Boston University
Electrical & Computer Engineering

EC464 Capstone Senior Design Project

Second Prototype Testing Report

CyberTap

by

Team 2
CyberTap

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

Required Materials:

Hardware:

- 1 Ethernet capable laptop
- TP-LINK TL-SG105E 5-Port Gigabit Easy Smart Network Switch
- Desktop PC
- PYNQ-Z1 Board
- 3 Ethernet Cables
- 1 USB-Micro USB cable

Software:

- Xilinx C/C++ SDK 2019.1
- Jupyter Notebook
- Node.js Web Client
    - main.html
        - Front end that receives data from server.js and puts it into the table
    - server.js
        - Back end that reads the csv file, sends it to front end
- Packets.csv - as packets are parsed, they are written into this csv file which is read in server.js
- Python
    - sniffEx.py - sniffs network activity, parses it and calls a csv formatting library to store the information in a csv file
    - Main.py - runs all python files
    - Scapy terminal - used to create and send packets of various protocol types
- Bash script main.sh - runs the web client in the background and then runs python sniffing and parsing

Setup:

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 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. After this test, the second test was performed. 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. 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.

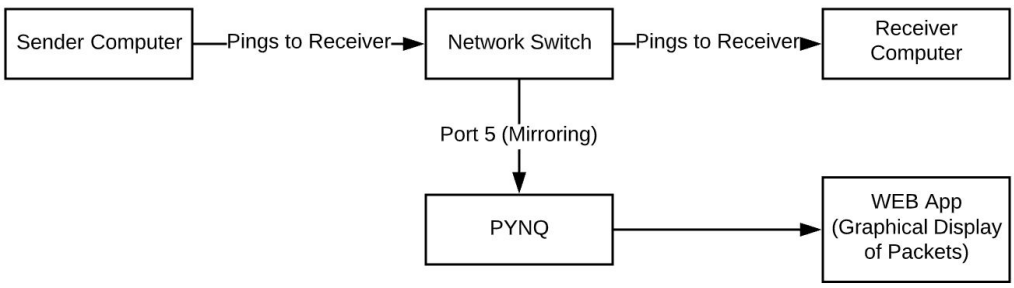

Figure 1: Stress Testing via Ping. This figure shows the first test performed.

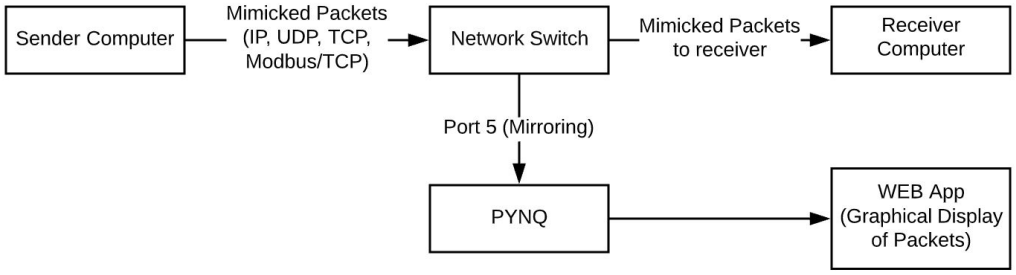

Figure 2: Modbus (and other protocols) Packet Testing. The figure of the second test.

Measurements:

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 measurements were taken from the terminal and web application.

Conclusions:

Based on the first part of testing, the current iteration 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 attempts to cut down on unnecessary or unrelated use of the boards 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 current method requires 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. As a result, this new method will be adapted into the project. The second test successfully illustrated the FPGA's ability to handle different types of packets. The FPGA correctly sniffed, parsed, and stored UDP, TCP/modbus, ICMP and raw IP packet metadata. The web app correctly displayed the correct source, destination, packet type, and time of the produced packets when compared to the scapy sender computer. While the Modbus packets were not explicitly labeled, as they are sent over TCP, all other packet types were correctly parsed and displayed. In order to remedy this obfuscation of lower layers, the sniffer will be modified to include the display of protocol layers below TCP, UDP, etc. in order to discern between different ICS environment packet protocols. The next step for packet parsing will focus on the addition of new ICS environment packets. This will more accurately represent

the environment the CyberTap will be used in. Additionally, the creation of a hardware parsing module will allow faster parsing when compared to the software form currently being used.

After speaking with Professor Pisano, the group has also recognized that CyberTap lacks the visual hook or interaction vital for success on ECE Day. Based on stress testing, the current method of receiving packet metadata into the web application is not as robust as it could be. Currently, one must refresh the page to get new data into the web application. Therefore, creating a more efficient method will be one of the priorities. One possible method considered is inserting packet information directly into a database and having the web application access the database. Another method being considered is creating the csv file as a memory mapped file. The goal of the team is to have live-data updates on the website similar to the software application Wireshark. This would allow a user to see live incoming traffic and be able to pause to get a better view of the data. The group has also decided that in order to increase the quality of the demo experience it must be made interactive. For example, buttons could be set up that when they are pressed, it causes the sender computer to send a specific packet. This would allow a spectator to see the real-time effectiveness of the packet sniffer and parser. Another potential interactive method would involve developing a dial to change the rate at which packets are sent. A user would turn the dial counter-clockwise to make the rate of packets go down and turn the dial clockwise to make the packets send rate go up. This would illustrate the ability of the Cybertap to handle high throughput loads. Other than interactiveness, making the web application as readable and user-friendly as possible is essential to CyberTap's presentation. By exhibiting the full project in a visually appealing and clear manner, onlookers will get a better

sense for the product's goals and uses without the prior computer networking and hardware knowledge needed to fully grasp it.