Boston University
Electrical & Computer Engineering

EC464 Capstone Senior Design Project

Final Report

CyberTap

by

Team 2
CyberTap

Team Members:
Felipe Dale Figeman fdale@bu.edu
Alex Fatyga afatyga@bu.ed
Evan Lang evanlang@bu.edu
Noah Malhi malhin@bu.edu
Justin Morgan justinfm@bu.edu

Submitted: April 10th, 2020

Table of Contents

Executive Summary

CyberTap
Group 2 – CyberTap

CyberTap is a high throughput hardware network tap designed for industrial control systems closed off from the internet. Other network monitoring solutions are software based with overutilized CPUs which could theoretically result in packet loss. Packet loss is particularly undesirable in the context of the high data throughput involved in industrial control systems, especially as said systems directly rely on input data for operation. CyberTap will be able to collect Operational Technology (OT) network packets, parse and generate metadata for all relevant network protocols of a system, and store said metadata in persistent storage. The product will be implemented on a Field Programmable Gate Array (FPGA) to utilize their ability to quickly process large data loads. The final deliverable will contain the following: An emulated high-data-volume OT network using 2 Raspberry Pis and a network switch, an FPGA that sniffs, parses, and outputs the packet data coming from the SPAN port of the switch, and a web application for querying the CSV and displaying the metadata.

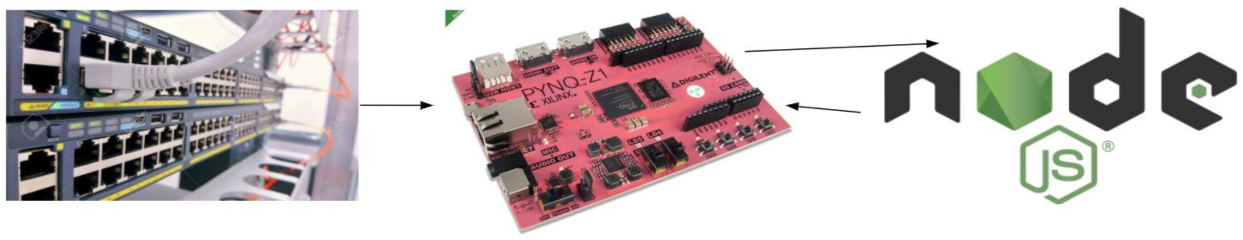
1.0 Introduction

CyberTap's client, Cybereason, is a cybersecurity company focused on enterprise endpoint protection. Cybereason has no way of gaining visibility into or performing long term data collection of their customers' network traffic. This is where CyberTap comes in. They need CyberTap to be able to generate and store useful metadata for them to analyze with their cybersecurity tools. Therefore, the purpose of CyberTap is to losslessly capture network packets and generate queryable metadata on them, persistently storing it for later analysis.

The device will be utilized by the client primarily for Operational Technology (OT) networks and Industrial Control Systems (ICS). OT networks support infrastructures for utilities, manufacturing, and defense. They utilize both hardware and software to detect changes via monitoring of physical devices, such as sensors. An ICS is a type of control system used for operation and automation of industrial processes.

Cyberattacks are becoming more and more prevalent than ever in an increasingly technologically connected world. Especially in the context of an ICS environment, CyberTap provides a clear and lossless record of the network traffic and fills in potential information gaps that may hinder a necessary response to malicious attacks. An ICS environment that involves complex and powerful machinery could be providing important infrastructural functions to society at large; any cyberattacks must be flagged and immediately dealt with to prevent system shutdowns and infiltrations, and CyberTap is the solution to filling any potential gaps in security analysis.

Figure 1: General Design



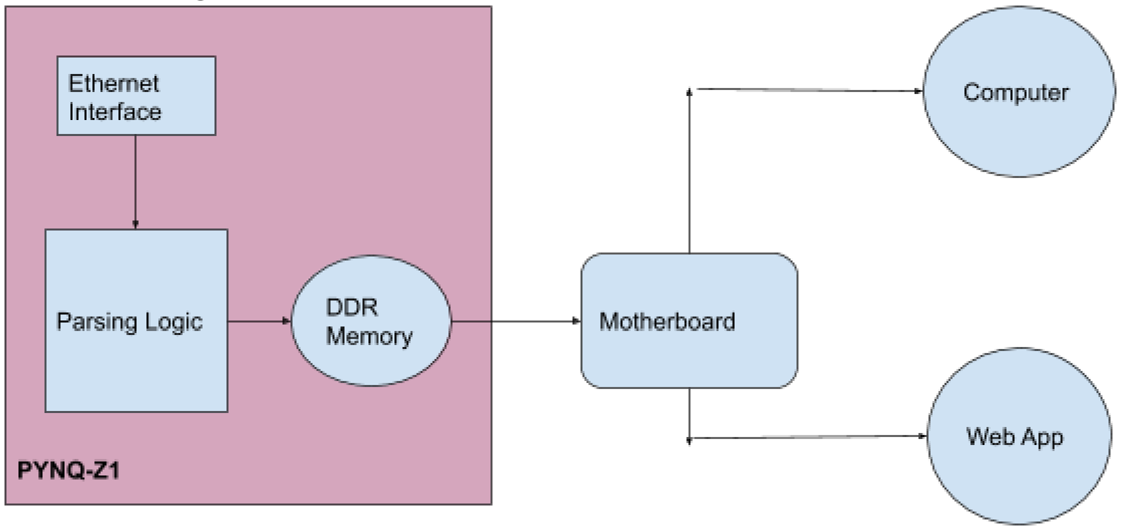
A general overview of the current CyberTap system, going from the network switch through the FPGA to the web application.

The general approach for the team was to utilize an FPGA, implement a packet sniffer to obtain the network activity flowing through a network switch, create a programmable logic packet parser to parse the incoming packets at high speeds, and output the generated metadata to a presentable user-friendly web app. By doing this in hardware, the typical throughput and sniffing capabilities from a software implementation can be extended. The FPGA is a blank slate of programmable logic which allows for it to be used specifically for the tasks of packet sniffing and parsing. Since there is full control over the operation of the FPGA, there are no other tasks that could interrupt the system, unlike what occurs with software on a CPU. Another key aspect of the FPGA is the sheer number of memory logic cells. This allows for the device to be scalable by enabling more protocol parsing modules to be added. Additionally, the FPGA module can be “hot swapped” into different network environments, containing the parsing modules needed for each specific network.

By using this approach, the needs of the client can be adequately met. Their first request was for a hardware device with the ability to sniff a network without high loss of data. By using an FPGA, it will allow for a lossless system that should be able to run under high stress networks. Additionally, Cybereason will later be able to adapt the device to their various needs as CyberTap will be easily scalable. The basic parsing and generation is completed in the FPGA and different protocols can be added to the logic as per need basis. Lastly, by writing to a CSV file that is saved to persistent storage, the data can be easily parsed, queried, and viewed. This satisfies their requirement for a long-term solution. Cybereason needs this long-term solution so they may perform long-term data analysis.

2.0 System Description

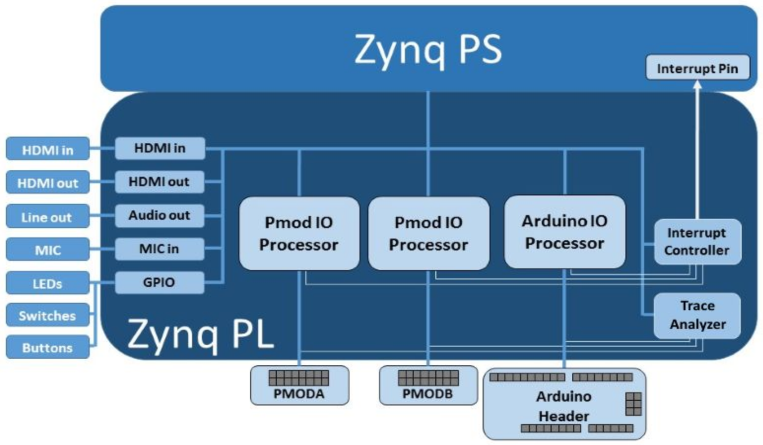
Figure 2: Block Diagram.



Hardware Diagram for the Network Tap. The flow goes from the PYNQ to an external computer. The PYNQ receives data through the ethernet interface, parses it, and stores it in a CSV file. On a computer, data is read from the PYNQ, the data is stored and also used for the hosted web app.

The hardware flow of the device first begins with an ethernet connection. The ethernet interface connects the FPGA to a network switch's pre-configured SPAN port. Through port mirroring, all of the network packets going through the switch are duplicated and sent to the FPGA. Due to the halting of group work, the final aspects of the project including the entirety of hardware parsing were not implemented, and so the parsing is performed on the PYNQ's 650MHz Dual core Cortex-A9 CPU.

Figure 3. PYNQ Overview



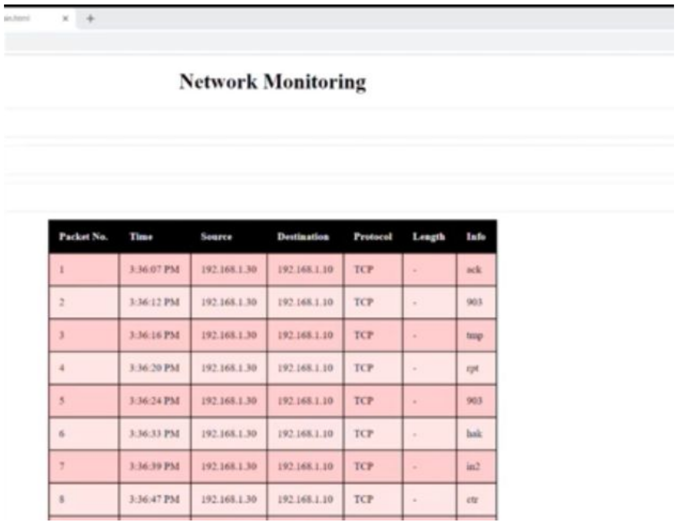
A block diagram describing the PYNQ hardware overlay. Each hardware element the PYNQ is simulating in hardware is represented and connected via the overlay in software

Packet sniffing was implemented by a raw socket sniffer. In networking terms, communication between devices occurs at the transport level and sockets are opened on each device to allow for a flow of data. The device opens a raw socket and is able to read all data that goes through it. By doing this at such a low level, it allows for higher speeds and much lower packet loss as it does not have the bulkiness of a library like in previous attempts. Utilizing a raw socket with simplistic software parsing allowed the device to have 0% packet loss. From this, the device could sniff all of the major protocols.

For the parsing of the sniffed data, early progress on using a parsing library was replaced with lower level bit manipulation. The format of the sniffed data was received in hexadecimal so the device converts hex to ascii. The payloads of the packets were kept in hexadecimal while the other key components such as type, size, and time were kept. From the conversion, different types of protocols were able to be determined and sorted.

The packet is passed into the board's on-chip DDR3 from the ethernet controller, and used to generate metadata including source IP, destination IP , network protocol, packet size in bytes, and the time it was received. This is done by extracting those elements from their corresponding fields in the packet object. The packet and metadata is then parsed to a CSV formatted row string and read from the computer.

Figure 4. Web Application.



Network Monitoring

Packet No.	Time	Source	Destination	Protocol	Length	Info
1	3:36:07 PM	192.168.1.30	192.168.1.10	TCP	-	ack
2	3:36:12 PM	192.168.1.30	192.168.1.10	TCP	-	903
3	3:36:16 PM	192.168.1.30	192.168.1.10	TCP	-	tmp
4	3:36:20 PM	192.168.1.30	192.168.1.10	TCP	-	rpt
5	3:36:24 PM	192.168.1.30	192.168.1.10	TCP	-	903
6	3:36:33 PM	192.168.1.30	192.168.1.10	TCP	-	hak
7	3:36:39 PM	192.168.1.30	192.168.1.10	TCP	-	in2
8	3:36:47 PM	192.168.1.30	192.168.1.10	TCP	-	ctr

A general view of the front end and how packets are displayed.

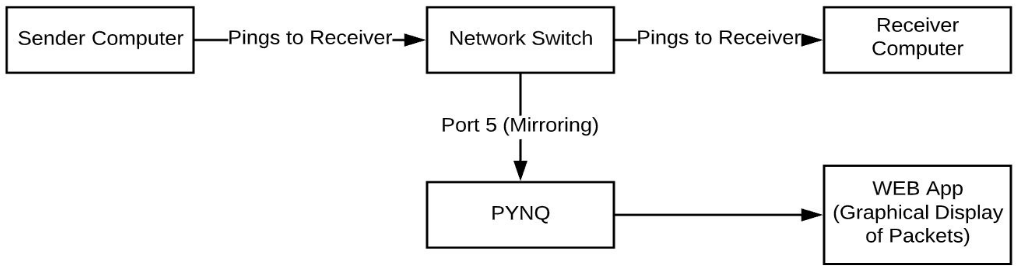
For viewing data, the web applications provide a real time look into the network activity. The web application reads from a CSV file and allows users to search through their packets in near real-time. This allows for a soft real-time analysis of the network data stream. The packet metadata is displayed on a table. Each packet and its corresponding metadata are a row. As each metadata field corresponds to a column on this row, it also enables finding of the packets through any known parameter such as time sent or protocol. Therefore, users may search by source, destination, and protocols, The web application is also hosted on localhost of the computer and not on the FPGA so the FPGA does not get slowed down.

3.0 Second Semester Progress

The first focus of this semester was setting up the new PYNQ-Z1 FPGA. Originally, the group was provided with 2 Nexys A7 FPGAs and a network switch by the University. It was decided a new FPGA board would need to be purchased that satisfied higher data requirements. After thorough consideration, the group decided on the PYNQ-Z1 because of its functionality and pricing. With its 1Gb/s ethernet port as well as HDMI in and out, the PYNQ was believed to offer significantly higher transfer speeds than the A7 while also being cheaper. Once the new PYNQ board arrived in late January, the team immediately began the setup process, creating a boot drive of the boards operating system on an SD card. Once it was confirmed the board was booting the base image properly, work immediately shifted to adding the features lost once the boards were switched.

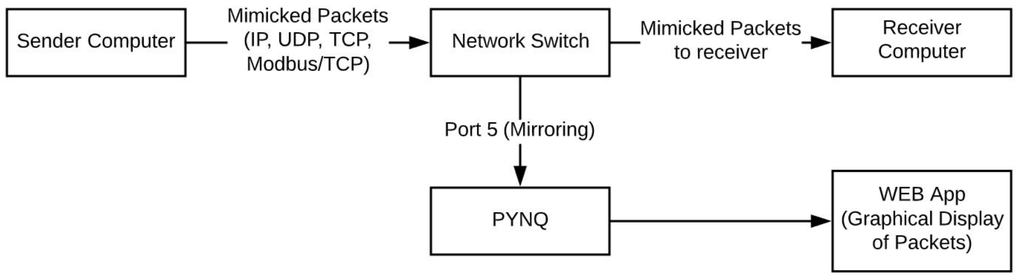
The next focus of the semester was on increasing the quality of packet sniffing to match the requirements of the client Cybereason. As the packet sniffer effectively works as the packet intake of the entire CyberTap system, the packet sniffer itself must not contribute to any packet loss in the system. The client desired less than 1% packet loss through CyberTap. Additionally, a functioning packet parsing system was required to actually guarantee that the CyberTap was not losing or incorrectly altering the packets. Scapy was then used for both packet sniffing and parsing. In order to gauge the capabilities of the board and Scapy, a comprehensive test format was created.

Figure 5: Stress Testing via Ping.



This figure shows the first test performed. The computer pings with ICMP packets at various speeds and the packet information is displayed in the web application.

Figure 6: Modbus (and other protocols) Packet Testing.


Sender Computer
Mimicked Packets (IP, UDP, TCP, Modbus/TCP)
Network Switch
Mimicked Packets to receiver
Receiver Computer
Port 5 (Mirroring)
PYNQ
WEB App (Graphical Display of Packets)

This figure shows the second test performed. The computer is sending mimicked packets of various protocols and these packets are parsed and then displayed in the web application.

Two tests were done to prove the functionality of the system. The first test started after setting up the FPGA and running the bash script. The web app hosted at 192.168.137.99:8080/main.html was opened in order to view the parsed information of any incoming or sniffed packets. Two computers were connected through the same switch with the FPGA and given static IP's 192.168.137.XXX/24. One computer was used to ping the other using the Linux ping command. The ICMP packets sent from the first computer to the second were sniffed by the FPGA, parsed, and displayed on the web app. During this process, the web app was refreshed multiple times to see incoming packets and their related information. This process was completed 3 times with 3 different ping intervals (.1s, .01s, and .001s) with 1000 pings sent per run. After a ping run finished, the packet sniffer was immediately turned off to prevent unrelated packets from being included in the test, as the attached devices had background processes that would attempt to use the network. If allowed, these packets would be sniffed and added to the CSV, tainting our results. Three metrics were required for the test to be considered successful. First, the packet loss on the .1s interval ping test needed to be less than 1%, the packet loss on the .01s interval test must be less than 10%, and the packet loss on the .001s interval test must be less than 20%. Second, the web app needed to properly display the list of sniffed packets. Third, the information (source, destination, time sent, etc.) included in each packet sniffed had to be accurate.

The second test consisted of the creation of packets of various protocols (IP, TCP, UDP, Modbus/TCP) using the Scapy console on a sender computer. This involved simply pulling up the Scapy console in the sender computer's terminal, creating packets of the specified protocols using native Scapy commands (or in the case of Modbus, a separate library specifically for creating Modbus packets), setting the source and destination IPs, and sending the packets in a loop. Through the console, these packets were sent to the receiver computer. The web app was used to confirm that the newly created and sent packets were being picked up by the FPGA and correctly processed. In order to ensure that the packets sniffed and processed by the FPGA matched those sent out via the Scapy console, the fields for sender IP, destination IP, and protocol type were matched against the sender's configurations used to create and send the packets.

The initial Scapy iterations of packet sniffing were displaying more than 40% packet loss during stress testing. This was determined using the Linux ping command to send large amounts of packets to the CyberTap and using our software parsing and storage to determine how many were captured and how many were lost. Based on the first part of testing during the 2nd prototype test day, the modified Scapy version of packet sniffing at worst only had 2.8% packet loss. This exceeded expectations from the previous testing iterations that indicated up to 20% packet loss during high interval pinging. This can be attributed to the group's attempts to cut down on unnecessary or unrelated use of the board's processing power during the packet sniffing tests. However, after testing, the group tested another method of packet sniffing using a raw socket which was able to handle sub millisecond ping intervals, as opposed to the previous method of sniffing which could not. Since the previous method required Scapy, which is a large python library, there is code bloat which can result in slower processing of incoming traffic. By viewing network traffic Gateway-to-Gateway from a transport layer socket, raw data can be manipulated freely. Using excessive libraries adds a latency to the design which inhibits the goal of less than 1% packet loss. Ultimately, the raw socket iteration of packet sniffing was chosen to replace modified Scapy.

Additionally, the web application's method of reading the results of board sniffing and parsing was greatly improved upon. Instead of reading results through the UART, the PYNQ board would create and constantly update a CSV file storing the generated packet metadata. The web application would also constantly read from this file, and add to the table accordingly. This prevented packet information from being lost between the FPGA and web application, like in the previous iteration on the Nexys A7. At first, the group hosted the web application on the PYNQ which slightly slowed downed packet sniffing and parsing. This method was used in the lab testing that showed, at most, 2.8% of packet loss and therefore, did not significantly impact packet loss. However, the group was able to host the web application off board by being able to read from the FPGA's network drive into the javascript backend on the computer and feed into the front end. The group then updated the front end to have a slightly better appearance and be more readable by making the table larger and rows skinnier on the web page.

After discussing with Professor Pisano at the 2nd prototype testing day the rest of the work for this team lead period was decided. The project lacked flair and visual impact, properties needed to be successful at ECE day. The current iteration of the project had a very basic UI, only listing packets received and their parsed information. Therefore, it was decided that the time before final testing would be devoted to increasing the visual appeal and personal interaction of the project. It was also agreed that hardware-based packet parsing would be worked on in parallel with the project's visuals. The team was divided into 2 main groups, one focusing on improving the project for ECE day and the other on hardware parsing.

4.0 Technical Plan

After the work completed in the second semester, the group has proposed the following plan to complete the project:

Task 1. Prepare for Long Term Use.
After parsing a packet, the packet is written into a CSV file that the web application is able to read from. Currently, this file is only written to and only emptied on start up. Consequently, writing to the CSV will have to be modified. The CSV file will need to also be used for the data storage. The following milestones will complete the necessary steps to have a system prepared for long term use:

- Milestone 1: Test to decide the period of time it takes for the CSV file to become half full.
- Milestone 2: Modify writing to CSV to empty file after this period of time.

Task 2. Web Application Improvements.
The web application can be improved in various ways to allow the user a better viewing experience. These simple modifications will greatly increase the usability of the application.The first improvement would be to introduce auto refreshing into the web app. This would be done in the front end and is a relatively easy addition. The second improvement would be to have new packets show in the top of the table rather than at the bottom. This means that newer packets are at the top and older packets are at the bottom. This addition would also be relatively easy and would be done by modifying the front end. The following milestones will complete the necessary web application improvements:

- Milestone 1: Auto refresh each second.
- Milestone 2: Modify front end to add new packets to the top of the table.

Task 3. Hardware Packet Parsing.
Although software packet parsing was completed and had great performance, hardware packet parsing would be even greater. In terms of hardware parsing, Xilinx SDnet software was being procured from Boston University. This software would have enabled easy production of the parsing modules. These modules would then have been integrated into the python overlay of the PYNQ board and used as a replacement for the current software parsing system. This would significantly increase the maximum bandwidth the Cybertap would be able to handle. The following milestones will allow for the creation of a hardware parsing system:

- Milestone 1: Create a hardware parsing module for one packet type in Xilinx SDnet.
- Milestone 2: Integrate the parsing module into the PYNQ-Z1’s python overlay system.
- Milestone 3: Create multiple parsing modules and a system for switching between the different modules for different packet types in the PYNQ-Z1.

Task 4. Simplifying Setup and Startup.
The current setup of the system would include plugging in the FPGA and downloading NodeJS, multiple other packages and the web application code from the group's GitHub repo. Simple steps would greatly simplify this process and allow for a better ease of use. A batch file would be created that would set up the web application and install NodeJS as well as many npm packages. Another batch file would be created for web application startup and would include running the main javascript file. Then, the GitHub must be updated with a zip file of all necessary files and instructions in the readme for the user to follow. The following milestones will allow for a more simplified setup and startup:

- Milestone 1: Complete a batch file for setup.
- Milestone 2: Complete a batch file for startup.
- Milestone 3: Add instructions to GitHub.

5.0 Cost Breakdown

Item	Description	Cost	Quantity
1	Raspberry Pi 3	$38	2
2	PYNQ-Z1 FPGA	$199.99	1
3	tp-link Pro Network Switch	$30	1
4	Nexys A7 FPGA*	$250	2
5	tp-link Desktop Switch*	$15.99	1
6	Ethernet Cables*	$4	3

* denotes an item donated by Boston University College of Engineering

The group was provided with 2 Nexys A7 FPGAs and a network switch by the University. These boards served as the original foundation of CyberTap's system. However, lab testing results showed greater than 5% packet loss, which was considered unacceptable. It was then decided a new FPGA board would need to be purchased that satisfied performance requirements. The FPGA is easily the most expensive component of the system so picking the right FPGA was vital to the group's success and budget. After thorough consideration, the group decided on the PYNQ-Z1 because of its functionality and pricing. This pricing fits extremely well within the budget without compromising the functionality. It also includes an advanced tool suite that streamlines work on the HDL module, with its Python HDL suite. The device's interface was superior to the Nexys A7 as it had 1 GB/s ethernet and various USB ports. Compared to other devices with the same ethernet port speeds, this device was drastically cheaper.

CyberTap is designed to be plugged into the SPAN port of a network switch within an ICS environment in order to "tap" the network by backing up all incoming/outgoing packets through the switch. In order to configure a SPAN port on a network switch, the switch must be managed and be capable of port mirroring, which allows the user to copy traffic going through the switch ports and send it through a specified mirroring port. The switch donated by the College of Engineering is an unmanaged switch, so the group purchased the mirroring-enabled Pro variety of the original switch.

For testing, the team needed to develop a mock ICS environment. To do this Raspberry Pi 3's were utilized. The team bought two Pis to communicate over the switch. The Pi's sent different protocols through the switch at varying speeds which allowed for an efficient way to test our devices progress.

6.0 Requirements

The following requirements were decided upon in December:

Requirement	**Value, range, tolerance, units**
Packet Loss through FPGA	0%
Packet Loss To SSD	0%
Packet Loss to Web App	0%
High Throughput/Stress Test	Network flow up to 1 Gigabit ethernet
Soft Real Time Web App	< 5 sec latency of memory packet transfer to display
Long-term Use	Continuous network flow >5 days
Ability to sniff common OT Protocols	Modbus, DNP3, TCP, IP, ARP
Hot Pluggable	Able to plug into any span port and begin data processing and storage

After various discussions, the group decided to modify a few requirements. Packet loss was no longer required to be 0% but instead, the requirement has been changed to be less than 1% after talking with Professor Pisano regarding the importance of the project. Additionally, writing to SSD was decided to no longer be necessary for what the group wanted to accomplish in the remaining timeline. Therefore, packet loss to SSD is no longer a requirement.

The following are the results from the lab testing completed:

Test 1: 0.1s Interval Ping

Number of Packets Sent	Number of Packets Received
1000	995
Packet Loss: 0.5%	

Test 2: 0.01s Interval Ping

Number of Packets Sent	Number of Packets Received
1000	972

Packet Loss: 2.8%

Test 3: 0.001s Interval Ping

Number of Packets Sent	Number of Packets Received
1000	991
Packet Loss: 0.9%	

Web App:
Are the packets and their information correctly displayed? Yes.

Test 2: Modbus
Does the Web app correctly label the packets as TCP packets? Yes.

These results prove the following requirements: <1% Packet Loss to Web App, High Throughput/Stress Test, Ability to sniff common OT Protocols, and Soft Real Time Web App.

During the latest lab testing, the group proved between 0% and 2.8% packet loss to the FPGA with pings. After receiving these pings, their packet information was added to the web application. The web application was then able to prove soft real time updates by manually refreshing. The web application showed no packet loss from FPGA to web app during our testing which meant the group was able to prove this requirement was fulfilled. Furthermore, lab testing included a stress test to prove that the system could handle network flow up to 1 Gigabit ethernet. After testing, the group worked out a new system to also have 0% packet loss through the FPGA using raw sockets as the new method of packet sniffing.

The group was not able to add in some finishing details and minor additions to complete the system. The long-term use requirement has yet to be fulfilled but would require modifying the process of writing to the CSV file to wipe clean the file after some period of time to stop it from reaching its maximum size. The hot pluggable requirement is close to being completed; however, the setup and startup process of the web application needs to be greatly simplified. Additionally, the web application needs to be improved upon to allow for a more seamless use.