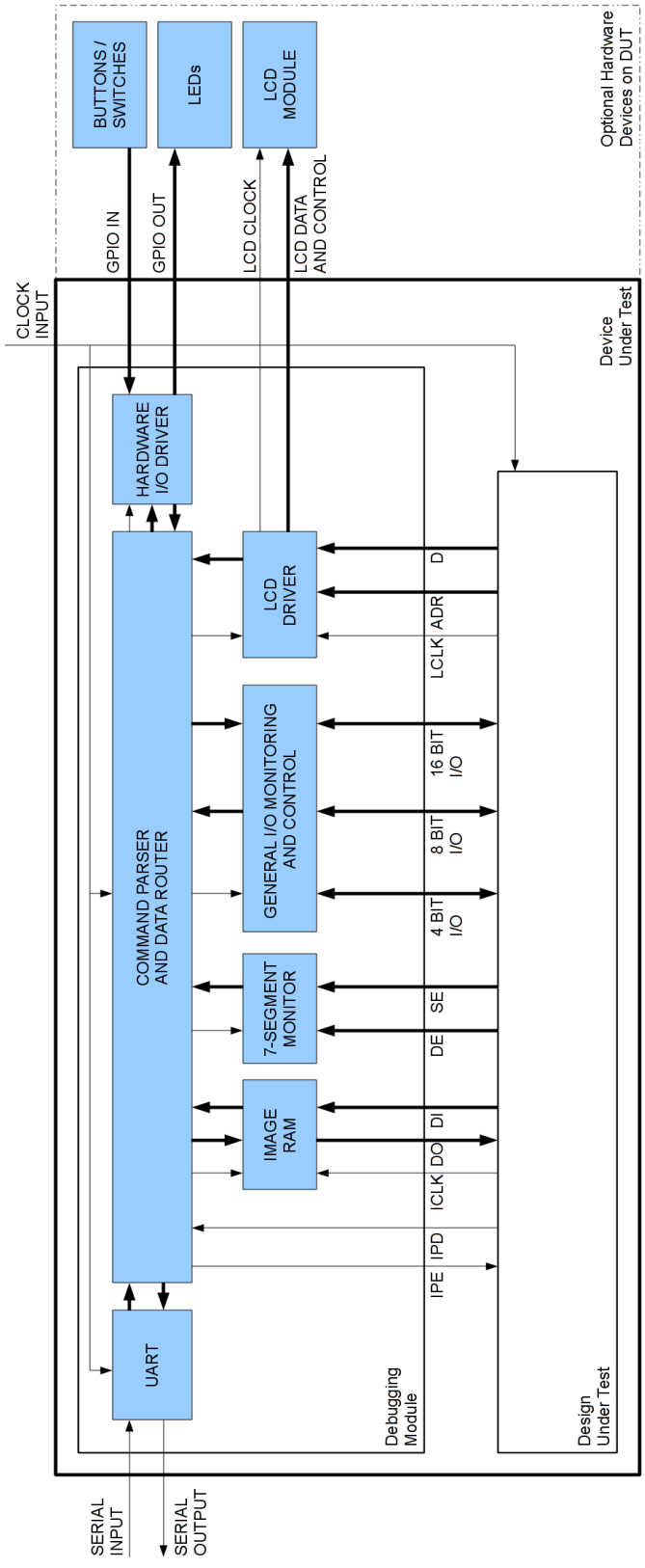

Figure 8: FPGA debug module internal structure

FPGA Programmer Part

The FPGA programmer is a radical departure from previous methods. Prior to this remote laboratory, the author was forced to use Xilinx's iMPACT CableServer running on dedicated i386 machines with no encryption or access control. Furthermore, this original system required the use of expensive, Xilinx-specific programming cables and pods to function. Much of this was due to the obscurity of the JTAG commands needed to program a .bit file into a Xilinx FPGA; the iMPACT software was the only widely known software package able to properly program the FPGA. Since then, the author has become aware of a small suite of tools which can convert the .bit file into a series of JTAG instructions contained within a Serial Vector Format (SVF) file. The SVF file can, in turn, be "played," or its instructions executed, over a JTAG link, via a small open-source SVF player application. These two applications provided the first hints that the prior dependency on the closed-source Xilinx programmer could be broken; however, an expensive JTAG pod and i386 machine still are needed to make this solution work.

Late last year, an interesting new development tool, called the Raspberry Pi, became widely available. The Raspberry Pi is a low-cost, low-power, single-board computer built around a 600MHz ARM processor. It provides user-accessible GPIO, two serial ports, two USB ports, and a network connection; it also can utilize stock Debian Linux. Together, these features make the Raspberry Pi a nearly perfect hardware interface server [23]. Upon obtaining one of these devices, the author ported the SVF player to use the GPIO pins of the Raspberry Pi, then connected a Spartan 3 FPGA to the GPIO interface, and was rewarded with a working FPGA programming solution that was built entirely on open source. The open-source .bit to SVF

conversion utility was subsequently and rapidly ported to the new Spartan 6 series of FPGAs through the use of a standard Spartan 6 FPGA development kit. Once the Raspberry Pi FPGA programmer concept was proven, a simple Kerberized access daemon was written, and its associated client part was created. Afterward, the FPGA viewer was tested over the built-in serial port of the Raspberry Pi, proving that the Raspberry Pi can be considered a complete replacement for the previously used i386 machines and their associated JTAG interfaces.

The Raspberry Pi allows a complete FPGA hardware-access server and target hardware to be contained in a small, low-power package, referred to hereinafter as a "pod." These pods can be generalized to support any type of hardware, as long as any required interface software can be compiled and executed successfully on the ARM architecture. Due to the small size and low cost of each pod, providing sufficient remote hardware resources to accommodate an entire class of students should prove both practical and economical. Furthermore, the fully open-source nature of the pods allows customization to suit any needs within a given laboratory, and allows immediate patching of any bugs or security holes that may be encountered without having to wait for an update from the device manufacturer. Pods may be either diskless or contain their system files on an integrated Secure Digital (SD) card. Since the FPGA pods in the uLab system are contained within the cluster, they have been installed, along with the terminal services, as diskless nodes to prevent unnecessary wear and tear on the Flash-based SD cards. A photograph of a single Spartan 6-based pod is shown in Figure 9. The Raspberry Pi is mounted on top of the Spartan 6 development board, with the USB connection to the integrated UART of the FPGA visible on the right and the JTAG/power wiring shown in the middle of the photograph. The flying leads terminating on the left side of the photograph are normally used to provide electrical

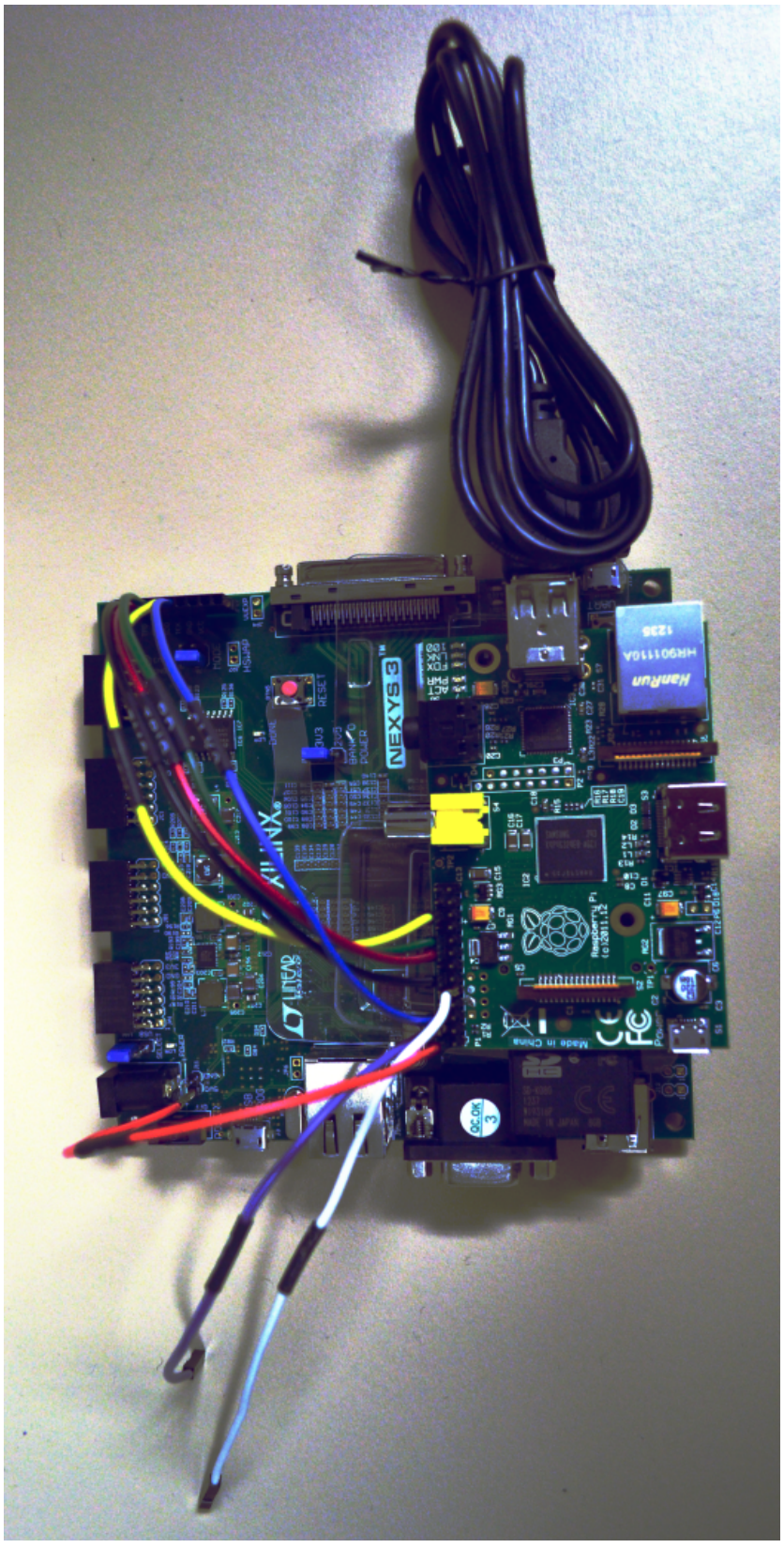

Figure 9: Single Spartan 6 hardware development pod

power to a second pod, reducing the number of separate power supplies required for the laboratory system installed at Northern Illinois University.

Final Design

The final design consists of five Spartan 6-based FPGA development pods (pi001.cluster90.edu through pi005.cluster90.edu), with the central arbiter installed on the master server (master001.cluster90.edu). Each pod consists of a diskless Raspberry Pi and a Digilent Spartan 6 development board, with the Raspberry Pi taking its 5V power directly from the Digilent board. Each Raspberry Pi utilizes a specially created SD card image that contains a read-only boot partition and the requisite kernel command line to enable booting via NFS. Each pod is joined to the Kerberos realm through the use of the TDE Realm Bonding utility, as is the machine running the central arbiter. All services are automatically started on boot via the standard Debian startup sequencing (init) system, with time from pod power application to full service availability typically several minutes. Machine startup order is not critical, with the exception that the master node (master001.cluster90.edu) be started before any other nodes because it contains all of the services needed for diskless client startup and operation. If diskless clients were not utilized, the servers and nodes could be started in any order, although portions of the system would remain unavailable until sufficient resources were brought online.

Scalability and Reliability

Scalability of this final component of the uLab remote laboratory is, like the terminal services component, limited primarily by the performance of the central arbiter. Similar to the terminal services discussion, the best solution for mitigating this bottleneck is the provision of several arbiters utilizing round-robin DNS or a similar load-balancing scheme. Because all reservation and session information is stored in a central MySQL database, many such arbiters may be employed to distribute the network and CPU load across multiple machines. Each arbiter must contain a copy of the central arbiter Kerberos ticket; therefore, each arbiter is a potential security risk and must be physically secured against attack. If any arbiter were to be breached and its persistent Kerberos ticket obtained, the attacker would gain access to any hardware-access servers that originally were linked to that arbiter. This risk can be mitigated somewhat through the use of several unique Kerberos tickets, one for each arbiter. While this will not be of benefit until an attack has been detected, it will prevent the entire system from going offline while new Kerberos tickets are generated. If this scheme is used, the compromised ticket simply could have its access revoked on each of the hardware-access servers–without requiring revocation of other arbiter tickets and subsequent temporary loss of access from uncompromised arbiters.

All other components of the remote hardware access system should scale nearly linearly with the number of available hardware-access servers. If a single pod were to fail, a spare of the same type could be brought online and added to the hardware access pool, thereby minimizing any user-visible work interruption. However, if a pod fails, the user will lose all state

information from the hardware he or she was previously using; while saved data and displayed results would not be affected, this could affect certain types of long-running experiments. The availability level of the pod hardware should, therefore, be matched carefully to the types of experiments being run on that particular pod to ensure a satisfactory user experience. If a central arbiter were to fail, the effects would be far more widespread and catastrophic. All students using that arbiter would be disconnected, and all laboratory workspaces to which the arbiter was connected would be reset to defaults, with all reservations made through that arbiter being immediately released. To minimize this risk, the central arbiter(s) should be installed on high-availability hardware that runs a highly stable operating system.

Miscellaneous Services

Several optional components of the system, such as the Xilinx ISE/EDK and MATLAB, require a dedicated license server in order to function correctly with floating licenses. This license server, if needed, should be installed in a dedicated virtual machine that runs one of the manufacturer-supported Linux distribution versions. VirtualBox [91] is an excellent open-source virtualization solution, and is recommended as the virtualization host. Similar to other central points of failure, if the license server were to fail, students would be unable to access any software packages that relied on it. Consequently, high-availability hardware should be used to support the virtualization host; in the final laboratory design, the ideal host is the master server (master001.cluster90.edu), although larger installations may see improved performance if a single high-availability machine is dedicated as a virtualization host. It is strongly

recommended, although not required, that a physical network port be dedicated to the license server's virtual machine; this ensures that the MAC address of the license server is never confused with the MAC address of the host machine, and makes replacement of failed host hardware somewhat easier.

DISCUSSION AND SUMMARY

Because a major goal of this research is not only to prove that a fully open-source remote laboratory can, in fact, be built, but also to ensure the sustainability of such an environment for multiple institutions to use, all source code written for the uLab system has been released in a set of Git trees. At the time of this writing, the uLab system uses donated space in the TDE project's infrastructure for bug tracking and patch submission; however, this may change in the future as the project grows more popular and more individuals begin to contribute code and resources. As of this writing, project code repositories are available at http://ulab.trinitydesktop.org; authoritative protocol documentation for developers also is included in a text file within the hardware-access package Git tree. Additionally, a highly condensed set of installation instructions for a lab using the design detailed herein is available at the same location. These instructions are designed to be read and understood by a Linux system administrator or similarly qualified individual, and assume familiarity with UNIX-like systems, the Linux command line, and various configuration files for several software packages. The aforementioned Kerberos realm setup and management tools have been integrated into the TDE project at http://www.trinitydesktop.org, and therefore are available in both source and binary form for TDE R14.0.0 and above.

This research largely accomplished its original goals, described at the beginning of this work; a functional, secure remote laboratory has been built on fully open-source software that also can support proprietary tools where needed, such as those required for FPGA synthesis and

for FPGA routing. Direct access to dedicated hardware is provided through the use of the uLab client container, which hosts client parts tailored to the needs of particular hardware types. Inexpensive laboratory hardware "pods" were created to make deployment of a large laboratory economical and practical; five of these pods were installed in the laboratory at Northern Illinois University. All pods host Spartan 6 FPGA development boards sourced from Digilent, Inc. A single terminal services node with 12 Opteron cores and 24GB of memory was installed in the laboratory, and remote desktop functionality, including Kerberos-based single sign-on, was verified. Finally, the new Kerberos realm control tools and other infrastructure components were installed on two dedicated servers, and full functionality of the infrastructure components, including the disk-related Infiniband links, was verified. Link speed was identified as a critical parameter, and students with downstream link speeds below approximately 3Mbps may experience difficulty using the remote laboratory. This may be optimized in the future as the FreeRDP project continues to work on the xrdp servers; however, further research needs to be done to identify the source of observed bandwidth surges and related interface stalls on slower connections. A block diagram of the completed laboratory, as installed at Northern Illinois University, is shown in Figure 10, and a photograph of this laboratory in its native environment is shown in Figure 11.

Future Work

While a capable remote laboratory system, fully usable for remote FPGA development, has been presented, there are several opportunities for enhancement present within the uLab

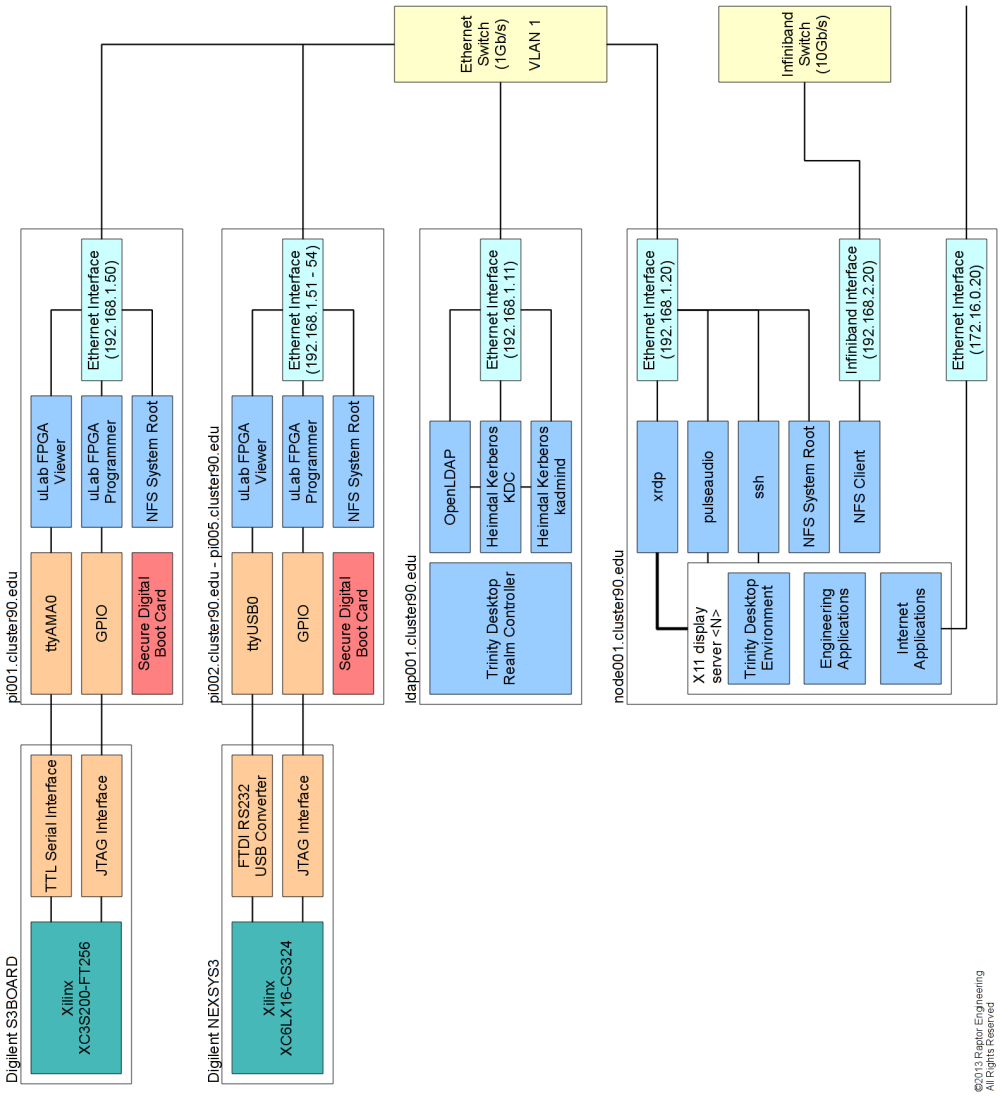

Figure 10: Internal structure of the uLab installation at Northern Illinois University (continued on following page)

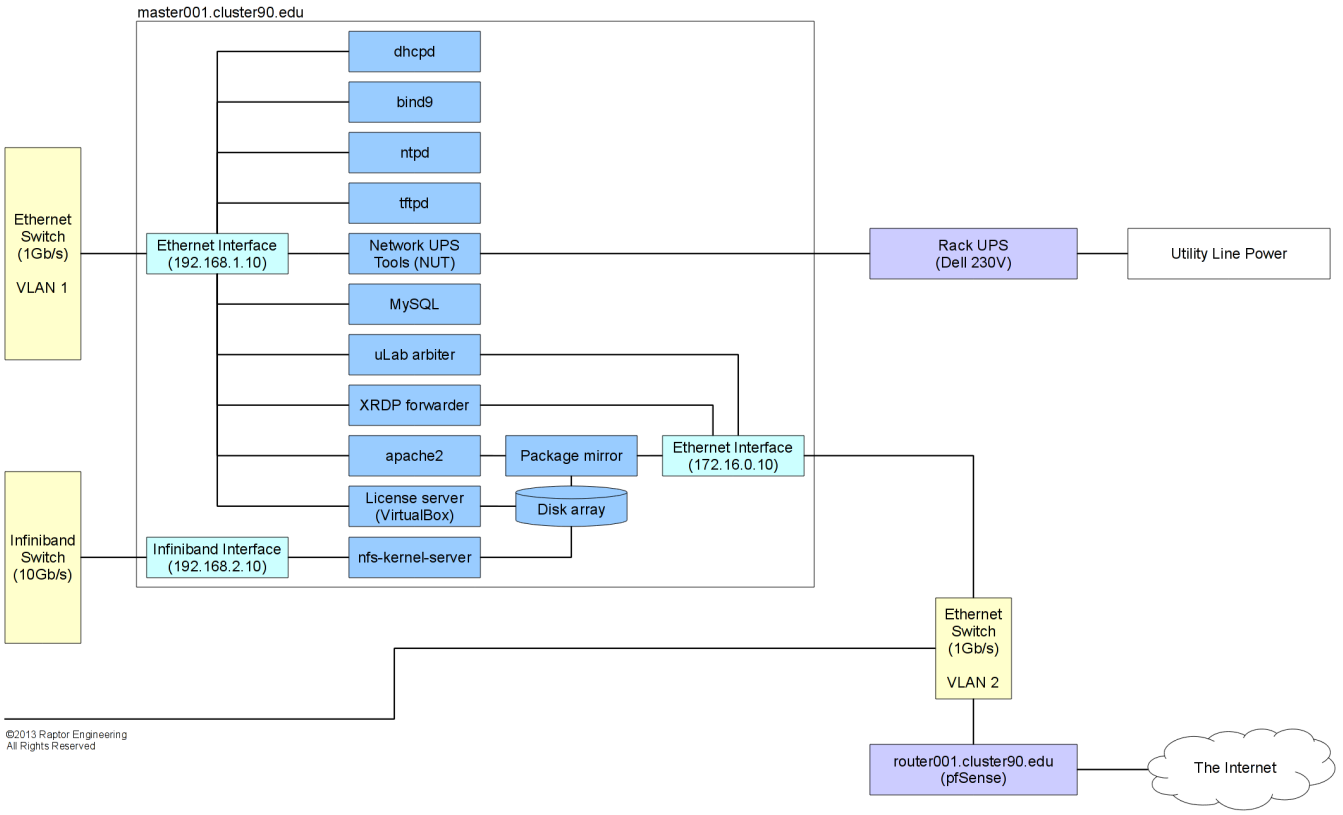

Figure 10: Internal structure of the uLab installation at Northern Illinois University (continued from previous page)

Figure 11: uLab installation at Northern Illinois University

system at the time of this writing. The laboratory, as currently installed at Northern Illinois University, does not include signal generation or waveform capture equipment; future development should concentrate on the creation and interfacing of a fully open-source and open-hardware oscilloscope pod. The author is aware of at least one FPGA-based oscilloscope project which fits these criteria; a developer interested in accomplishing this task would need to create the interface daemon between the oscilloscope module and the uLab Kerberized network. The existing GPIB interface server can be utilized as a template to reduce the overall work needed to create the interface server, while the client part should need no modification, as discussed earlier. In a similar manner, a fully open-source and open-hardware signal generator pod may be created, although such a pod will require the writing of an interface daemon and a client part, since no client part currently exists for a signal generation tool. Both pods preferably would utilize a Raspberry Pi as the test-module-to-network interface in order to lower overall laboratory cost and power consumption.

A touch screen was specified for eventual use in the laboratory in order to provide a simple interface that could be used by a laboratory manager to accomplish basic management tasks, such as viewing the laboratory state, terminating user sessions if necessary, and marking machines and pods as online or offline. This interface was not developed as part of this thesis; therefore, it presents an opportunity for anyone interested in touch-based interfaces, namely to create a functional and efficient laboratory management system. It should be noted that the use of touch in this area is not in conflict with previous statements related to the superiority of mouse-and-keyboard driven interfaces for engineering; rather, laboratory management, as with other industrial control and automation applications, is simply an area in which the high I/O

bandwidth of a mouse-and-keyboard-driven interface is not required. This touchscreen interface should run on a dedicated, low-power system, such as the Raspberry Pi, and utilize a secure form of passwordless, Kereberized authentication, such as that provided by SmartCards and their readers. Since TDE includes software to handle SmartCard logins, the existing TDE-based functionality simply could be extended as needed to accomplish this goal. It should be noted that the usage of an on-screen keyboard or any other means of on-screen authentication data entry is inherently insecure and should be avoided wherever possible.

Another area of possible future work relates to the integration of the uLab system with electromechanical laboratories, such as an electrodynamics laboratory. An interface pod could be used to control motors, switch gear, and other equipment in such a laboratory, while the oscilloscope pods, mentioned earlier, could be used as monitors. Where needed, standard GPIB-based test equipment, such as multimeters, could be added to the system via appropriate interface daemons and client parts. Video feeds of running electrodynamics experiments could be provided via a standard USB UVC-compatible webcam; this would require a dedicated camera interface daemon using V4L2 and a new client part for viewing video and audio.

A final area of possible enhancement relates to collaborative tasks on the remote laboratory system. It may be advantageous to provide a means for instructors to interact directly with their students in real time; such interaction could take the form of IRC chat, collaborative document editing, such as that available through an Etherpad instance, or even videoconferencing if link bandwidth is sufficient. All of these potential interaction methods are available through the use of existing open-source software; therefore, installing them should be a relatively simple matter. Once installed, however, they should be integrated with the Kerberos

realm in such a way as to retain the single sign-on experience, and must also be secured and encrypted to ensure that eavesdropping, spoofing, spamming, or malicious destruction of data cannot occur.

REFERENCES

[1] *The first computers: history and architectures*. Cambridge, Mass: MIT Press, 2000.

[2] Computer Terminal Corporation, "The Datapoint 3000."

[3] B. Ram, *Computer fundamentals: architecture and organization*. New Delhi: New Age International, 2000.

[4] J. P. Kanter, *Understanding thin-client/server computing*. Redmond, Wash: Microsoft Press, 1998.

[5] R. W. Bemer, "How to Consider a Computer," *Automatic Control*, pp. 66–69, Mar-1957.

[6] J. McCarthy, "REMINISCENCES ON THE HISTORY OF TIME SHARING." [Online]. Available: http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html.

[7] P. E. Ceruzzi, *A history of modern computing*, 2nd ed. London, Eng. ; Cambridge, Mass: MIT Press, 2003.

[8] W. J. Glenn, *MCSE: Exchange 2000 Server Administration Study Guide*. Hoboken: John Wiley & Sons, 2006.

[9] B. G. Lipták, *Instrument Engineers' Handbook, Third Edition, Volume Three: Process Software and Digital Networks*, 3rd ed. CRC Press, 2002.

[10] K. Johnson, "Chrome OS: Return of the Dumb Terminal," *T-GAAP*.

[11] G. B. Shelly and H. J. Rosenblatt, *Systems analysis and design*. Boston, Mass.: Thomson Course Technology, 2010.

[12] P. O'Connor, *Using computers in hospitality*. Australia; [Belmont, CA.?]: Thomson, 2004.

[13] L. Torvalds and D. Diamond, *Just for fun: the story of an accidental revolutionary*. New York, NY: HarperBusiness, 2002.

[14] E. S. Raymond, "The X Windowing System." [Online]. Available: http://www.catb.org/esr/writings/taouu/html/ch02s06.html.

[15] “A history of Windows - Microsoft Windows.” [Online]. Available: http://windows.microsoft.com/en-US/windows/history.

[16] T. Berners-Lee, “WorldWideWeb: Proposal for a HyperText project,” 12-Nov-1990. [Online]. Available: http://www.w3.org/Proposal.html.

[17] “cern.info.ch - Tim Berners-Lee’s original WorldWideWeb browser.” [Online]. Available: http://info.cern.ch/NextBrowser.html.

[18] S. Deffree, “NeXT Computer debuts, October 12, 1988,” *EDN Moments*. 12-Oct-2012.

[19] P. Raj, *Cloud enterprise architecture*. Boca Raton, Fla.: CRC Press, 2013.

[20] D. Kravets, “Guess What, You Don’t Own That Software You Bought,” *Threat Level*.

[21] B. Proffitt, “Free software for end users facing extinction?,” *Open for Discussion*. 19-Jun-2012.

[22] Copyright Office, *Exemption to Prohibition on Circumvention of Copyright Protection Systems for Access Control Technologies*. 2012, pp. 65260 –65279.

[23] “Raspberry Pi | An ARM GNU/Linux box for $25. Take a byte!” [Online]. Available: http://www.raspberrypi.org/.

[24] D. Gentner and J. Nielsen, “The Anti-Mac interface,” *Communications of the ACM*, vol. 39, no. 8, pp. 70–82, Aug-1996.

[25] “Has the KDE Social/Semantic Desktop been worth the hassle to anyone?” [Online]. Available: http://kde.6490.n7.nabble.com/Has-the-KDE-Social-Semantic-Desktop-been-worth-the-hassle-to-anyone-td1200181.html.

[26] A. Prlić and J. B. Procter, “Ten Simple Rules for the Open Development of Scientific Software,” *PLoS Computational Biology*, vol. 8, no. 12, p. e1002802, Dec. 2012.

[27] M. Syamlal, T. J. O’Brien, S. Benyahia, A. Gel, and S. Pannala, “Open-Source Software in Computational Research: A Case Study,” *Modelling and Simulation in Engineering*, vol. 2008, pp. 1–10, 2008.

[28] E. Schindler, “An Abbreviated History of ACP, One of the Oldest Open Source Applications,” *Great Wide Open*. 20-Aug-2009.

[29] *Open sources: voices from the open source revolution*, 1st ed. Beijing ; Sebastopol, CA: O'Reilly, 1999.

[30] R. Hillesley, "GNU HURD: Altered visions and lost promise," *The H Open*, 30-Jun-2010.

[31] "GNU General Public License v2.0 - GNU Project - Free Software Foundation." [Online]. Available: http://www.gnu.org/licenses/gpl-2.0.html.

[32] "Web Server Survey," Security Space, Research, Feb. 2013.

[33] "OS/Linux Distributions using Apache," Security Space, Research, Feb. 2012.

[34] "September 2012 Web Server Survey," Survey, Sep. 2012.

[35] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: the Apache server," pp. 263–272.

[36] R. Freeman, "Configurable electrical circuit having configurable logic elements and configurable interconnects," U.S. Patent 4,870,302.

[37] "Xilinx:: Our History," 02-Jan-2007. [Online]. Available: http://web.archive.org/web/20070102044237/http://www.xilinx.com/company/history.htm.

[38] "History of Xilinx, Inc." [Online]. Available: http://www.fundinguniverse.com/company-histories/xilinx-inc-history/.

[39] "15 Years of innovation," *Xcell*, no. 32, p. 64, 1999.

[40] "AR #18053 - XACTstep 6.0.1 - What operating systems (OS) are supported by the XACT step 6.0.1 software release?" [Online]. Available: http://www.xilinx.com/support/answers/18053.htm.

[41] "This is the log book for the Sun workstation started 11-JAN-1995." [Online]. Available: http://www.pa.msu.edu/hep/D0/ftp/run1/l1/inventory_logs/sun.lbk.

[42] R. Hashemian and T. Pearson, "Teaching Hardware Design with Online Laboratories," in *Internet Accessible Remote Laboratories*, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[43] P. Cotfas, D. Cotfas, D. Ursutiu, C. Samolia, and D. Iordache, "New Tools in Hardware and Software Design Applied for Remote Photovoltaic Laboratory," in *Internet Accessible Remote Laboratories*, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[44] C. Mwikirize, J. Nombo, A. Tumusiime, B. Maiseli, P. Musasizi, T. Sapula, S. Tickodri-Togboa, A. Mwambela, and A. Jiwaji, "Collaborative Development and Utilization of iLabs in East Africa," in *Internet Accessible Remote Laboratories*, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[45] P. Buschiazzo, M. Niederstätter, and A. Scapolla, "A Lab Server Model for the iLab Shared Architecture," in *Internet Accessible Remote Laboratories*, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[46] C. Maiti and A. Maiti, "Teaching Technology Computer Aided Design (TCAD) Online," in *Internet Accessible Remote Laboratories*, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[47] I. Gustavsson, U. Jayo, L. Claesson, L. Håkansson, K. Nilsson, J. Bartunek, J. Zackrisson, T. Lagö, J. Zubia, and I. Claesson, "The VISIR Open Lab Platform," in *Internet Accessible Remote Laboratories*, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[48] C. Dubey, H. Wong, V. Kapila, and P. Kumar, "Web-Enabled Remote Control Laboratory Using an Embedded Ethernet Microcontroller," in *Internet Accessible Remote Laboratories*, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[49] "Homepage of the Linux NIS/NIS+ Projects." [Online]. Available: http://www.linux-nis.org/.

[50] "Samba - opening Windows to a wider world." [Online]. Available: http://www.samba.org/.

[51] "Kerberos: The Network Authentication Protocol." [Online]. Available: http://web.mit.edu/kerberos/.

[52] "Heimdal." [Online]. Available: http://www.h5l.org/.

[53] "OpenLDAP, Main Page." [Online]. Available: http://www.openldap.org/.

[54] "ntp.org: Home of the Network Time Protocol." [Online]. Available: http://www.ntp.org/.

[55] “RFC 2743 - Generic Security Service Application Program Interface Version 2, Update 1.” [Online]. Available: http://tools.ietf.org/html/rfc2743.html.

[56] “SASL: Simple Authentication and Security Layer.” [Online]. Available: http://asg.web.cmu.edu/sasl/.

[57] “BIND | Internet Systems Consortium.” [Online]. Available: https://www.isc.org/software/bind.

[58] “DHCP | Internet Systems Consortium.” [Online]. Available: http://www.isc.org/software/dhcp.

[59] Intel Corporation, “Preboot Execution Environment (PXE) Specification.” 20-Sep-1999.

[60] “Linux NFS faq.” [Online]. Available: http://nfs.sourceforge.net/.

[61] “coreboot.” [Online]. Available: http://www.coreboot.org/Welcome_to_coreboot.

[62] “tftpd(8) - Linux man page.” [Online]. Available: http://linux.die.net/man/8/tftpd.

[63] “About Infiniband: IBTA.” [Online]. Available: http://www.infinibandta.org/content/pages.php?pg=about_us_infiniband.

[64] “IEEE 802.3 ETHERNET.” [Online]. Available: http://www.ieee802.org/3/.

[65] “MySQL :: The world’s most popular open source database.” [Online]. Available: http://www.mysql.com/.

[66] “pfSense Open Source Firewall Distribution - Home.” [Online]. Available: http://www.pfsense.org/.

[67] “Project: OCFS2 - oss.oracle.com.” [Online]. Available: https://oss.oracle.com/projects/ocfs2/.

[68] “The Z File System (ZFS).” [Online]. Available: http://www.freebsd.org/doc/handbook/filesystems-zfs.html.

[69] “Understanding the Remote Desktop Protocol (RDP).” [Online]. Available: http://support.microsoft.com/kb/186607.

[70] “FreeRDP.” [Online]. Available: http://www.freerdp.com/.

[71] “X Display Manager Control Protocol.” [Online]. Available: http://www.x.org/releases/X11R7.6/doc/libXdmcp/xdmcp.html.

[72] “VNC - Virtual Network Computing from AT&T Laboratories Cambridge,” 15-Aug-2000. [Online]. Available: http://classic-web.archive.org/web/20000815062533/http://www.uk.research.att.com/vnc/index.html.

[73] “freedesktop.org - Software/PulseAudio.” [Online]. Available: http://www.freedesktop.org/wiki/Software/PulseAudio.

[74] “KDE - Experience Freedom!” [Online]. Available: http://www.kde.org/.

[75] “GNOME.” [Online]. Available: http://www.gnome.org/.

[76] “LXDE.org | Lightweight X11 Desktop Environment.” [Online]. Available: http://lxde.org/.

[77] “Xfce Desktop Environment.” [Online]. Available: http://www.xfce.org/.

[78] “Cinnamon.” [Online]. Available: http://cinnamon.linuxmint.com/.

[79] “Trinity Desktop Environment.” [Online]. Available: http://www.trinitydesktop.org/.

[80] “The GTK+ Project.” [Online]. Available: http://www.gtk.org/.

[81] “Qt Project.” [Online]. Available: http://qt-project.org/.

[82] “Welcome to KDevelop.org | KDevelop.” [Online]. Available: http://www.kdevelop.org/.

[83] “Eclipse - The Eclipse Foundation open source community website.” [Online]. Available: http://www.eclipse.org/.

[84] “GCC, the GNU Compiler Collection.” [Online]. Available: http://gcc.gnu.org/.

[85] “gplEDA: All things GPL and EDA.” [Online]. Available: http://www.gpleda.org/.

[86] “Home of LibreCAD, 2D-CAD.” [Online]. Available: http://librecad.org/cms/home.html.

[87] “GIMP - The GNU Image Manipulation Program.” [Online]. Available: http://www.gimp.org/.

[88] “Git.” [Online]. Available: http://git-scm.com/.

[89] “Technical Specifications | VXI Bus.” [Online]. Available: http://www.vxibus.org/?q=node/206.

[90] S. Sharples, “VXI11 Ethernet Protocol for Linux.” [Online]. Available: http://optics.eee.nottingham.ac.uk/vxi11/.

[91] “Oracle VM VirtualBox.” [Online]. Available: https://www.virtualbox.org/.