ECE 833 - Final Report

Aaron Husz

May 10, 2013

1 Introduction

The independent study topic of choice described in this report is the implementation of a networked control system test bed which will be used for the testing of distributed control algorithms. Although the initial implementation of this test bed is designed for exchanging data from phasor data concentrators (PDCs) and phasor measurement units (PMUs), the test bed architecture may be used in other networked control system applications with similar testing requirements.

At the beginning of the semester, three objectives were established as part of a road map for the work to be conducted. Those objectives were:

- Create an IEEE C37.118 PMU data reader in C,
- Create a network topology on the Breakable Experimental Network (BEN), and
- Integrate the IEEE C37.118 data reader with the BEN topology and test it.

The objectives listed above were met this semester by using various software packages and tools freely available to the public so that the results may be shared with other entities without restrictive licensing agreements. Additionally, the creation of the network topology on BEN could not have been done without the support of colleagues at Renaissance Computing Institute (RENCI).

1.1 Background and Motivations

There is a growing interest within industry and academia as to how networked data systems interact with physical hardware, commonly referred to as cyberphysical systems. The reason for adding physical hardware to "cyberspace" is so that data can be remotely accessed for increased levels of system awareness. In order for data to be made available to system operators, the cyber-infrastructure which makes this possible may introduce security vulnerabilities. Additionally, it is expensive to create dedicated network infrastructure which connects dispersed physical assets like PMUs; a factor which compels firms to use commodity internet services to transfer data. Unfortunately it may be difficult to guarantee a data-rate or latency with commodity internet - a factor which may negatively impact controllers which rely on remotely collected data.

The earliest work on this project during the fall semester of 2012. Research was conducted during that semester to determine how to proceed with building the test-bed for distributed control systems. The test-bed would need a way to read data from PMUs using the IEEE C37.118 protocol and would have to support a variety of network topologies as required for different system models. It was determined early on that BEN would serve as the infrastructure that would allow us model various distributed systems. The acquisition and interpretation of IEEE C37.118 data presented its own challenged. There are several publicly available programs which allow users to collect and decode C37.118 messages, but almost all of them are exclusively available for the Microsoft Windows operating system. For the remaining applications which were compatible with Linux/Unix systems, software licensing agreements prohibited their use in our test-bed. For this reason, it was decided to build a proprietary C37.118 message reader for the Linux family of operating

systems. The advantage of this approach is that both the operating system and actual application software have no licensing restrictions.

Two iterations of the C37.118 message reader were written in fall, 2012. These message readers were written for MATLAB and SciLab. The advantage of using MATLAB or SciLab was the overall ease of use and potential for sharing the software with others. MATLAB is widely used by many researchers and is often less intimidating than software written in C/C++. Additionally, the processing toolboxes in MATLAB could allow for rapid algorithm development. However, the functions written in MATLAB and SciLab were not fast enough for real-time control systems, and so it was determined that a C37.118 reader would have to be written during spring 2013.

2 Software Tools Used

2.1 Wireshark

Wireshark is a freely available, multi-platform packet inspection tool. A user can use this tool to look at the bytes of data as they are transmitted and received. This tool was extremely useful in verifying the operation of the C37.118 reader as Wireshark is able to decode C37.118 messages (referred to as "SYNCHROPHASOR" messages in Wireshark) and display the contents of the message in a human-readable format. Although the source code of Wireshark is available for use, it was not used due to the licensing requirement that a program derived from Wireshark cannot be sold without including the new modified source code. Building the C37.118 reader from Wireshark also would have had unwanted and unnecessary features which would have reduced overall performance.

2.2 Flukes

Flukes is the GUI tool used for creating network topologies on BEN. The tool allows users to create topologies by establishing nodes which are connected by broadcast links (VLANs). Each node can be individually placed at any BEN point-of-presence (PoP) with any available virtual machine image. RENCI provides links to several pre-built virtual machine images on their website (`https://geni-orca.renci.org/trac/wiki/neuca-images`). Figure 1 below shows the network topology used for testing.

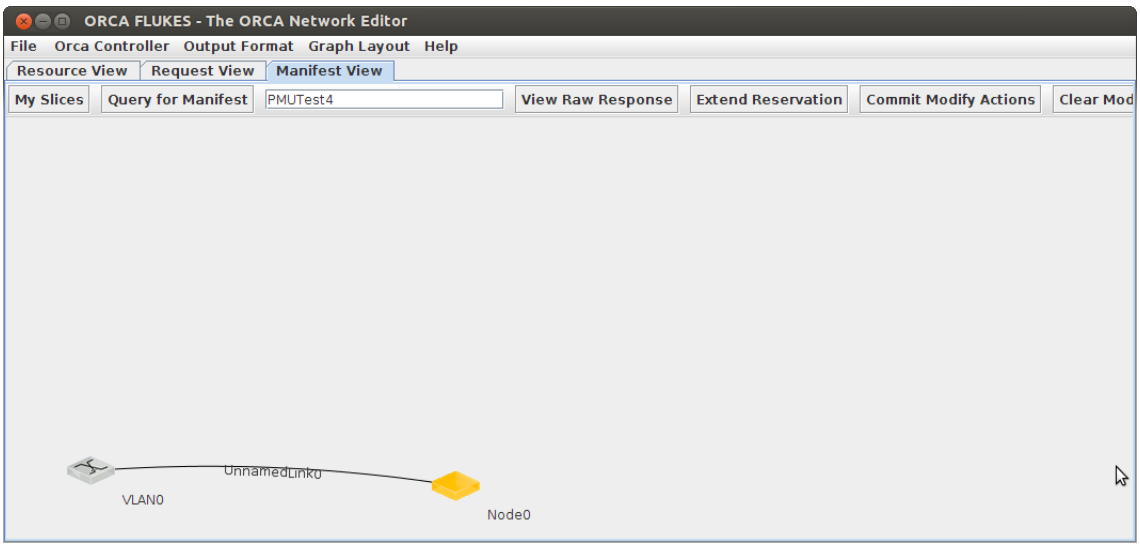

Figure 1: Screenshot of network topology created in Flukes.

2.3 OpenPDC

OpenPDC is an open source application which collects, displays, and logs PMU data. OpenPDC was useful during the development of the C37.118 reader so as to better understand how the C37.118 protocol works. This program was used to connect to one of the PMUs in the RTDS lab as Wireshark monitored the network activity. Unfortunately, OpenPDC was ruled out as a candidate for inclusion in the test-bed since it is available only on the Microsoft Windows operating system.

3 Software Written

3.1 IEEE C37.118 Message Reader

The IEEE C37.118 message format is one of the standard protocols used for transmitting PMU data over Ethernet. The protocol defines command messages used to communicate with a PMU as well as the structure of data messages which contain information like phase angle, bus voltages, and bus frequency. These messages can be sent through either a UDP or TCP/IP socket, however we are most interested in TCP/IP packet transmission since packet delivery is guaranteed.

The most important part of the test bed is the C37.118 message decoder. Without this piece of software, there is no way to process the PMU data coming from various parts of the network. The message reader had to be created from the ground up due since there was no commercially available C37.118 message reader available for the Linux platform that had no licensing restrictions. There are two versions of the C37.118 message reader that share a majority of their code. A command line version of the message reader allows a user to quickly determine whether or not he/she can connect to a PMU on the network. The C37.118 GUI was created from the C37.118 code, but modified to allow multi-threaded operation which simultaneously processes incoming PMU messages and updates a graph.

3.1.1 IEEE C37.118 Command Line Interface (CLI)

The CLI was the first implementation of the C37.118 message reader in C. The reason for using C was the potential for extremely fast code so as not to take away computational resources from future experiments. The program uses the Linux sockets library to connect to the PMU, a factor which prohibits this code from being used on a Windows machine. Upon startup, the program issues a series of commands to a PMU via the IEEE C37.118 protocol to start the data flowing. As new sample data is received, it is processed by the program. A flow chart presented in figure 2 shows the flow of program execution. What is remarkable about the command line interface is its speed. In one test case, a 158 byte message could be processed in approximately 20 microseconds, almost 1000 times less than the sample period of 16.66 milliseconds. The CLI may be started from the command line with the command:

```
./Client_socket <PMU IP Address> <Port>
```

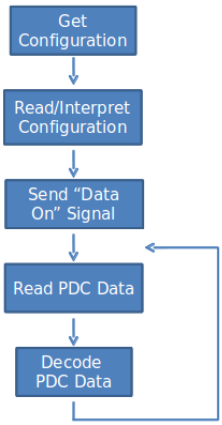

Figure 2: Flowchart of C37.118 reader program execution.

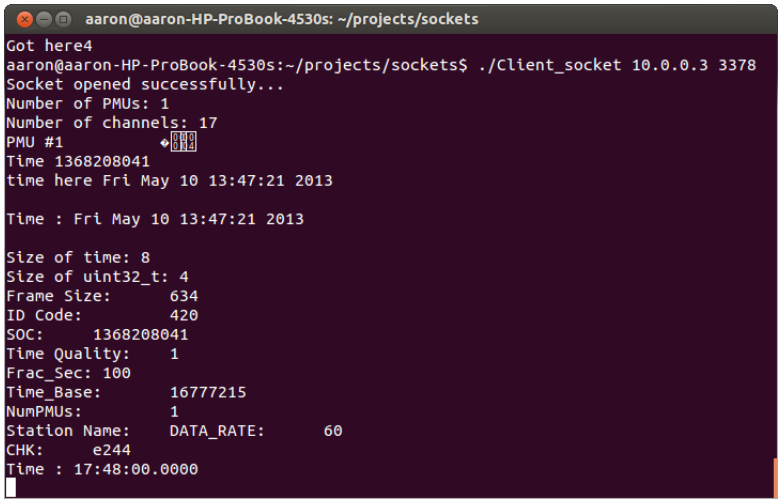

Figure 3: Screenshot of C37.118 Reader CLI.

3.1.2 IEEE C37.118 Graphical User Interface (GUI)

The C37.118 GUI was created as a visualization tool for the data received by the C37.118 interface. The GUI was written in C++ using the Qt framework and incorporates a modified version of the code used for the C37.118 CLI. The reason for modifying the C37.118 CLI was that it could not be easily put into a thread which was separate from the thread which updates the GUI display.

As seen in figure 4, a user can enter the IP address of the PMU and the port number on which the PMU is sending data. The program continuously adds bus voltage data to the chart. The GUI will eventually be updated with phase angle data, frequencies, and will allow the user to select which voltages/angles he/she would like to view. Unfortunately, the GUI requires far more resources than the CLI. The CLI used less than one percent of the total processor power whereas the GUI regularly uses more than 36 percent of the CPU. Despite this increase, the program is still able to update the display in real-time.

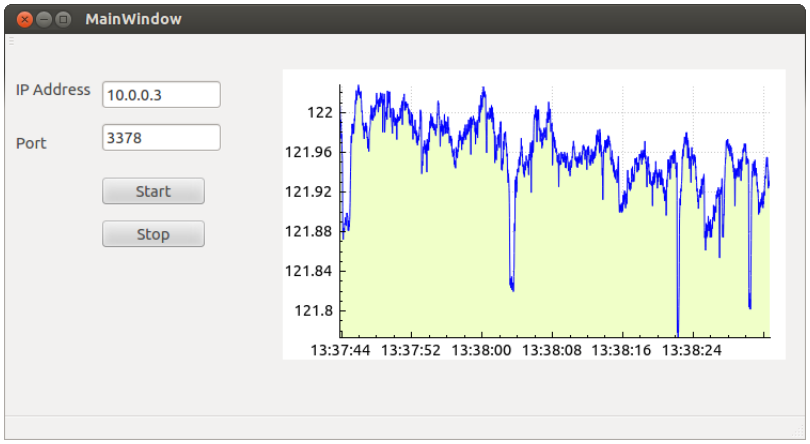

Figure 4: Screenshot of C37.118 Reader GUI.

3.2 IP Address Script

A Linux shell script (replace_ip_address) was written to streamline the process for configuring the network settings of the PMUs. The PMUs must know the IP address and port number of the device to which it

will be streaming data. The script is run on the computer which runs the C37.118 message reader. The script determines the IP address of the computer using the `ifconfig` command and then updates the PMU configuration through a telnet session. CKermit was used as the telnet client since it works in shell scripts unlike other telnet clients like Minicom. The complete code is listed in listing 1 .

Listing 1: replace_ip_addres script

```
#!/bin/bash
#Set_PMU_Client_IP
#default address
ARGS1="$1"
DEVICE="$2"
addr="152.14.125.38"
comp="me"

if [[ "$ARGS1" = "$comp" && "$DEVICE" = "wlan0" ]];
then
addr=`ifconfig $DEVICE | grep 255.255 | sed 's/^.*inet\ addr:*//' | sed 's/\ *
    Bcast.*//'`
echo `ifconfig $DEVICE | grep 255.255 | sed 's/^.*inet\ addr:*//' | sed 's/\ *
    Bcast.*//'`
elif [[ "$ARGS1" = "$comp" && "$DEVICE" = "cscotun0" ]];
then
addr=`ifconfig $DEVICE | grep 255.255 | sed 's/^.*inet\ addr:*//' | sed 's/\ *
    P.*//'`
echo `ifconfig $DEVICE | grep 255.255 | sed 's/^.*inet\ addr:*//' | sed 's/\ *
    P.*//'`
elif [[ "$ARGS1" = "$comp" && "$DEVICE" = "eth0" ]];
then
addr=`ifconfig $DEVICE | grep 255.255 | sed 's/^.*inet\ addr:*//' | sed 's/\ *
    Bcast.*//'`
echo `ifconfig $DEVICE | grep 255.255 | sed 's/^.*inet\ addr:*//' | sed 's/\ *
    Bcast.*//'`
fi

echo "Address is $addr"
sed "9s/.*/lineout $addr/" Kermit_set_IP > kermit.tmp
cp kermit.tmp Kermit_set_IP
rm kermit.tmp
kermit Kermit_set_IP
```

Listing 2: Kermit Input Script (Kermit_set_IP)

```
SET HOST 10.0.0.3
pause 1
lineout ACC
lineout OTTER
lineout 2AC
lineout TAIL
define \%d "SET P 5 PMOIPA1"
lineout \%d
lineout 10.0.0.2
lineout
lineout
lineout 152.14.125.236
lineout 4426
lineout Y
pause 1
SET EXIT WARNING OFF
exit
```

3.3 BEN Virtual Machine Image

During the process of creating a network topology in Flukes, a user must specify what operating system he would like to run on each virtual machine (VM). The operating system is stored in an image file on a web-server. The image can be thought of as a copy of the collection of files that would be on a desktop computer's hard drive. When an experimenter attempts to launch a VM, the image file is loaded onto the server at the BEN PoP For some topologies, the experimenter may require a several virtual machines, each with a different operating system. For this project, the same operating system was used for every virtual machine in the network.

The virtual machine images were created from a base image located at `http://geni-images.renci.org/images/standard/debian/deb6-neuca-v1.0.7.xml`. This image is a pared down Debian 6 image with the software tools required for BEN pre-installed. After downloading the image, many additional software packages had to be installed to support the C37.118 data reader and the IP address script. Those software packages include:

- ia32-libs
- build-essential
- libqt4-dev
- qt4-dev-tools
- ckermit
- libgtk2.0-0
- libpam-dev
- libssl-dev
- libc6
- libncurses5
- libpam0g
- libsocks4
- openssh-client
- nmap
- gnome-core
- xorg
- gdm3

It should be noted that ckermit is no longer in active development and in some cases, it may have to be compiled from its source code (`http://www.kermitproject.org/`).

With all of the preceeding software packages installed, the C37.118 reader source code was copied to the image. The source code must be compiled once after the VM is loaded since the source was not developed on the same hardware or operating system as the VM.

4 BEN - The Breakable Experimental Network

BEN is an ongoing cloud computing/software defined network project managed by The Renaissance Computing Institute (RENCI). BEN is a valuable resource for testing distributed algorithms as it can be configured to model a variety of network topologies which represent a distributed system. The RTDS lab is connected to BEN through a single ethernet port. At the present moment, connecting the RTDS lab to virtual machines on BEN is not an automatic process. First, a network topology must be created using Flukes. Once the network topology is active, a network administrator must create a network rule which translates traffic from the VLAN Flukes created to the static VLAN which services the RTDS lab (VLAN 904). Although this is not a technically challenging problem, it is a process that will need to be automated for future convenience. Figure 5 shows how the RTDS lab, BEN, and VMs were connected for the final system test. After a few weeks of testing and troubleshooting, this topology was realized and PMU data was successfully passed from the RTDS lab to a VM hosted on the RENCI ExoGENI rack in Chapel Hill, NC.

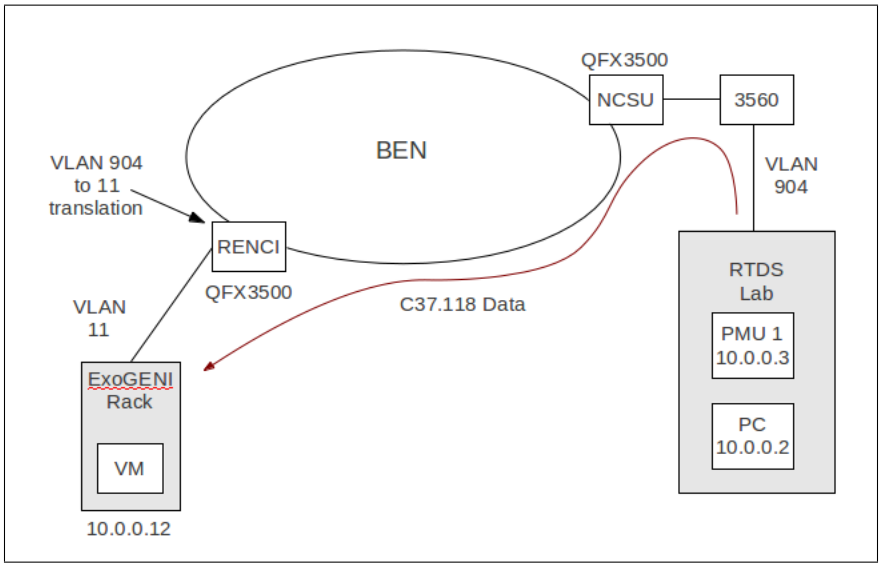

Figure 5: High level network architecture that shows how the RTDS lab, BEN, and VMs are connected.

5 Challenges

The greatest challenge of the semester was establishing a network connection between the RTDS lab in the FREEDM Center and BEN. The first attempts of connecting PMUs to BEN were unsuccessful in that the virtual machines launched on the RENCI ExoGENI server could not communicate with the PMUs. There were three reasons for this:

- Network rules which connect the virtual machine to the RTDS lab had to be manually entered into the BEN system,
- The VLAN number which connects the RTDS lab to BEN was assumed to be 900 when it was actually 904, and
- Physical connections between the servers and network switches at RENCI in Chapel Hill, NC, were not connected as network administrators thought.

The process for troubleshooting the network problems were complicated when in one rare but specific case, all of the network parameters happened to be set correctly, which led the network administrators to believe that all other cases should work as well - a case where the system was inconsistently inconsistent.

One confusing element in working with BEN was the many different options for provisioning network resources. Within the GENI portal there is a tool which allows an experimenter to create a network slice, similar to Flukes. However, any slice that a user creates through the GENI portal cannot be shown in Flukes and vice-versa. In addition to Flukes and the GENI portal there is also a JAVA applet, Flack, which allows users to connect BEN resources to other GENI resources across the country. For the foreseeable future, Flukes should be used for all experiments connecting the RTDS lab to BEN.

6 Future Work

The next phases of this project will expand the capabilities of the test-bed and reduce the complexities of running experiments on it. A linear algebra library will be added to the C37.118 data reader for determining system modes. Several open source libraries exist for doing such calculations including Trilinos, Armadillo, and PETSc.

The C37.118 reader GUI must be optimized for speed and memory use. The original command line interface which read C37.118 messages used less than one percent of total system processing power whereas the GUI typically utilized 40 percent or more of the CPU. This level of utilization will significantly limit the number of mathematical computations that can be done on the data if it is not addressed.

The C37.118 reader GUI also needs several features added. Currently, only one phasor voltage is displayed in real-time, and that phasor is hard coded. A better version of the GUI would allow the user to select any phasor and display it in real-time, much like the OpenPDC software. The GUI should also allow the user to view other parameters like frequency, phase angle, and even network statistics like packet size and latency.

What is really unique about this test-bed is that it is on BEN - a software defined network with cloud computing resources. Network paths can be defined on the fly which changes the characteristics of the network. If data which is being fed into a control algorithm is suddenly delayed, or becomes accessible sooner, how should the control system adapt to these changes? Are there cases in which a controller may choose to change the rate at which data is sent to it to optimally use the network resources it has? These questions will be examined in more detail in the coming months.

7 Conclusions

The three objectives for the semester were met. An IEEE C37.118 protocol reader was created in C. This reader was found to be much faster than the MATLAB and SciLab implementations tested in Fall, 2012. The RTDS lab at FREEDM was successfully connected to BEN after a long period of troubleshooting. Finally, the capstone test of passing PMU data from the RTDS lab to a VM on BEN was successfully completed, demonstrating the integration of the C37.118 reader with BEN.