Summer 8-2022

A Novel Testbed for Evaluation of Operational Technology Communications Protocols and Their On-Device Implementations

Matthew Boeding
mboeding@huskers.unl.edu

A NOVEL TESTBED FOR EVALUATION OF OPERATIONAL TECHNOLOGY COMMUNICATIONS PROTOCOLS AND THEIR ON-DEVICE IMPLEMENTATIONS

by

Matthew Boeding

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Telecommunications Engineering

Under the Supervision of Professor Hamid R. Sharif-Kashani

Lincoln, Nebraska

August, 2022

A NOVEL TESTBED FOR EVALUATION OF OPERATIONAL TECHNOLOGY COMMUNICATIONS PROTOCOLS AND THEIR ON-DEVICE IMPLEMENTATIONS

Matthew Boeding, M.S.

University of Nebraska, 2022

Advisor: Hamid R. Sharif-Kashani

Operational Technology (OT) and Infrastructure Technology (IT) systems are converging with the rapid addition of centralized remote management in OT systems. Previously air-gapped systems are now interconnected through the internet with application-specific protocols. This has led to systems that had limited access points being remotely accessible. In different OT sectors, such as manufacturing, building automation, and the energy sector, legacy protocols previously transmitted over serial communication were updated to utilize TCP/IP connections for communicating with legacy devices. New protocols such as IEC-61850 were also introduced for monitoring of different Distributed Energy Resources (DER). The IEC-61850 standard's Generic Object Oriented Substation Event (GOOSE) protocol outlines the representation and communication of a variety of different components through Publisher and Subscriber roles. Each publisher and subscriber are defined specifically on Intelligent Electronic Devices (IEDs), which may differ in manufacturer and capabilities. Therefore, each defined publisher and subscriber are network specific, so the different topologies and data types sent can vary between networks. To support the different objects represented in the protocol, customizable configurations for GOOSE supporting components is required.

In this thesis, an effective, flexible, and practical testbed is introduced for evaluating OT protocols, with a case study in the implementation of the GOOSE protocol on IEDs. Common cyberattacks on the GOOSE protocol are identified and implemented on the testbed with variable data rate generation. The display the versatility of the test bed's capabilities, attacks implemented include a Denial of Service (DoS), replay, and False Data

Injection Attack (FDIA). The tests are then executed on three separate GOOSE devices, two devices from reputable manufacturers, and a Raspberry Pi running off an open source library, libiec61850. Each device is configured with a separate IED Capability Description in accordance with manufacturer instruction to ensure the test operated under valid operating conditions.

The results of the tests show that, while complying with the protocol standard, each device has different processing methods resulting in different strengths and weaknesses in implementation. This thesis examines each device's response to different attacks and their implications in real world deployments. One such test resulted in the disabling of a relay, which would lockout the relay until physical intervention to power cycle the relay is available. Other attacks prevent devices from responding to any valid packets at all. Regardless of implementation, the testbed provides a means of evaluating different protocol's on-device implementation and identification of vulnerabilities.

To my family.

Acknowledgments

A special thank you to my fiancée Julie. Your constant support and reassurance motivated me to persevere through the many difficulties I encountered completing this degree. Thank you to my family and friends for the endless supply of encouragement. I would like to thank my advisor, Prof. Hamid Sharif, for his indispensable experience, guidance, support and inspiration during the course of my research work. I am also grateful to Dr. Hempel for his insights and directions in my graduate research. A final thank you to Dr. Juan Lopez Jr. and Dr. Kalyan Perumalla. The guidance and advice you provided helped shape this work into a thesis I can be proud of.

Contents

List of Tables **x**

List of Figures **xii**

List of Algorithms **xiii**

List of Acronyms **xiv**

1 Introduction **1**

1.1 Testbeds 3

1.1.1 Simulated Working Conditions 3

1.2 IEC-61850 5

1.2.1 IEC-61850 Data Representation 7

1.3 IEC-61850 Substation Configuration Language 7

1.3.1 IEC-61850 Substation Configuration Description 8

1.3.2 IEC-61850 Configured IED Description 8

1.4 Thesis Organization 9

2 Problem Statement **10**

2.1 GOOSE Protocol 11

2.2 Device Under Test 12

2.2.1 GOOSE Configurations for Device Under Test 13

2.3 Deliver Cyberattacks 14

2.3.1 Existing Software Packages for Cyberattack Delivery 14
2.3.2 New Software Package for Cyberattack Delivery 15
2.3.3 Hardware Package for Cyberattack Delivery 15

3 Literature Review **17**

4 Methodology **19**

4.1 Evaluation Structure 19
4.2 Response Time Measurement 21
4.2.1 Input Measurement 22
4.3 GOOSE Packet Generation 23
4.3.1 Packet Creation 24
4.3.2 Packet Alteration 25
4.3.3 Packet Sending 26
4.4 Attack Traffic Generation 27
4.4.1 Traffic Rate Control 27
4.5 Cyberattacks 28
4.5.1 DoS Attack 30
4.5.2 Replay Attack 30
4.5.3 FDIA 31
4.6 Result Collection 32
4.6.1 Scripting Process 32
4.6.2 Testing List 33
4.7 DUT Configuration 34

5 Results, Analysis and Discussion **35**

5.1 Publishing Agent 35
5.2 Attacking Agent 36
5.2.1 Packet Generation 36

5.2.2 Attack Generation 37
5.2.3 Attack Results for Device Testing 38
5.3 Device 1 40
5.3.1 Device 1 baseline 42
5.3.2 Device 1 DoS and Replay Attack 43
5.3.3 Device 1 Relay Disabled 45
5.3.3.1 Replicating Device 1 Relay Error 46
5.3.3.2 Possible Causes of Device 1 Error 48
5.4 Device 2 49
5.4.1 Device 2 baseline 49
5.4.2 Device 2 DoS and Replay Attack 50
5.5 Raspberry Pi 52
5.5.1 Raspberry Pi baseline 52
5.5.2 Raspberry Pi DoS and Replay Attack 53
5.6 FDIA - All Devices 53
5.7 Device Comparison of Attack Responses 56
5.8 Device GOOSE Implementation Analysis 57

6 Conclusion and Future Work **58**

Bibliography **60**

List of Tables

1.1 SCL File Types 8

2.1 802.1Q Priority Codes 11

4.1 Hardware Utilized 20

4.2 Switches Utilized 21

4.3 GOOSE PDU Tags 24

4.4 GOOSE PDU Tags 25

4.5 Test List 34

5.1 Traffic Test: Expected vs. Actual 41

5.2 Traffic Test: Disabling Relay 47

List of Figures

1.1 IEC-61850 Communication Between Substation Levels 5
1.2 IEC-61850 Data Representation 7

2.1 GOOSE Packet 11
2.2 GOOSE APDU 12
2.3 Raspberry Pi 4B 16

4.1 Evaluation Testbed Topology 20
4.2 Input Measurement Connections 23
4.3 BER Data Encoding 25
4.4 GOOSE Packet Creation 26
4.5 Test Scripting Flowchart 33

5.1 Publisher's Wireshark Packet Decoding 36
5.2 Device 1's GOOSE Statistics 36
5.3 Attacking Agent Wireshark Decode 37
5.4 Replay Attack from Device View 37
5.5 Attacking Agent Sustained Traffic 38
5.6 Attacking Agent With Low Priority 39
5.7 Device 1's Attack Traffic 40
5.8 Device 2's Attack Traffics 41
5.9 Raspberry Pi's Attack Traffic 42
5.10 Device 1 baseline Response Time 43

5.11 Device 1 DoS vs Replay Response Time 44
5.12 Device 1 Disabled After Attack 45
5.13 Device 1 Post Disabled Restart 46
5.14 Device 1 Attack Disable for Packet Size 125 Bytes 48
5.15 Device 2 baseline Response Time 50
5.16 Device 2 DoS vs Replay Response Time 51
5.17 Raspberry Pi baseline Response Time 53
5.18 Raspberry Pi Response Times 54
5.19 State Number Attack Wireshark Decode 54
5.20 State Number Attack Device 1 Statistics 55
5.21 Replay Attack Successful Response Rates 56

List of Algorithms

1 GOOSE Publisher State and Timing Measurement. 22

2 Traffic Generation Timing 29

List of Acronyms

APDU Application Protocol Data Unit

ASN Abstract Syntax Notation

CID Configured IED Description

DER Distributed Energy Resources

DoS Denial of Service

DUT Device Under Test

FDIA False Data Injection Attack

FTP File Tranfer Protocol

GOOSE Generic Object Oriented Substation Event

GPIO General Purpose Input Output

HART Highway Addressable Remote Transducer Protocol

HIL Hardware in the Loop

ICD IED Configuration Description

IED Intelligent Electronic Device

MMS Manufacturing Message Specification

OT Operational Technology

PDU Protocol Data Unit

RTU Remote Terminal Unit

SCADA Supervisory Control and Data Acquisition

SCD Substation Configuration Description

SCL Substation Configuration Language

CHAPTER 1

Introduction

Operational Technology (OT) systems have begun to enhance interconnection capabilities with recent advancements in distributed computing power, allowing for centralized remote control of different systems. OT systems such as factory lines, building access systems, or power girds, were once limited in remote management capability and did not have capabilities for remote access. This achieved a means of security through physical access, since attacks would not be feasible without physical access to the system. Now with greater computing capabilities and affordability of devices have led to increased internet access across OT networks.

The capability to utilize the internet to access once air-gapped systems allows for greater monitoring of an entire system architecture. However, the additional connections create security vulnerabilities that application-specific protocols are not designed to encounter. OT systems also cover a wide range of industries and protocols used within them. A few examples of industries are manufacturing, building automation, and the energy sector. In my previous work with the Advanced Telecommunications Laboratory, we examined the different protocols in the energy sector, and how these protocols can be implemented to follow NIST Security Framework for Critical Infrastructure [1]. These protocols include Modbus, DNP3, C37, and IEC-61850. In manufacturing, protocols such as Highway Addressable Remote Transducer Protocol (HART), Foundation Fieldbus, Ethernet/IP, and

Profibus [2] are used and were often updated from serial physical connections to Ethernet. In building automation, BACnet is the international standard for communication between different components [3].

A similar theme continues throughout each sector, that serial protocols such as Modbus[4] and DNP3[5] were updated to support TCP/IP connections, but were not updated to support new data types, such as analog values like synchrophasors, provided by new components. Their communication structure also relies on a Master and Slave topology, requiring a request sent by the Master and lacking any encryption or data validation. The challenge when implementing this over TCP/IP is the inability to validate the author or keep a malicious actor from intercepting traffic. This provides a particular challenge when keeping cybersecurity in mind for networks growing more interconnected.

Before the commissioning of devices to a live environment, thorough testing of a configuration should be completed. Standard testbeds take a large investment of time and space for accurate representations of specific configurations. As an alternative to standard test beds, hardware in the loop (HIL) simulation platforms have been introduced to emulate working conditions on the power grid, suitable for evaluation of individual device configuration performance. However, validation should be completed in some form before implementing devices into a live environment. In contrast to IT devices, the devices in the OT sector are designed to be in use with zero downtime for months if not years. Due to this, software or hardware updates can take extended amounts of time to incorporate, and a device installed with an improper configuration could lead to outages on critical infrastructure.

For this thesis, a case study will be performed on the Generic Object Oriented Substation Event (GOOSE) protocol specified in IEC-61850. This protocol is becoming commonly implemented in the energy sector, due to the diverse data types and supported network topologies. The additional context give to the protocol through the Substation Configuration Language allows for customization for each systems specific services and required data exchange. The exclusion of protocols such as IEC-61850 will become a less viable option

in the future, since smart grids are increasing the number of Distributed Energy Resources (DERs). The introduction of additional DERs creates additional challenges for load balancing, and without proper information distribution to engineers, the grid would face the potential for preventable power failures.

1.1 Testbeds

OT testbeds, particularly in the energy sector, require a either a large investment in land for a live testbed, or financial investment with the purchase of HIL simulators such as the OPAL-RT or Typhoon HIL as examined in [6]. Testbeds for protocols specifically have been less investigated but often utilize simulation rather than actual hardware. This is a concern due to the assumption of identical protocol implementation and does not account for hardware differences in different components.

1.1.1 Simulated Working Conditions

The ability to create working network conditions to evaluate protocol performance is available in a variety of packages. The packages can be software developed for standard operating systems, hardware devices with set functionality, or a combination of both. When creating a testbed for evaluating OT protocols, the design choices can be differentiated by the requirement for external hardware sources.

- **Software Simulation Platforms** allow for evaluation of network traffic and device performance in a single or multiple software packages. Example software like OPAL-RT's Hypersim or OPNET network simulator have support for IEC-61850 protocols without the need for additional hardware investments.
  - **Advantages:** The software packages have limited hardware requirements to run other than the specific operating system requirements. The simulations do not

run in real time, which means results can often be obtained faster than real world results. These packages are ideal for measuring metrics like network latency.

  - **Disadvantages:** Without additional hardware, these packages are not designed to generate working signals that can be used to evaluate physical devices and their responses to different operating conditions. Therefore, these packages are not ideal for testing and commissioning of devices.

- **Hardware Simulation Platforms** provide many of the same benefits of Software Simulation packages but with additional hardware to interact with physical devices. However, configuration and setup of these devices can be a costly time investment. The added benefit is the ability to test protocols and operation on live equipment. Example Hardware platforms are OPAL-RT and Typhoon HIL.

  - **Advantages:** The additional hardware is equipped to generate operating conditions with real time requirements. Given a finite step length, the hardware package generates different signals at fixed intervals allowing devices under test to perform as they would in a real world application.

  - **Disadvantages:** Each hardware platform requires device specific software and additional hardware setup after software completion. The simulations run in real time as well, so creating and running a simulation model may take much longer than their software counterparts. The devices also have limited hardware resources, so additional designing may be required to create a model that can adhere to the fixed interval step.

- **Protocol Only Implementations** provide a means of creating the live protocol traffic to evaluate different devices. An example of a protocol only testbed is presented in [7]. While protocol only implementations cannot test fully test the effect protocol vulnerabilities will have on other operations within the Device Under Test (DUT). For this thesis, we will implement a protocol only testbed, to evaluate differences in

protocol implementation across different devices. In this thesis, we implement an efficient, flexible, reliable protocol evaluation testbed, with the flexibility for creating different cyberattack traffic at variable data rates.

1.2 IEC-61850

Substations have a variety of functionalities, ranging from transmission, distribution, and increasing or decreasing voltage. This allows for a wide variety of configurations and communication channels in different power systems. With an increase in distributed processing capabilities and IT/OT convergence, performance metrics that were previously unavailable can be transmitted from substations to system operators. IEC-61850 outlines methods to increase connectivity across a power grid, allowing for delivery of different system updates through three different levels of communication, the process level, bay level, and station level. For each of these different levels, the Sampled Values, GOOSE, and Manufacturing Message Specification (MMS) protocols were defined, as shown in Fig 1.1.

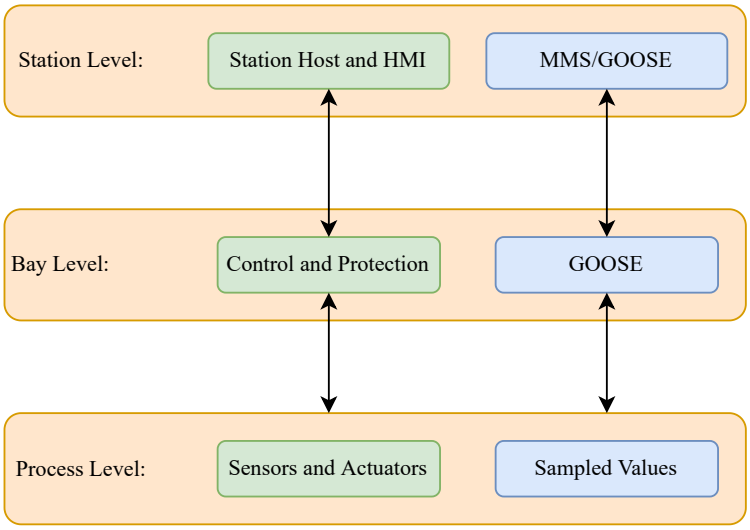

Figure 1.1: *IEC-61850 Communication Between Substation Levels*

Sampled Values

At the process level, analog values that would pertain to sensors or actuator measurements

are sent over the Sampled Values protocol. The devices monitored at this level are almost entirely analog, so the packet's data is formatted to represent analog values through digital format. The protocol utilizes a publisher/subscriber model, that sends multicast Ethernet packets to specific MAC addresses. There is no acknowledgment feature to the protocol and individual device configurations determine reception and interpretation of Sample Value packets.

GOOSE

Similar to the Sampled Values protocol, GOOSE protocol utilizes a similar publish and subscribe model between components. However, GOOSE is intended for a variety of data types that cannot be accurately represented through Sampled Value's physical measurement representation. For this, Abstract Syntax Notation (ASN) with individual data fields determined by tag, length, and data fields. Due to the range of data that can be represented by these packets, the GOOSE protocol is commonly used for both bay level and station level communication. At the bay level, communication takes place between different control and protection devices, such as relays and feeders. For communication with the station level, the communication takes place between Intelligent Electronic Devices (IEDs) and user level interfaces, such as Supervisory Control and Data Acquisition (SCADA) or Remote Terminal Unit (RTU) systems.

MMS

The MMS protocol is outlined in ISO-9506 and is based on clients and servers utilizing the protocol over Ethernet. In smart grids, this is intended for use at the station level, performing data transmission between deployed IEDs and operator monitored applications, such as a RTU or SCADA system. This protocol utilizes TCP/IP routing, with specific IP and port addresses for each client and server in the system.

1.2.1 IEC-61850 Data Representation

IEC-61850 has defined structures for many forms of data that would be required for transmission in a substation. The data representation is defined in IEC61850-7-4[8], with individual IED data being defined as logical nodes. Each of these logical nodes are then broken down into data object types, with predefined data attributes and data structures. From a substation perspective, the station level has individual bay levels to communicate with and each of the bay levels have functions related to logical nodes. Each node has defined data objects based on their logical node type, as shown in Fig 1.2.

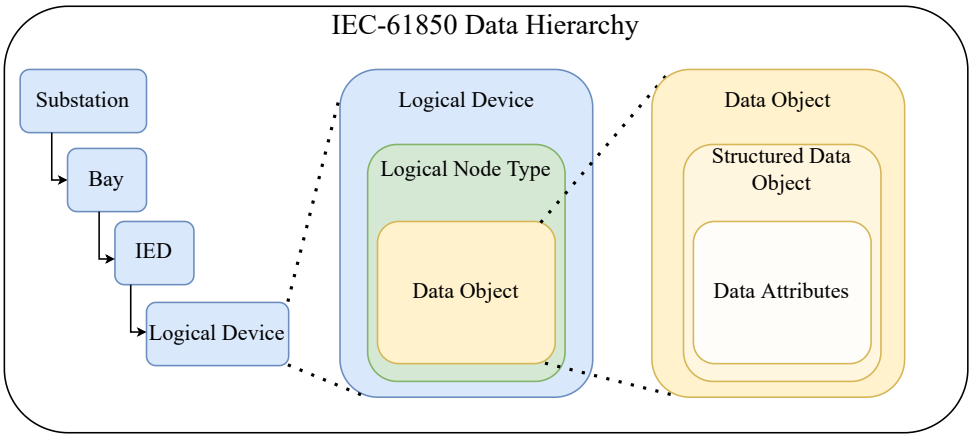

Figure 1.2: *IEC-61850 Data Representation*

1.3 IEC-61850 Substation Configuration Language

The Substation Configuration Language (SCL) is an XML based language implemented to exchange pertinent information about a whole system and individual component's data and services[9], [10]. The number of components in an individual system can vary, so different SCL file types are utilized to only send device specific information. The different types of SCL files can be found in Table 1.1. For our implementation, a single substation is represented, so the System Configuration Description (SCD) and Configured IED Description (CID) files are described in detail.

File Type	Ext.	Description
System Specific Description	SSD	Combination of single line diagram system description and individual device functions
System Configuration Description	SCD	Substation Automation system's communication and function configuration.
IED Capability Description	ICD	Individual IED communication functions and data model capabilities
Configured IED Description	CID	IED configuration for all data needed from the system.
System Exchange Description	SED	Information for data exchange between substations

Table 1.1: *SCL File Types*

1.3.1 IEC-61850 Substation Configuration Description

The SCD file contains all functional and data information of a specific substation. This file will contain information on each of the bay levels contained in the station, their IEDs and logical devices. Since each IED configuration will also contain any logical relations such as GOOSE or Sampled Value subscriptions, the SCD file also contains information regarding the interconnection between the bay level and process level. These files are used at the station level and give information to applicable RTU or SCADA systems.

1.3.2 IEC-61850 Configured IED Description

The CID file type is used to describe an individual IED's configuration, specifically regarding logical node, data type, and communication configurations. This file will contain information specific to any GOOSE or Sampled Values publications and subscriptions, their MAC addresses, priority, and data contents. While an IED may have the ability to publish additional data, the specific values that will be monitored defined in this file. This file is used to facilitate communication between an IED and the specific configuration tool software for updating the IED configuration. The information in a CID file is also contained in the SCD file, but the CID is limited to a specific IEDs configuration.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 presents the problem statement of this thesis, concerning different GOOSE configuration performance in the presence of cyberattacks. Chapter 3 provides a literature review of related works in this domain. Chapter 4 presents the proposed solutions and performance quantifiers for the problem statement. Chapter 5 presents the numerical results, analysis and discussion of the proposed solutions. And finally, Chapter 6 concludes the thesis.

CHAPTER 2

Problem Statement

In this thesis, we propose an effective, flexible, and practical testbed to evaluate different OT protocols. These protocols have use across different OT sectors, such as manufacturing, building automation, and energy sectors. As a case study, the IEC-61850 GOOSE protocol will be evaluated on different devices. While the testbed is capable of implementing different protocols, cyberattacks are of particular interest to the GOOSE protocol due to the multicast publisher and subscriber model. When taken into account that the events transmitted over the protocol may have system critical information, such as a breaker's tripped status, the ability to evaluate different configurations and their susceptibility to attacks is a valuable tool for vendors and consumers. The flexibility of SCL allows for systems to be as interconnected or isolated as desired, so a focus on GOOSE network configurations with a variety of size and number of publishers and subscribers to mimic different network topologies would need to be developed and implemented.

These differing configurations would directly impact the normal operation conditions of the IED, depending how many subscriptions it would process from the network. Moreover, the ability to emulate these different working conditions without the development of a standard test bed was evaluated. To accomplish the above objectives, we create working conditions for an IED capable of GOOSE communication, select and configure a device under test, and perform cyberattacks on potential vulnerabilities outlined in our literature

Value	Priority	Traffic Type
0	1 (Lowest)	Background
1	0 (Default)	Best Effort
2	2	Excellent Effort
3	3	Critical Applications
4	4	Video
5	5	Voice
6	6	Internetwork Control
7	7 (Highest)	Network Control

Table 2.1: *802.1Q Priority Codes*

review. There are different methods for completing each of these objectives, so evaluations of each option are outlined in this section, with our eventual implementation contained in Chapter 4.

2.1 GOOSE Protocol

The GOOSE protocol has a defined structure for each packet being sent. The high level fields can be seen in Fig 2.1. The source and destination MAC addresses are defined by configuration files such as the SCD and CID files. As such, different stations may be using the same MAC addresses for different data packets. The standard 802.1Q VLAN tag is used each Ethernet packet, including the priority field that denotes a priority between 0 and 7[11].

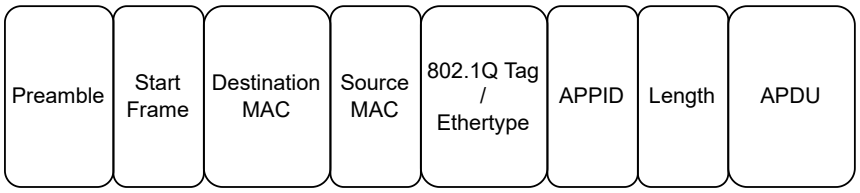

Figure 2.1: *GOOSE Packet*

The remaining data fields are specific to the GOOSE protocol, with the APPID relating directly to the publication frame in the SCD and CID files. The remaining packet fields are contained in the APDU field, which is broken down into the fields shown in Fig 2.2. These

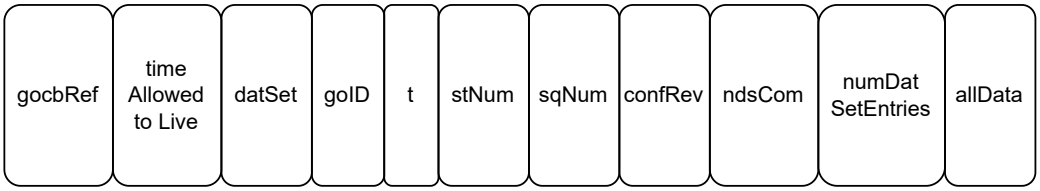

Figure 2.2: *GOOSE APDU*

fields do not all have self descriptive names, so a more in depth description of each of these fields is found below:

- **gocbRef:** Reference to the logical node control block associated with the publication.
- **timeAllowedtoLive:** Time between next publication.
- **datSet:** Reference to logical node data set associated with the publication.
- **goID:** Individual message identifier.
- **t:** Timestamp of last data set change. Multiple packets may be sent with the same value in the t field.
- **stNum:** State number of the GOOSE publication, which increments with changes in data set.
- **sqNum:** Sequence number of the GOOSE publication, increments with each transmission the state number does not change.
- **confRev:** Increases with each data set configuration change.
- **ndsCom:** Needs commissioning. This field indicates errors in GOOSE configuration.
- **numDatSetEntires:** Number of entries to follow in allData field.
- **allData:** Contains data set in ASN.1 tag, length, data format.

2.2 Device Under Test

There are a variety of IED capabilities and configurations that can be evaluated as the device under test for this model. With the availability of open-source libraries that can generate

packets for different OT protocols, the ability to evaluate protocols without high cost IEDs is possible. Low cost emulated IEDs can be evaluated for different configurations on low cost embedded environments. Libiec61850[12] is one such example of a software library with a completely implemented 61850 protocol stack. However, these devices have limited use in live operations within the power grid. The alternative to this are production IEDs from vendors aimed at the power grid.

The benefits of utilzing production IEDs for performance evaluation is the ability to test specific vendor implementation of protocols. While conformance testing done by vendors will confirm the correct implementation of the protocols that is not guaranteed with open source libraries, the presence of vulnerabilities to cyberattacks may exist. The evaluation can provide valuable information for consumers before implementing a product into their system. The potential for identifying vulnerabilities can also aid existing systems in implementing mitigation strategies.

2.2.1 GOOSE Configurations for Device Under Test

Substations have a variety of configurations that can be used to meet individual functional needs. For this reason, GOOSE configurations can vary in different levels of interconnected IEDs across different bay levels within a substation. There are added benefits to keeping the system as connected as possible, such as logic implementation for load balancing that can respond faster than physical contacts in the system. However, each publication that an IED subscribes to can generate another avenue for attack. The evaluation should then compare the performance of differing publications and subscriptions on the IED.

The configurations can also have differing priorities and VLAN configurations to alter the deliver of information across a network. The priorities of each publication and subscription are defined within the IEC-61850 standard [13]. By default, the VLAN ID assigned will be 0x001 but different devices for different bay levels can be placed on different VLANs for efficient use of network space. The priority fields defined in 802.1Q denote the different

types of packets being sent and their priority of evaluation, as outlined in Table 2.1.

2.3 Deliver Cyberattacks

When determining the method for cyberattack delivery, separate hardware and software agents need to be identified and selected. In regards to hardware requirements, network interfacing is the only hard requirement. Other functionality is not required for the scope of cyberattacks, as packet manipulation can be done with an appropriate software package. Existing hardware platforms such as Raspberry Pi or Arduino then have the available hardware requirements. For this reason, creating a new hardware package for the delivery method is not required. Currently available software packages that support the GOOSE protocol have a rigid structure that do not allow for flexibility in packet manipulation. In order to perform different cyberattacks on the device under test, either a new software package would need developed or alteration of an existing package could be used.

2.3.1 Existing Software Packages for Cyberattack Delivery

The increasing number of open-source software packages that implement OT protocols allows for a large number of options for introducing cyberattacks to the device under test. When discussing the device under test, we already mentioned the software library for libiec61850. However, the implementation of this library does not allow for individual packet field manipulation for different attacks regarding FDIA or replay attacks. Scapy[14] is a popular python package that allows for such packet manipulation, as well as custom layer implementation. By default this package does not have the 61850 protocols, so the requirement of writing an additional layer is added.

There are also existing dissectors from popular network analyzing softwares such as Wireshark [15]. The benefit of these packages is the built in capability to interpret IEC-61850 protocols. However, interacting with wireshark or its dissectors via tShark is not

designed for real time applications due to the requirement of calling an external process and awaiting returned results. This limits the ability of an agent to listen to current network traffic, manipulate packets, and deliver attacks without considerable delay.

2.3.2 New Software Package for Cyberattack Delivery

Without the availability of a software package that meets all required needs for the protocol evaluation, we could write a custom package for this purpose. The benefits of the package are design customization to the specific requirements of our evaluation. However, there is an increased development time for this approach and the requirement of recreating functionality that is widely implemented and documented in existing packages. The advantage of creating a new package will allow for precise timing of packet delivery, allowing for evenly spaced and precise attack traffic that are not directly supported by any packages mentioned previously.

This reason, along with the ability to precisely control timing with the C programming language, led us to use this option for this thesis. The customized software also allowed for the optimization of required resources, creating the opportunity to use lower cost hardware without degradation of performance. Chapter 4 will outline the steps taken to generate required traffic and mitigate resource usage. This method also lends itself to scripting of different test scenarios, allowing for limited intervention to collect the results that will be displayed in Chapter 5.

2.3.3 Hardware Package for Cyberattack Delivery

With the selection of a custom software package being utilized for the testbed, the next step was selecting a hardware package for attack delivery. There were multiple factors to consider when deciding on a hardware package. These factors are network interacting to send and monitor network traffic, and the ability to send or monitor signal lines. The most widely available packages for this testbed would be a standard personal computer or embedded

environment. In regards to a standard personal computer, the networking capabilities could be easily implemented. However, the requirement for interacting with any signal lines would require specialized hardware, that would have to be interfaced through a USB device. In this regard, an embedded Linux device would be ideal, since these devices are easily programmable, capable of reliable network communication, and have General Purpose Input and Outputs (GPIO) for interfacing with signal lines. Embedded Linux devices also has a variety of open source devices, such as the Raspberry Pi. This particular project uses the Raspberry Pi 4B, shown in Fig 2.3. Utilizing a Raspberry Pi, the testbed can implement a variety of OT protocols at standard data rates and monitor signals required for any OT signal processing. In this case, the testbed will be used to send GOOSE protocol messages to the DUT and read the digital output from the device to evaluate response time.

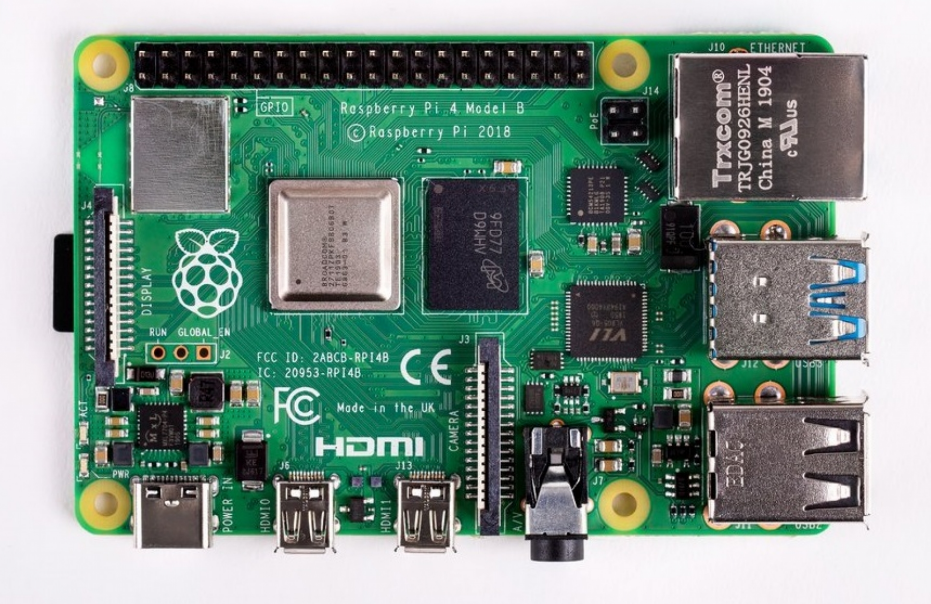

Figure 2.3: *Raspberry Pi 4B*

CHAPTER 3

Literature Review

With the multicast nature of the GOOSE protocols, research has been conducted since the protocol's inception to outline vulnerabilities and mitigation strategies. Ashraf et. al [16] outline the simple and effective nature of Denial of Service (DoS) attacks on the GOOSE protocol, which may cause failure of IEDs to respond as intended to legitimate grid events. The DoS attacks can be simple as overloading the network with packets, which do not necessarily need to have specific payloads. The authors in [17] show the efficacy of false data injection attacks (FDIA) with limited knowledge of a subsystems configuration. By replaying a packet with the same information, increment the state number, and flipping a boolean value, an IED contact was tripped incorrectly. This attack could also be repeated on different sniffed packets of the network until a desired result was achieved.

In [18], the authors illustrate different flooding attacks that IEC-61850 is susceptible to, with specific examples for the station level. The outlined attacks utilize flooding at both the MAC layer and application layer of a substation with direct access to a network switch. These attacks showed delay in time critical delivery of Sampled Values and GOOSE protocol packets. Specific attacks on the GOOSE protocol with protocol packet manipulation is outlined in [19]. The attacks outlined show that affecting individual packet fields, such as rapidly increasting the state number field of a GOOSE publication could pause reception of critical events for 2^{32} cycles. This added with the packet processing largely dependant on

manufacturer's protocol processing state machine[20], outlines the need for vulnerability assessment and external detection.

Several methods have been proposed for threat detection in systems implementing the GOOSE protocol. An example of external detection of attacks utilizing an algorithm is outlined in [21]. The reintroduction of one-time pads for lightweight authentication between publishers and subscribers is outlined by the authors in [22] and an external detection system resulting from monitoring network traffic is proposed in [23]. While each of these methods show promising results with security focus, each method requires additional commissioning of protocol features or external tools for specific systems.

The ability to add encryption would mitigate the affect of the aforementioned attacks. In [24], IEC-62351 is utilized to encrypt GOOSE communication. Evaluations of the key security was evaluated with the Scyther tool with live operation of ABB relays and an OPAL-RT OP5600. The results from this experiment founds that compliance with IEC-62351 still has security vulnerabilities. However, the required network latency for GOOSE events being published under 4ms [13] were shown to be breached when implementing the encryption of IEC-62351 in [25].

A method of testing valid GOOSE communication with the OPAL-RT is introduced in [26]. The test acted on a ABB relay to evaluate GOOSE performance from the ABB relay on a DER as part of a larger network. This work also outlines operation of fixed times steps within the simulator itself. OPAL-RT's line of simulators operate by compiling MATLAB/SIMULINK models into C code. As the complexity of a model increases, the time required for computation to finish also increases. The authors [27] utilized an HIL simulator platform to evaluate the whole IEC-61850 suite of protocols, with a specific focus on the MMS protocol. Other evaluations of specific protection scheme configurations are introduced in [28], with a focus on individual breaker configuration on system performance.

CHAPTER 4

Methodology

This thesis aims to address the aforementioned prominent problem statements in the evaluation of vulnerabilities in vendor GOOSE protocol implementation. In the literature review, we mentioned that compliance with GOOSE protocol standard does not guarantee resistance to protocol vulnerabilities. To evaluate vendor implementation and possible vulnerabilities, emulation of working conditions and cyberattacks would need to be developed.

4.1 Evaluation Structure

The evaluation structure relied on creating normal network traffic on the Ethernet port of the Device Under Test (DUT), measuring response time and accuracy, and comparing those results when the device is under protocol specific cyberattacks. In the literature review chapter of this thesis, we mentioned the GOOSE protocol's susceptibility to DoS attacks, replay attacks, and FDIAs. The testbed outlined in this chapter utilizes two Raspberry Pi's to create normal GOOSE traffic and measure the response time, while the second generates attacks at specific traffic rates for each vulnerability attack. A high level diagram can be seen in Fig 4.1

The tests for each of the attacks were then tested against three different devices with GOOSE communication capabilities. The first two devices were from reputable manufactur-

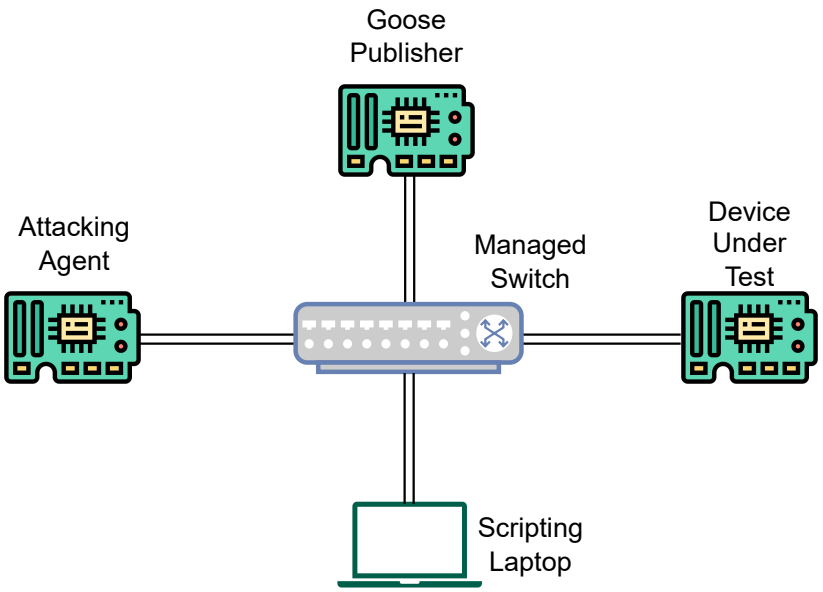

Figure 4.1: *Evaluation Testbed Topology*

ers within the energy sector. The first device is a protection system, which allows for a wide range of customization with the number of publishers or subscribers assigned. Configuration for this device is a single subscription with one bit being tracked for measured response time. This configuration is the lightest available for the device, allowing for prioritization of only the measured subscription. The virtual bit measured for this configuration are to measure the current value of a boolean value in the GOOSE subscription. For the rest of this paper, this device will be referred to as Device 1.

Table 4.1: *Hardware Utilized*

Device	CPU	RAM	OS
Raspberry Pi 4B [29]	BCM2711 @ 1.5GHz	4 GB	Rasbian
Raspberry Pi 4B [29]	BCM2711 @ 1.5GHz	4 GB	Rasbian
HP Spectre x360 [30]	i7-8550U @ 1.8GHz	16 GB	Windows 10

The second device tested is a protective relay. The device is advertised as a more general purpose device, and allows for a set number of GOOSE subscriptions. Similar to the configuration of Device 1, this device, Device 2, was configured with a single subscription with monitored boolean bit for output outlined in the traffic generation section of this chapter.

The final device for testing was another Raspberry Pi with a GOOSE subscriber configured with the C implementation of libiec61850. All devices for these tests were configured without any other communication protocols active. These configurations were generated to give the best case results for each device in response to each attack. The hardware utilized for each role of the testbed framework can be seen in Table 4.1.

In order to streamline the testing of three devices without encountering a time constraint, three different testbeds were created for each device. This required the utilization of three different, separate switches to avoid any conflicts created with additional components plugged into the switch. The switches utilized for the test beds can be found in Table 4.2. This also allows us to eliminate the possibility of a switch affecting the testing of a certain device. If results were unexpected or unable to be validated on a given switch, the tests could be rerun on a separate switch to ensure results were directly caused by the attack traffic.

Table 4.2: *Switches Utilized*

Manufacturer	Model	Link	Ports
Netgear	GS724T [31]	Gigabit	24
Netgear	GS716T [31]	Gigabit	16
Cisco	SG110-16 [32]	Gigabit	16

4.2 Response Time Measurement

The first component created for the testbed was the GOOSE publisher. This device generates GOOSE publications sent to the switch for the MAC address subscribed by the DUT. For the tests conducted in this thesis, the publishing address is 01:1C:CD:01:00:03. The device then reads the response from the DUT, expecting a 1 for a true boolean value and 0 otherwise. The time response time is then measured, or timed out if more than 700 ms have passed. If a timeout is detected, the sequence number is incremented rather than the state number. This gives the device an opportunity to detect the same event, and follows the required GOOSE messaging timing since 700 ms have already passed. Algorithm 1 summarizes the timing

measurements when considering timeouts or a successful response.

Algorithm 1 GOOSE Publisher State and Timing Measurement.

Require: state, sequence, input
  $t_0 \leftarrow$ Packet Send Time
  $t_c \leftarrow$ Current Time
  $b_0 \leftarrow$ Expected Boolean State
  $b_1 \leftarrow$ Pin Logic State
  failure = 1
  while $t_1 <$ 700ms OR b_0 != b_1 **do**
    $t_1 = t_c - t_0$
  end while
  if $t_1 >$ 700ms **then**
    failure = 0
    sequence++
  else
    state++
  end if
  return *failure*

This method of measuring responses from the devices allows for the GOOSE publisher to react to the DUT and removes wasteful cycles, as switching to a high when the input is already high would result in a false positive. The tests run for each of the response times were run for 1000 total cycles to allow for accurate measurement of response time and successful measurement in a statistical approach. A cycle in this sense is defined as any change in the state number of the GOOSE Publisher. For these tests, only the boolean value was altered by the publishing agent to allow for usage of each cycle produced.

4.2.1 Input Measurement

While the Raspberry Pi would only require the connection of a GPIO and common ground for measurement, the other devices tested used contact type digital outputs. The main difference between these is the inability to produce voltage on their own. This is typical of devices used on the power grid, as voltages and loads can vary for different applications. This required an alteration to the connections from the publishing agent, as a voltage was

required, but no common ground since the agent is reading back its own output. An example of these connections can be seen in Fig 4.2.

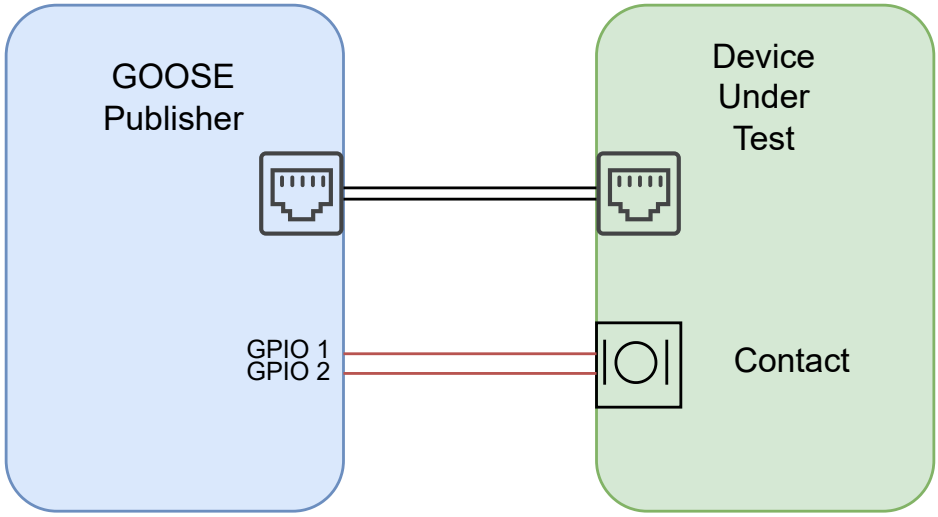

Figure 4.2: *Input Measurement Connections*

This method of connections may lead to variation within the measured response time due to contact uptake or release timing, so the statistical approach towards the results mentioned in the previous section gives stronger confidence to the results outlined in Chapter 5. However, there are still expected variations within the response, since the Raspberry Pi is not a real time operating system. Due to these constraints, small variations within data will not be considered a device shortcoming. Measurement of the response time was completed with the WiringPi [33] library, since the library gives simple API calls to the individual GPIO on the Raspberry Pi.

4.3 GOOSE Packet Generation

There are multiple software packages available that are capable of creating and generating GOOSE packets. The most notable is scapy [14], which utilizes python to create customizable packets through customizable fields. However, scapy has not built in support for the

GOOSE protocol and the sending methods utilized provided are not well suited for generating high traffic rates. For this reason, a new program was written in the C programming language to allow for creation of new packets, alteration of existing packets, and sending of packets.

4.3.1 Packet Creation

Packet generation is either completed through the use of default arguments or provided parameters. The attacks in this thesis don't require large alterations between attacks, so a list of default parameters, which are hard coded into the program, were used. The required parameters were outlined in 2, which included all fields required to generated a GOOSE PDU and APDU. The process of generating a packet required parameters to be encoded using the BER and ASN encoding, which required definition of each tag for the GOOSE fields. These tags can be seen in Table 4.3.

Table 4.3: *GOOSE PDU Tags*

Field	**Tag**		**Field**	**Tag**
PDU	0x61		gocbRef	0x80
timeAllowedtoLive	0x81		datSet	0x82
goID	0x83		t	0x84
stNum	0x85		sqNum	0x86
test	0x87		confRev	0x88
ndsCom	0x89		numDatSetEntires	0x8A
allData	0xAB			

With proper construction of the header, data types needed to be encoded as well. The tags for each data type can be seen in Table 4.4. The ability to craft different packets has a few challenges regarding differing length between packets, as shown in Fig 4.3. A valid GOOSE packet also contains 3 different length fields that are reliant on the remaining length of the packet. These are, the length field, PDU length field, and allData length field. This is in addition to the length fields for every field listed in the previous table, along with each entry in the data set.

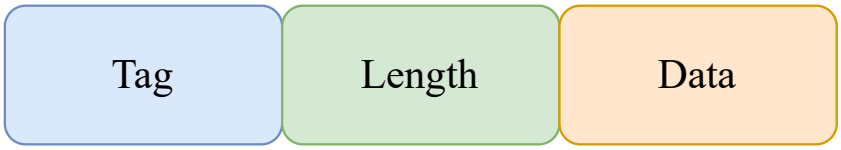

Figure 4.3: *BER Data Encoding*

Due to these challenges, the packet was generated starting at the data set. Since every field in the packet is required to use the same encoding scheme, a function which would take the tag, raw data, and length to return the encoded string was created. With this basic functionality implemented the packet was formed by creating the dataset, calculating the length for the allData field, creating the rest of the APDU, calculating the PDU length field, and then generating the rest of the GOOSE packet. This process is illustrated in Fig 4.4.

Table 4.4: *GOOSE PDU Tags*

Field	**Tag**		**Field**	**Tag**
Boolean	0x83		Bitstring	0x84
Integer	0x85		Unsigned Integer	0x86
Fload	0x87		Real	0x88
Octet String	0x89		Visible String	0x8A
UTC Time	0x8C		Binary Coded Decimal (BCD)	0x8D
Boolean Array	0x8E		Object ID	0x8F
UTF-8 String	0x90		Structure	0x2A

An important step in the packet generation that will save time in later alterations is the indexing of data fields in the GOOSE packet. For every packet that will be created in the program, the packet generation function will alter a specified packet buffer and list of field indices. These indices are made available for every GOOSE field listed in the tables above. For this reason, the data can be altered and resent as in the case of an FDIA, or to imitate a valid GOOSE publisher.

4.3.2 Packet Alteration

The packet generation step mentioned the availability of packet indices for each field of the GOOSE packet. With this information, attacks that require packet updates, like the FDIA

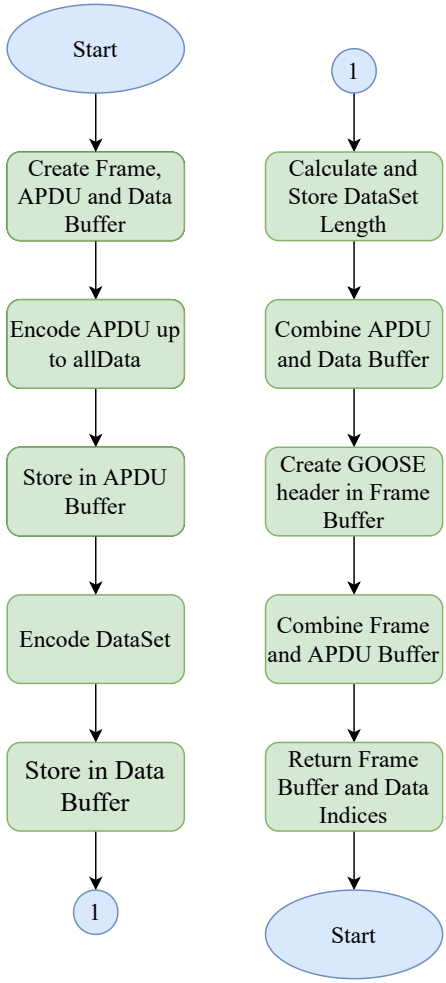

Figure 4.4: *GOOSE Packet Creation*

with updated stNum and t fields can be properly executed. The additional capabilty that should be considered is the additional length that may be added while changing packet fields. While the time stamp is always 8 bytes long in the GOOSE protocol, the state number can have any value up to a 32 bit integer. When this field exceeds 255, the length of the field increases along with the packet length and pdu length, so any successful packet alteration need to consider all effected length fields.

4.3.3 Packet Sending

The most influential step for writing our own packet generation is the requirement for precise traffic control in attacks. While initial testing with large packets was able to sustain constant

data rates, smaller packets closer to 100 bytes were much less reliable when using available open source packages. To resolve this issue raw sockets were utilized, providing an method to send the raw bytes directly to an interfaces address.

4.4 Attack Traffic Generation

The central component of the evaluation structure is the attacking agent, that generates attack traffic to the switch. By having the attacker on a separate port, the attacks are similar to real world attacks against the grid, where a malicious actor gains access to a network and not the physical device itself. A central challenge to generating cyberattack traffic is the data rate required of each attack. The devices tested in this thesis come attached with a 100BASE-Tx Ethernet ports, so accurate control of network traffic from 10-100 Mbps is required for accurate results in response to cyberattacks.

When executing an attack, the attacking agent requires the option of changing packet contents for each packet transmission. An example would be the False Data Injection Attack (FDIA), which requires an updated timestamp and state number field before transmission of a new packet. This is different from carrying out a DoS or replay attack, which can utilize the same packet retransmitted for the desired traffic rate.

4.4.1 Traffic Rate Control

Transmitting at specified rates is a challenge for hardware constrained devices such as the Raspberry Pi. While the provided architecture can be executed without issue on desktops with greater resources, validating operation on the Raspberry Pi will allow for accurate control on a wider range of hardware. The first technique used to ensure accurate traffic generation is traffic generation in bursts. The goal with bursting the traffic is to reduce the required precision timing on the part of the Raspberry Pi, as 100 Mbps can require as many as 100k packets to be sent per second. With the introduction of bursts, traffic under 30 Mbps

will send 100 bursts per second, 30-80 Mbps traffic will send 300 bursts per second, and greater than 100 Mbps will send 500 bursts per second. This limits the maximum number of packets sent in a single burst, while allowing the built in Linux timers to keep up with scheduling packet transfers.

These burst numbers were created through trial and error, as smaller burst sizes with more occurrences worked well on desktop environments, but not in an embedded environment. An advantage to the bursts is the required resolution for our timers only needs to track down to 2 ms, compared to 10 μs required if bursts were not utilized. The tests for determining burst size were executed with a 125 byte long packet. This is small for the GOOSE protocol and requires more packets to match the required traffic generation. Completion of testing with these small packets led to the same accurate performance with larger packets, since there are less packets that require sending to the socket. The method of timing and scheduling packets is described in Algorithm 2.

The main addition in traffic generation to the burst is the addition of the slept flag. In linux operating systems, there is no guaranteed resources for the whole run time of the attack. This is mitigated by checking the current elapsed time compared to the next scheduled burst. If the current time is behind, the next burst will double in size and attempt to complete 2 bursts in a single cycle. This, along with increasing the priority of the attacking process using the "nice" command created reliable traffic generation that will be outlined in Chapter 5.

4.5 Cyberattacks

Chapter 3 outlined the current research being undertaken for GOOSE communication protocol vulnerabilities. To include the most common attacks against the GOOSE protocol specifically, DoS, Replay, and FDIA attacks were implemented in a single attack program. The software for the attacking agent was written in the C programming language, utilizing a

Algorithm 2 Traffic Generation Timing

```
Require: load, duration, packets, bursts
  t_n ← current time nanoseconds
  t_s ← current time seconds
  burst size = packets / bursts
  burst duration = 1000000000 / bursts
  temp burst = burst size
  elapsed = 0
  while elapsed < duration do
    sent = 0
    slept = 0
    current burst = t_n
    while sent < packets do
      next burst = current burst + burst duration
      if slept then
        burst size = 2 * burst size
        next burst = current burst + burst duration
        slept = 0
      end if
      for i in burst size do
        send packet
      end for
      burst size = temp burst
      sent += burst size
      current burst = t_n
      if current burst > next burst then
        slept = 1
      else
        while current burst < next burst do
          current burst = t_n
        end while
      end if
    end whileelapsed = t_s
  end while
```

packet generation format implemented without external libraries. The packets for generating the attacks were all created using default parameters provided in the attack program itself.

The parameters provided externally are the traffic load, required duration, and state number attack flag. These parameters all expect integers for input values, so precision of attacks is accurate to 1 Mbps increments and whole seconds. While operation of attacks higher than 100 Mbps are accepted from this method, these would exceed the saturation point of the Ethernet link for our DUTs and will not be tested. The final flag indicates the usage of the state number attack. When this flag does not equal 0, the state number attack is implemented with the timestamp and state number fields updated at ever packet sending iteration.

4.5.1 DoS Attack

The DoS attacks goal is to saturate a link, overwhelm the attack endpoint, and cause an inability of a device to respond to regular traffic. These attacks are easily detectable by netowrk monitoring software and are typically mitigated by cybersecurity practices outside of the DUT itself. For the implementation of this attack, a predefined GOOSE packet that is not regularly in use by the network is generated and delivered at the specified data rate calculated by the attacking agent. This attack has the least complexity of the other attacks and requires no knowledge of the current GOOSE implementation on the network.

4.5.2 Replay Attack

The replay attack generation has many similarities to the Denial of Service attack, with an exception to the packet contents. The replay attack intercepts an valid GOOSE packet present on the network, and re-transmits the packet at a later time to confuse the IED into discarding newer valid packets. These attacks are mitigated by the state number and timestamp fields of the GOOSE protocol, but sent at high rates can cause the IED to fail to process valid packets in addition to attack packets. In terms of complexity, this attack

typically requires access to the network to capture packets before attacking. This requires some knowledge of the GOOSE protocol but is low complexity once the initial GOOSE packet is obtained.

4.5.3 FDIA

The FDIA in this thesis is also referred to as the state number attack. For this attack to be properly carried out, a valid GOOSE publication needs to be generated, with a valid time stamp and state number field to trick the IED into only processing the attacking packets. In the GOOSE protocol, changes in a publication status are indicated with the state number field. In normal operation of the protocol, the state number field rarely changes. So the injection of valid packets with higher state numbers can trick the IEDs into interpreting valid packets as outdated. To mitigate the chances of devices interpreting the attack traffic as out of sequence, the attacks performed in this thesis start with a state number of 0 and increase until the attack is completed.

With this attack, there is some concern for the state number rolling over. However, the state number is an unsigned 32 bit integer, allowing for 4,294,967,296 states before this would happen. Given our highest attacking rate is 100 Mbps, and our smallest packet is 125 bytes, then we can attack a DUT for over 11 hours without fear of the state number rolling over. However, this attack can increase in complexity and generate incremented sequence numbers as well. By updating the state number well above the current state of the device, the sequence number can be incremented in normal GOOSE intervals without detection.

While this attack has the highest complexity of the three, it is also the most difficult to detect. A well executed state number attack does not require the same bandwidth that DoS or replay attacks consume and would be harder to detect without manual review of network traffic. This attack is usually mitigated by controlling port access on a switch. Most IEDs recommend that typical remote access protocols be disabled in order to prevent external access. This, paired with dedicated VLANs for IED networks can reduce the availability of

the attack on deployed devices.

4.6 Result Collection

The final component of this test bed is the scripting laptop. The laptop is connected to the switch and has the task of starting regular traffic from the GOOSE publishing device, attack traffic from the attacking agent, and Wireshark network monitoring to verify attack traffic rates. The publisher and attacking agent have SSH enabled, allowing for tasks to be started remotely at the same time without manual intervention. Individual test and response times are logged at the GOOSE publisher, so the scripting laptop collects and stores individual run results for ever iteration of the testing process.

4.6.1 Scripting Process

With the tasks defined, the scripting process was completed with the Python programming language. This allows for simple cross-platform implementation. While the current scripting laptop is running Windows 10, the code could be repurposed to run on other operating systems with limited code alterations. The scripted is intended to allow for multiple long running tests to be run sequentially without the need for physical intervention. In order to accomplish this task, a list of parameters for each run are given to the script. For each parameter, individual SSH sessions are setup for each publishing and attacking agent. Their respective programs are initiated along with Tshark for monitoring of attack traffic. Once the scripts have run to completion, the next set of parameters is loaded and the process is repeated, until no long parameters remain. The software flowchart for the script process can be seen in Fig 4.5.

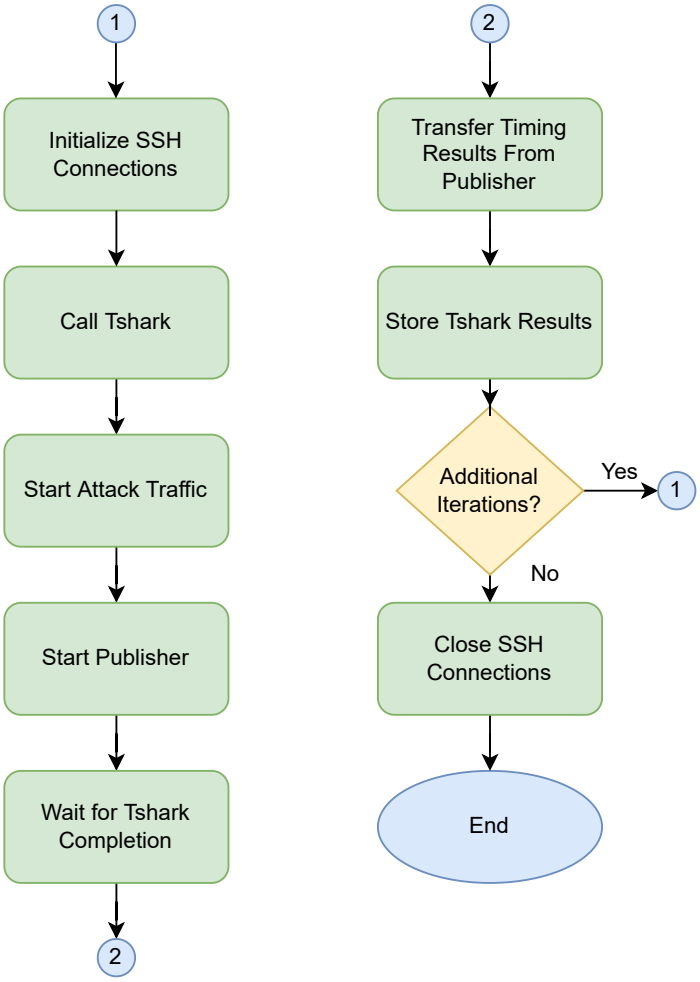

Figure 4.5: *Test Scripting Flowchart*

4.6.2 Testing List

To compare the different operating characteristics of the 3 different devices tested in this thesis, a baseline response time for each device is established by running the device with no attack traffic for 1k cycles. Each attack outlined in the previous sections will then be run with traffic levels of 10 - 100 Mbps, incrementing by 10 Mbps each time. This allows for comparison of the device response to each attack to the device baseline as well as other attacks. The complete list of tests can be found in Table 4.5. The three devices tested were discussed in a previous section and are labelled as Device 1, Device 2, and a Raspberry Pi.

Table 4.5: *Test List*

Device	**Baseline**	**DoS** 10-100 Mbps	**Replay** 10-100 Mbps	**FDIA** 10-100 Mbps	**Total**
Device 1	1k Cycles	10k Cycles	10k Cycles	10k Cycles	31k Cycles
Device 2	1k Cycles	10k Cycles	10k Cycles	10k Cycles	31k Cycles
Raspberry Pi	1k Cycles	10k Cycles	10k Cycles	10k Cycles	31k Cycles

4.7 DUT Configuration

Each individual device requires a different configuration to properly respond to the publishing agent's messages. Device 1 requires the use of multiple software to enable GOOSE subscription, and then the virtual bit needs to be mapped to an output in a separate software to enable the output in response to specific messages. Device 2 can manage GOOSE implementations and map a virtual bit to their output logic matrix for responding to a GOOSE message. Each of the devices for these tests were left in their default state, without other protocols enabled to prevent interference with the test results.

CHAPTER 5

Results, Analysis and Discussion

In this chapter, we will present the numerical results for the attacking agent's traffic generation, the response to different attacks from the different DuT, and compare their results. The results for each device tested will be broken down into their baseline, response to DoS and Replay attacks, and response to FDIA. Analysis of these results and future work will be outlined at the end of this chapter.

5.1 Publishing Agent

The publishing agent has limited requirements for successful implementation, with the requirements being publishing valid GOOSE protocol and having it be interpreted by the DuT. For this section, we demonstrate successful decoding from Wireshark, shown in Fig 5.1. This is meaningless however, if the DuT does not recognize the packet, so Fig 5.2 shows Device 1's GOOSE subscription status, with incremented state and sequence number. If these packets were not interpreted, the state and sequence number would remain at 0.

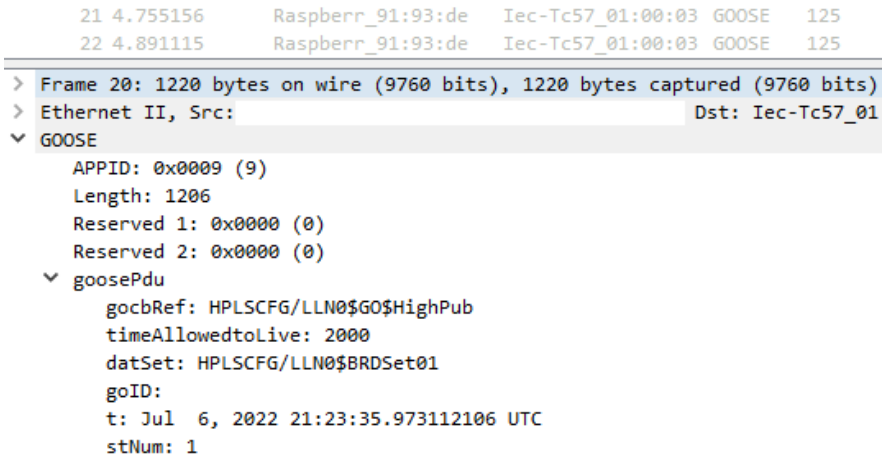

Figure 5.1: *Publisher's Wireshark Packet Decoding*

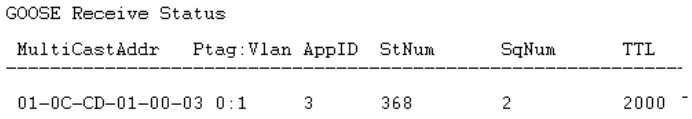

Figure 5.2: *Device 1's GOOSE Statistics*

5.2 Attacking Agent

When developing the attacking agent, there were two different traffic scenarios to consider, the traffic at the attacking agent, and the traffic after the switch. However, this traffic is also useless if the packet does not conform to the GOOSE protocol. The results shown in this section outline the correct generation of GOOSE protocol packet and sending these packets at the desired data rate for a given attack.

5.2.1 Packet Generation

The first step for the attacking agent was ensuring the individual packets sent by the attacking agent were valid GOOSE packets. Without this step, the attacks would not have the intended effect on each DuT, and would be discarded by each device for being improperly formatted. Fig 5.3 shows the decoding of the packet in wireshark, showing that the packets are recognized as GOOSE packets.

These packets were also received by Device 1 for testing, since a live view of subscription

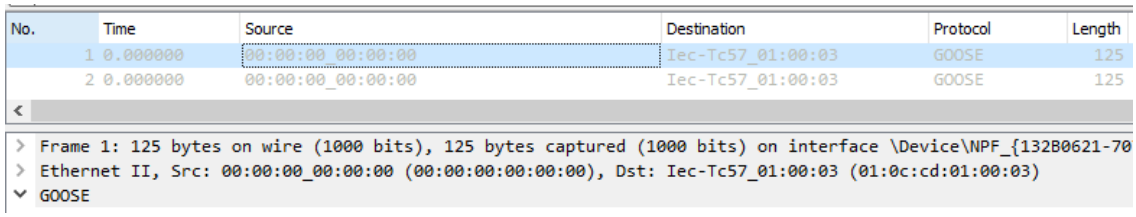

Figure 5.3: *Attacking Agent Wireshark Decode*

status is available. For this screenshot, shown in Fig 5.4 the replay attack packet was utilized, so the device would attempt to decode given the valid destination address. However, the timestamp is invalid, so the packets are shown as corrupted. A screenshot showing valid packet decoding will be shown in the FDIA section of this chapter.

```
Ctrl Ref:
AppID    : 3
From     : 06/22/2022 18:02:05.140  To: 06/29/2022 19:41:55.612

Accumulated downtime duration                      : 0162:51:59.952
Maximum downtime duration                          : 0099:34:57.334
Date & time maximum downtime began                 : 06/22/2022 18:02:05.169
Number of messages received out-of-sequence(OOS)   : 1157
Number of time-to-live(TTL) violations detected    : 37
Number of messages incorrectly encoded or corrupted: 238946
Number of messages lost due to receive overflow    : 16109435
Calculated max. sequential messages lost due to OOS: 150
Calculated number of messages lost due to OOS      : 264
```

Figure 5.4: *Replay Attack from Device View*

5.2.2 Attack Generation

With the packet generation properly operating, the traffic generation is executed as shown in 2. The results shown in Fig 5.5 show the traffic levels generated by the attacking agent for values 10-120 Mbps for 35 minutes each. During this test, there are variable spikes, as resource allocation is not guaranteed in the Raspberry Pi, but Table 5.1 shows the average traffic generation over the whole test duration has an error rate of less than 0.7%.

A reason for measuring the traffic of the network at both the attacking agent and device under test shows that the link saturation occurs at an earlier traffic rate than expected. In networks with managed switches, we expect there to be a point before the link meets the 100 Mbps data rate that the switch no longer allows packets to go through. In this case, we

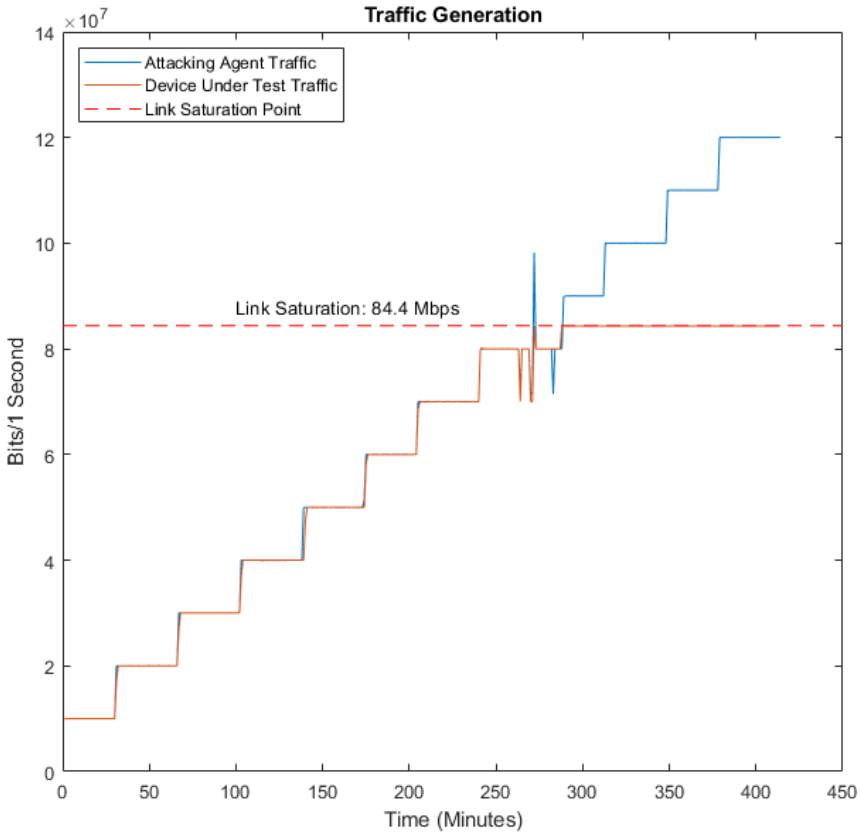

Figure 5.5: *Attacking Agent Sustained Traffic*

see that point to be 84.4 Mbps. For this section, the test was repeated at 1 minute intervals rather than 35 minutes, and returned the same saturation point for each test. That explains why there is a higher error rate at the DUT past 80 Mbps, as the device is receiving the maximum allowed traffic.

5.2.3 Attack Results for Device Testing

Once the traffic generation was validated with the long running test shown in Fig 5.5, the testing was initialized on all 3 devices. Initially, a test run was completed on a new Raspberry Pi, without the increased priority of the "nice" command and WiFi and Bluetooth still enabled. The results shown in 5.6 illustrate the issue without requiring more resources for the Raspberry Pi. Lower traffic cases are unaffected, but higher data rates can get suck,

such as 90 Mbps being stuck at 83 Mbps. There is also an issue of failing to catch up to the desired data rate. The traffic will jump up to 180 Mbps for a few seconds to try and recover. This is a downfall of not using a more complex algorithm that tracks per second traffic but can be mitigated utilizing the increased operating priority.

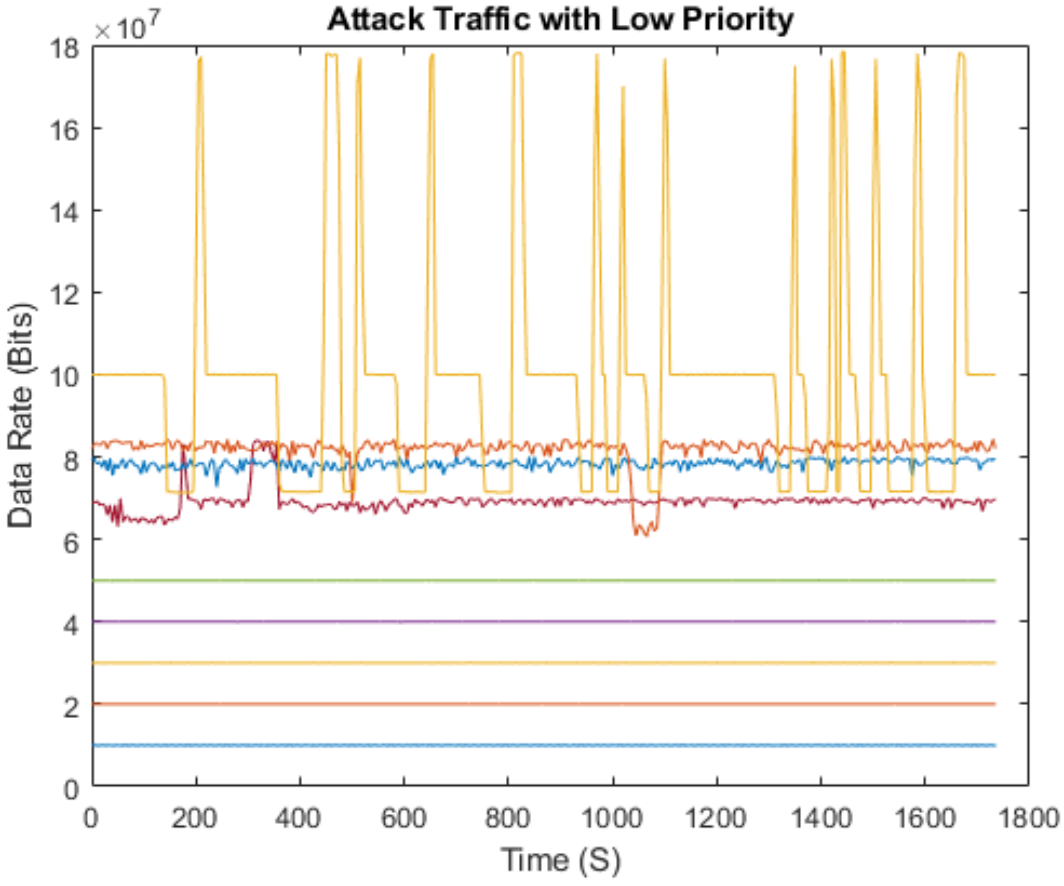

Figure 5.6: *Attacking Agent With Low Priority*

With the method for producing consistent traffic generation identified, the DoS, Replay, and FDIA attacks were executed on the three devices. Later in this chapter, we'll discuss why the FDIA attacks were not included in this figure, due to inability of devices to respond at any attack traffics. The attack traffic generated for each device is shown in Fig 5.7, Fig 5.8, and Fig 5.9. The results shows the output from the attacker for each test, and we can observe that the traffic rate errors encountered with sporadic traffic were eliminated. When running these tests on each DUT, three different Raspberry Pi's were utilized, showing each

is able to output comparable results.

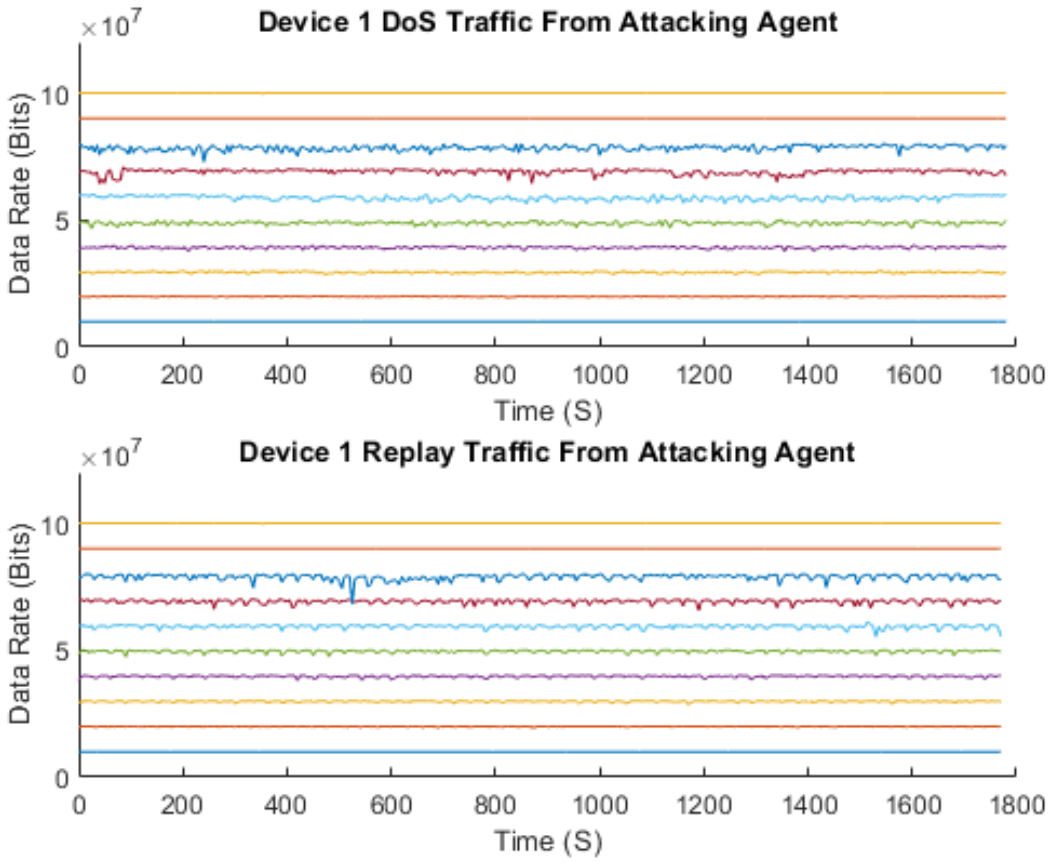

Figure 5.7: *Device 1's Attack Traffic*

5.3 Device 1

The first device tested was Device 1. This device was the most expensive tested and delivered the most unexpected results. The baseline results showed a fast response time to any GOOSE event and a resistance to DoS and low traffic replay attacks. However, later sections will show the device was unable to handle higher attack data rates and was actually disabled and unable to respond.

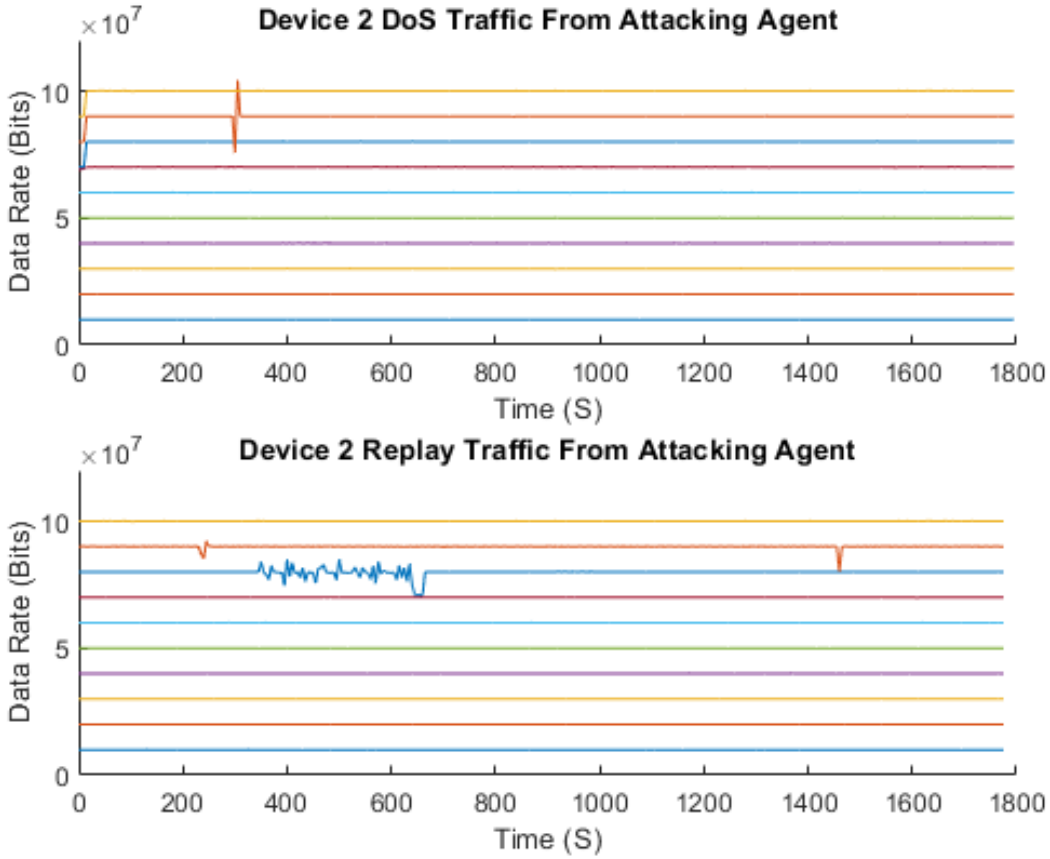

Figure 5.8: *Device 2's Attack Traffics*

Table 5.1: *Traffic Test: Expected vs. Actual*

	Attacker		Device Under Test	
Expected	Actual	% Error	Actual	% Error
10000000	10010050.7	0.100507	10011456	0.11
20000000	20013620	0.0681	19927015	0.36
30000000	30013394.8	0.044649	29922947	0.26
40000000	40274380.7	0.685952	39924444	0.19
50000000	50057098.7	0.114197	49933272	0.13
60000000	60015123.4	0.025206	59933035	0.11
70000000	70018766.7	0.02681	69964742	0.05
80000000	79683505.8	0.395618	79567073	0.54
90000000	90012522.9	0.013914	84326689	6.30*
100000000	100014383	0.014383	84326625	15.67*
110000000	110022673	0.020611	84326537	23.34*
120000000	120014597	0.012164	84326552	29.73*

* - Caused by link saturation

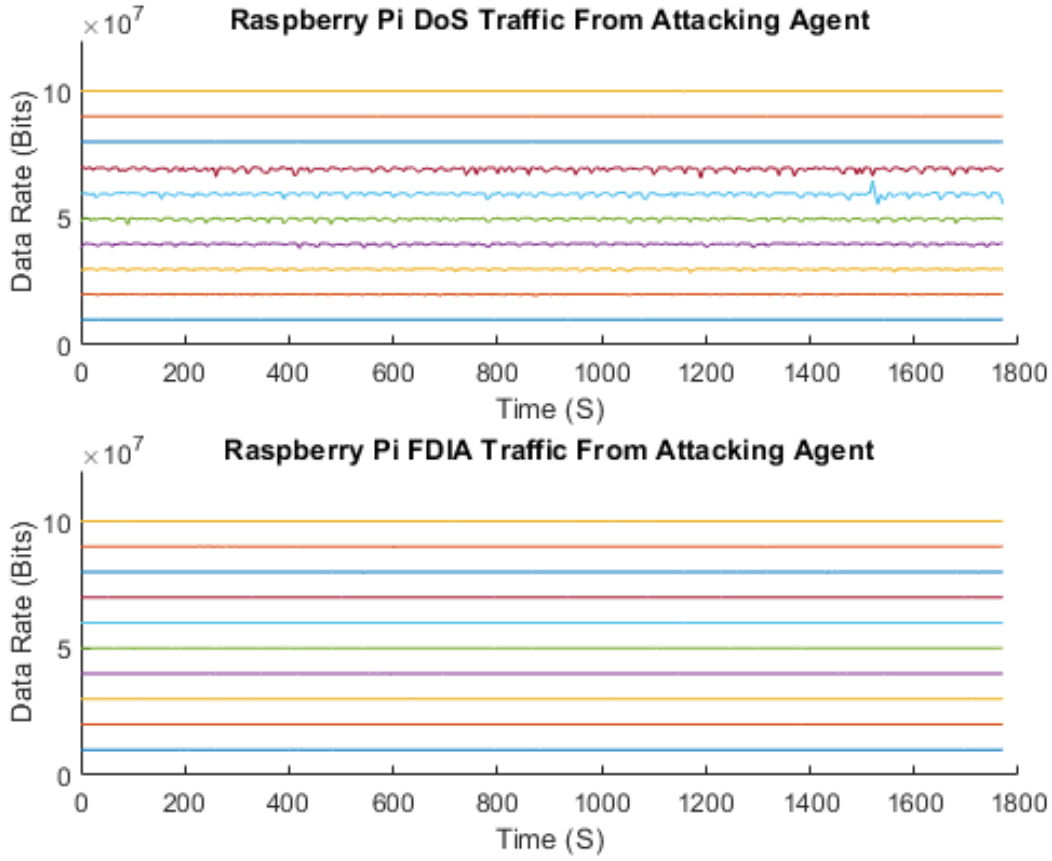

Figure 5.9: *Raspberry Pi's Attack Traffic*

5.3.1 Device 1 baseline

The initial results without attack traffic shown in Fig 5.10, give a tight grouping of responses under 10 ms. The publisher for this test cycled the boolean between on and off states every 1-4 seconds. This created a random element to the switching time, while creating valid sequence increasing packets in between the rising state value. Device 1 also allows for live view of GOOSE subscription statistics, which showed 0 dropped packets during testing. This shows under normal operating conditions, Device 1 is a fast, reliable device in response to GOOSE events.

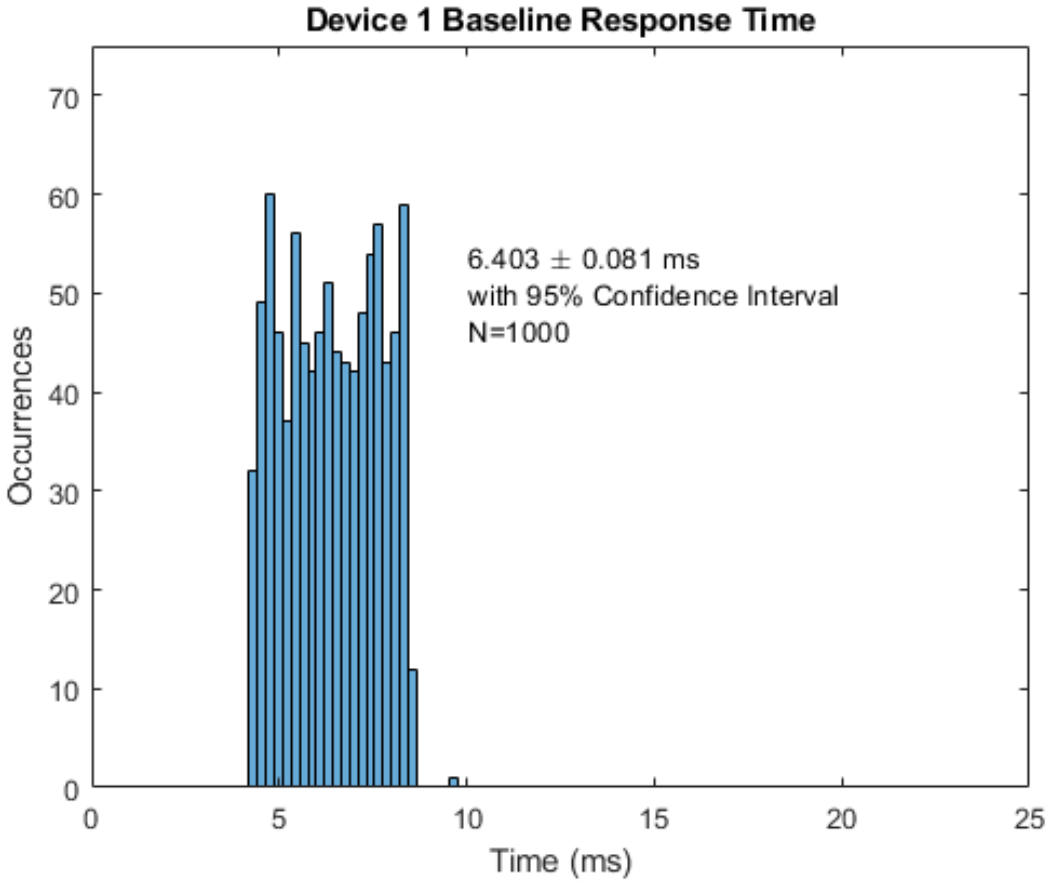

Figure 5.10: *Device 1 baseline Response Time*

5.3.2 Device 1 DoS and Replay Attack

The DoS and Replay attacks provided surprising results, which are shown in Fig 5.11. The DoS attack had seemingly no effect on Device 1's response time, with the responses hovering around 6ms and not exceeding 7ms on average. This is due to the processing cycles Device 1 uses, which do not process any packets that have an invalid destination MAC address. From the packet generation discussion of Chapter 4, the packets utilized for this attack are coded to utilize a valid GOOSE destination MAC address. However, this does not guarantee the packets are being sent to an address that the device is listening on. The lack of effect on at data rates up to 100 Mbps, show that the managed switch is limiting the number of packets sent to the device from our attacking agent, while allowing regular traffic to be sent to the device. We can also see that the device has no issue filtering the packets,

even at data rate close to the link maximum.

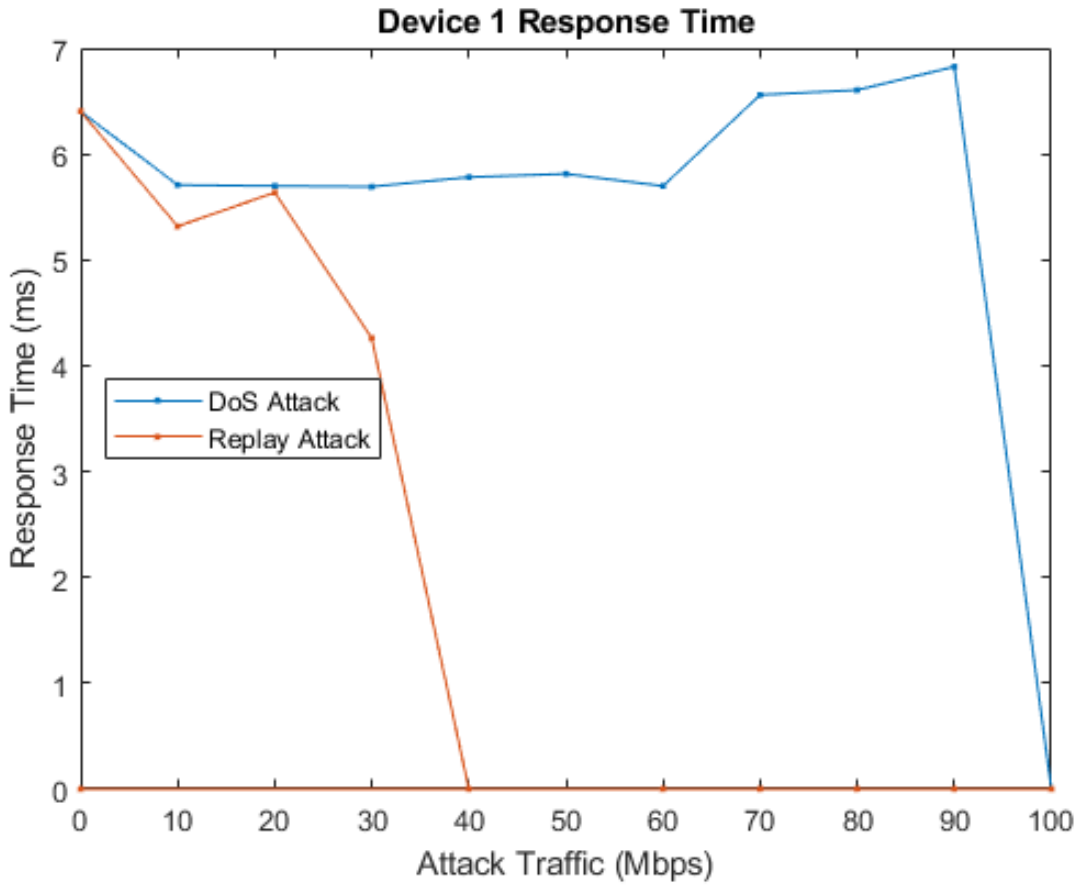

Figure 5.11: *Device 1 DoS vs Replay Response Time*

The replay attack provided the most surprising responses. We can observe the curves that the response time seems to uptick and match DoS at a traffic level of 20 Mbps. With additional testing, it is expected that all response times will closely converge with the DoS response time. This variance is likely due to the small number of successful responses in the data set. The tests were run sequentially, with each 1k cycles taking roughly 35 minutes, with a total run time of roughly 7 hours.

The replay tests were scripted and run overnight, with the error shown in Fig 5.12 awaiting in the morning. This explains the drop from responding at 30 Mbps to no response at 40 Mbps. The exact cause of this error is further explored in the next section of this chapter. The success rate of each device is graphed and compared in the Comparison section

of this chapter. An important take away from this graph is the lack of impact the attack had on response time. If the valid packet sent from the publisher was not discarded due to receive overflow, the response time remained consistent for ever iteration. This implies that, while the relay remains active, attacks will not cause a response time issue on the network. It also shows that given enough re-transmissions, the relay will eventually operate to the correct state. This can still be detrimental to the network's operation, as GOOSE messages are meant for high priority event with fast reactions. The exact success rate of Device 1 against replay attacks will be examined in the Comparison section of this chapter.

5.3.3 Device 1 Relay Disabled

At the conclusion of the replay attack testing, the error shown in Fig 5.12 was shown on Device 1's front display. While there is no indication of what caused the relay to become disabled, the User's Manual shows either operating system check failure or CPU error would cause this response during a self test. The corrective action recommended for this error is automatic restart of the relay. However, no method of communicating with the device proved effective, and the device required a power cycle to resume normal operation.

Figure 5.12: *Device 1 Disabled After Attack*

After resuming normal operation, a self test was conducted on the device to ensure no lasting errors persisted through the restart. As shown in Fig 5.13, the device restarted without any present errors and additional self tests showed the same results. There were no external

devices outside the testbed structure connected to the switch, so different configurations were tested for causing the error in the Replicating the Error section of chapter.

```
      IA      IB      IC      IN      VA      VB      VC      VS      MOF
OS   -0      -0      -0      -0      -0      -0      -0      -0      -0
OSH  -0       0      -0       0

      15V_PS  5V_REG  3.3V_REG
PS    14.97   5.00    3.26

      RAM     ROM     FPGA    EEPROM  FLASH   A/D     USB_BRD COM_BRD
      OK      OK      OK      OK      OK      OK      OK      OK

      TEMP    RTC     HMI
      40.9    OK      OK

 Relay Enabled
```

Figure 5.13: *Device 1 Post Disabled Restart*

5.3.3.1 Replicating Device 1 Relay Error

The first step taken to identify the cause of relay disabling was to verify the same operation occurred with different configurations of the device. The first option tested was different attack rates for both replay and state number attacks. The results from these attacks can be seen in Table 5.2. In order to identify reliable replication of the results, traffic rates starting at 100 Mbps down to 10 Mbps were executed in 5 minute intervals to determine the time the relay disabled for each attack. From the results, we can see that when the relay did disable, it occurred at 30 seconds into a sustained attack, and tripped every time the attack traffic was at a threshold traffic level. For replay attacks, this rate was 58 Mbps, while FDIA only required 53 Mbps.

Intuitively, these numbers make sense, as GOOSE message headers take 8 processing points and the boolean value we have mapped takes an additional 1 point. However, 58 Mbps is not exactly $\frac{9}{8}$ of 53 Mbps, and the disabling time is a consistent 30 seconds regardless of traffic rate over this threshold point. Through multiple tests of 57 and 52 Mbps for the respective attacks, the relay would occasionally disable at attacks greater than 60 seconds, but these results could not be reliably reproduced. There was also a concern that

Table 5.2: *Traffic Test: Disabling Relay*

Replay Attack		FDIA	
Traffic Rate	**Disabled Time**	**Traffic Rate**	**Disabled Time**
10 Mbps	Not Disabled	10 Mbps	Not Disabled
20 Mbps	Not Disabled	20 Mbps	Not Disabled
30 Mbps	Not Disabled	30 Mbps	Not Disabled
40 Mbps	Not Disabled	40 Mbps	Not Disabled
50 Mbps	Not Disabled	50 Mbps	Not Disabled
58 Mbps	30 Seconds	53 Mbps	30 Seconds
60 Mbps	30 Seconds	60 Mbps	30 Seconds
70 Mbps	30 Seconds	70 Mbps	30 Seconds
80 Mbps	30 Seconds	80 Mbps	30 Seconds
90 Mbps	30 Seconds	90 Mbps	30 Seconds
100 Mbps	30 Seconds	100 Mbps	30 Seconds

external factors may have caused these results, so the tests were rerun with different device configurations of output contacts, switches, and virtual bits.

The output contacts utilized for this testing were OUT101, OUT102, and OUT103. Each contact was tested with the same data traffic rate and length with the same results. The process was repeated for each switch utilized outlined in Chapter 4. The same process was then completed with the use of a different virtual bit in Device 1 with the same results. The only hard requirement found for duplicating the disabling result is the sustained traffic for 30 seconds. This led to us identifying two factors that affect the disabling of the relay as the attack data rate and sustained traffic time. If the attack traffic was not sustained for a full 30 seconds, and the data rate was not at or above the threshold, the relay would not be disabled. This was validated by running an identical attacks but pausing the traffic before a full 30 seconds completes. If the traffic was stopped short before the full 30 seconds had elapsed, the relay would not disable. The relation of attack traffic compared to sustained time is located in Fig 5.14.

The implications of this error could be detrimental to systems that employ multiple models of Device 1 with GOOSE subscriptions. The attack complexity required to completed this attack is low, requiring only a valid MAC of a GOOSE subscriber and the ability to

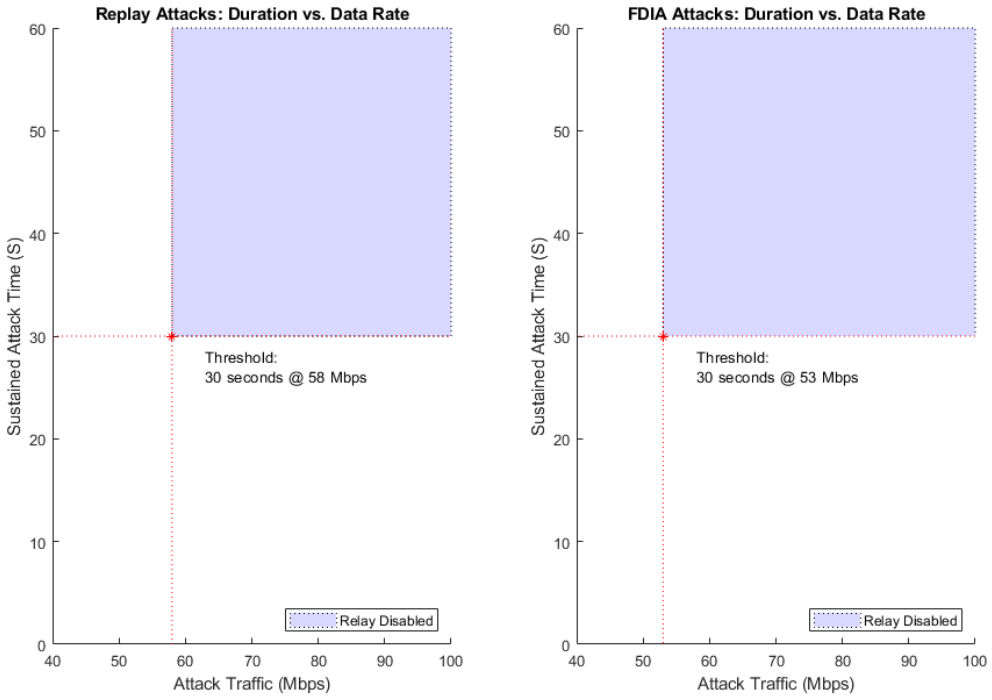

Figure 5.14: *Device 1 Attack Disable for Packet Size 125 Bytes*

replay that packet. The device is also disabled until a hard restart of the system, requiring a complete power cycling to return to normal operation. Properly configured networks may take into account cybersecurity methods that would mitigate the affects of these attacks, but a resolution to this problem that does not require physical access to the device should be investigated.

5.3.3.2 Possible Causes of Device 1 Error

The first cause may be directly related to the CPU usage, the two causes of the specified error could the the Operating System check failure or CPU error. Since there is an identifiable threshold of traffic that causes the error, the CPU may be maxed resources once the traffic rate is hit. From there, a sustained threshold for 30 seconds may be an internal timer, such as a Watchdog Timer, that throws the error itself. Device 1 runs on Device 1's own special Linux operating system, so identifying this as the cause would be infeasible from our current

attack setup.

The second possibility is the error is caused by firmware issues. There have been a number of firmware updates since Device 1 was purchased, and the current version of firmware on Device 1 is outdated. This is particularly interesting because the more recent releases of firmware mention deliberately crafted Ethernet traffic could cause either diagnostic or safety restarts. However, neither of these update notes mention the relay being disabled, and diagnostic restarts should generate an event report on the device itself. In future work, testing the device on upgraded firmware would be required to verify if the device has already been patched for this issue.

5.4 Device 2

Device 2 was the next device to be tested with the evaluation structure. Device 2 is marketed toward different functionality than Device 1 and is aimed toward medium voltage general purpose use. The configuration for Device 2 consisted of a single GOOSE subscriber, with a single virtual bit mapped to the output matrix for an output contact. The baseline results show a far wider range of response times in comparison to Device 1, with a much higher response time at just under 15 ms. The DoS and replay attacks showed similar behavior to Device 1, but Device 2 was still able to respond even after a full 6 hour run through all traffic levels of the attack. While the response time my not have been impressive, the robustness of the system was a strong point.

5.4.1 Device 2 baseline

From the baseline results shown in Fig 5.15, we can see that the responses from Device 2 are consistently slower than that of Device 1. While Device 2 is unable to respond in the same timeframe as Device 1, the response rate was 100% at the baseline testing with 0 dropped packets. There is an alternate configuration available for Device 2 that was not available

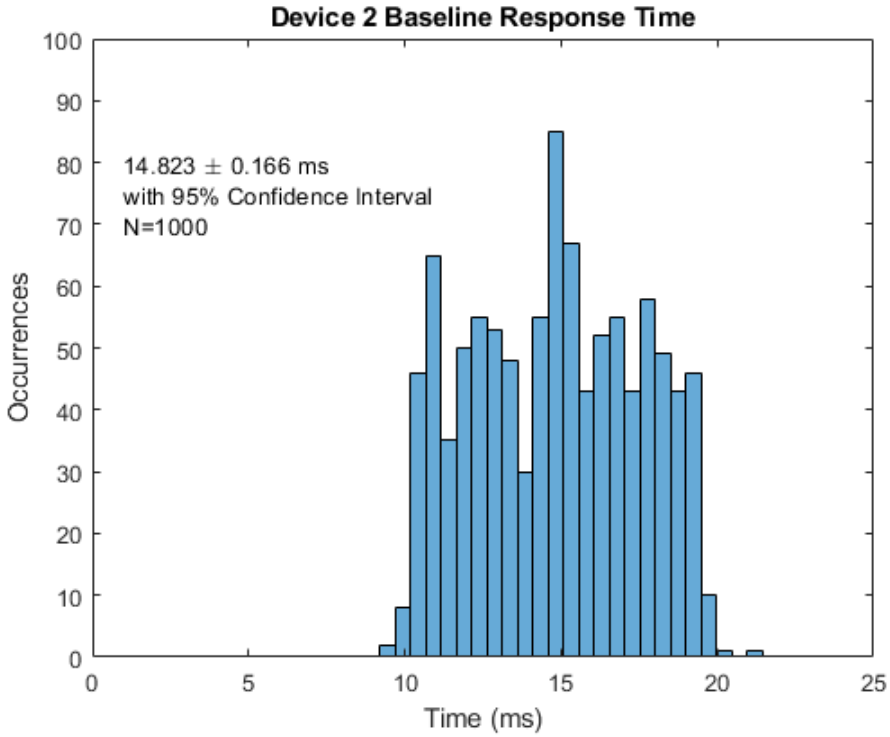

Figure 5.15: *Device 2 baseline Response Time*

on other devices tested, which is the ability to ignore the goID field of a subscription. This configuration was not tested in this Thesis, but examination of the configuration could lead to similar results and slow response time under attack traffic, even without properly encoded packets. If Device 2 isn't checking the goID of the packet, the main checks would need to be done at the MAC address. Without additional checks, attackers could broadcast to an existing GOOSE endpoint without having to properly craft a packet.

5.4.2 Device 2 DoS and Replay Attack

The timing results from the DoS matched closely to what was expected after examining Device 1 results. The timing of responses remained close to the initial baseline testing, staying around 15 ms per response time. However, as soon as Device 2 began processing replay attack packets, the response time grew by 30 times. Responses that normally took 15 ms now took an excess of 450 ms. However, Device 2 ran into no issues after the attacks had

stopped and would resume normal operation without any issues from the attacks previously. The dip at 60 Mbps is likely due to the small number of successful responses by the device, and it is expected to rise to above 450 ms with additional testing. The main takeaway from this execution is Device 2 is unable to maintain solid response times when attack packets must also be processed. This is in contrast to Device 1, which as a defined processing cycle and will discard any overflow. The responses here indicate Device 2 attempts to decode more packets but can't keep up with the processing time.

Device 2 may not have the same issues as Device 1 because it does not identify the operating system utilized. Attempts to establish a virtual com connection through the USB port gives limited information when compared to the protocol statistics shown above. In fact, typing in typical linux commands gives the error "This is not a UNIX machine". So without a proper investigation into the operating system in use, there would be little information to draw further conclusions on the response time.

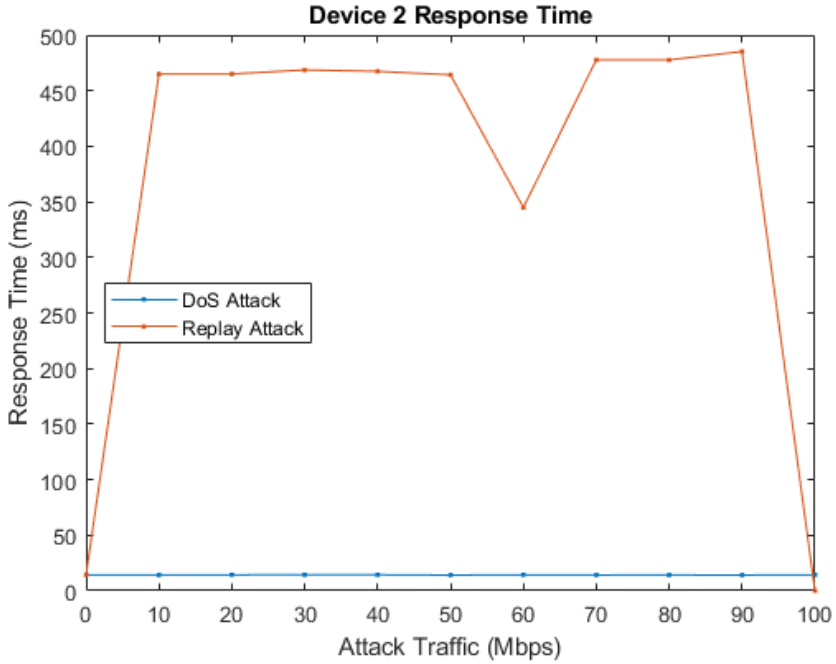

Figure 5.16: *Device 2 DoS vs Replay Response Time*

5.5 Raspberry Pi

The Raspberry Pi has the most unique results from the 3 devices that were tested in this thesis. This device is the only that doesn't activate a contact that can be used in higher voltage scenarios. In this situation, the only shared connections are a single digital I/O signal and a reference ground. Activating a contact can add multiple milliseconds, and in the case of Device 1, is rated for less than 5 ms. This means that the timing of the Raspberry Pi should not be directly compared to the previous devices.

The purpose of this comparison is to examine the implementation of the GOOSE protocol utilizing the libiec61850 library. There is also a consideration that should be made for the other processes the previous devices complete. For this testing the Raspberry Pi is running the Xorg desktop environment, but no other userspace processes than the GOOSE subscriber. The results from the Raspberry Pi were surprisingly positive. The use of a 1 Gigabit Ethernet port mitigated the reduction of response times found in the other tested devices, and the response time was remarkably stable up until the 100 Mbps attack traffic.

5.5.1 Raspberry Pi baseline

The baseline results from the Raspberry Pi show a tight grouping with a response time centering around .18 ms. There are more outliers in the graph than compared to the other devices however. The reason for this is the inability to guarantee resources in the Rasbian operating system. While most iterations will receive the required resources, the outliers are created when other processes require additional resources that limit the processing of the subscriber. This fact should be considered when using the Raspberry Pi for GOOSE applications.

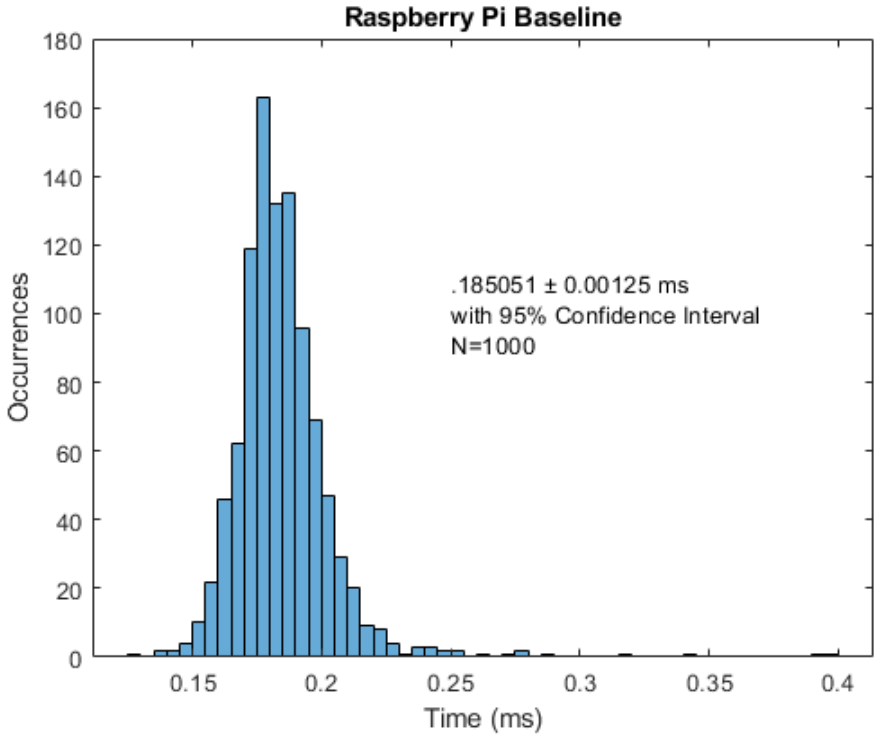

Figure 5.17: *Raspberry Pi baseline Response Time*

5.5.2 Raspberry Pi DoS and Replay Attack

Similar to the previous devices tested, the Raspberry Pi had no noticeable issues when processing packets while the DoS attack was executed. This is not a surprise when considering the attack was being executed on a Gigabit link. The replay attack resulted in similar results as the other devices. However, there were limited missed responses from this device than the previously tested. This is likely caused due to the limited processes run in the user space compared to time sensitive operations required of the previous devices.

5.6 FDIA - All Devices

The FDIA executed in this thesis was a state number attack. The goal of this attack is to force the IED to miss valid responses from the publishing agent by producing valid packets with a higher state number. An example of FDIA traffic can be seen in Fig 5.19. When

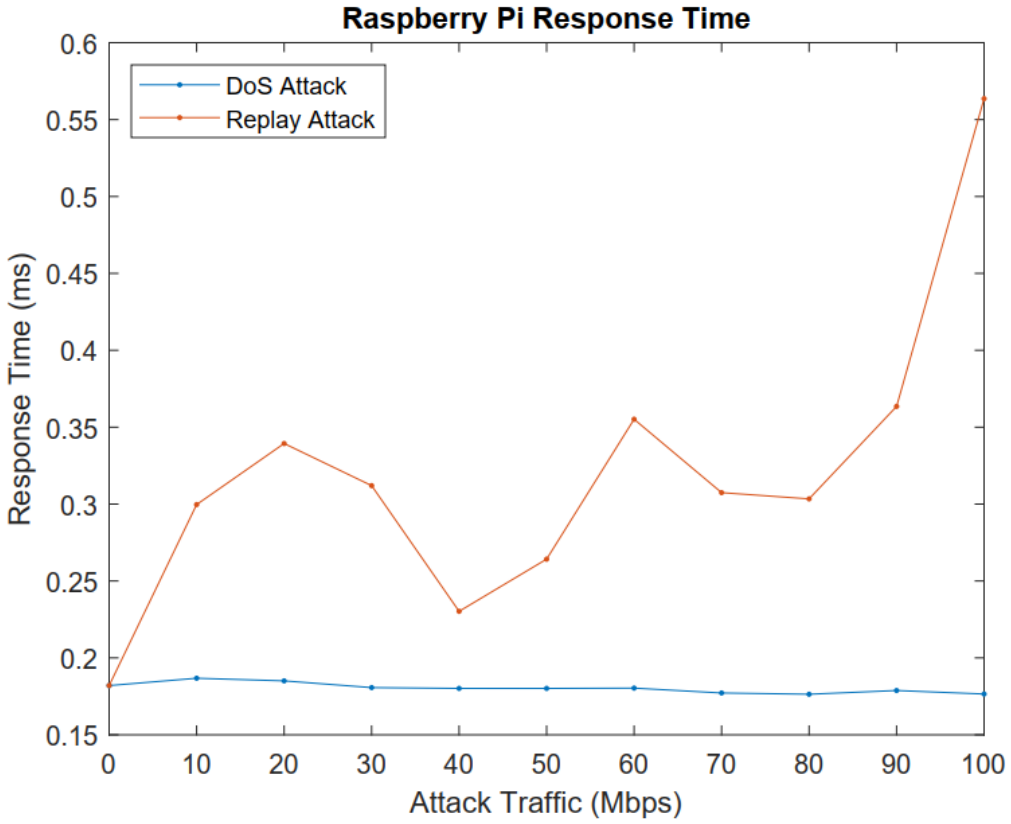

Figure 5.18: *Raspberry Pi Response Times*

executing this attack, each device was unable to respond to valid packets. The cause of this is the limited number of packets required to outpace the state numbers found in regular GOOSE traffic.

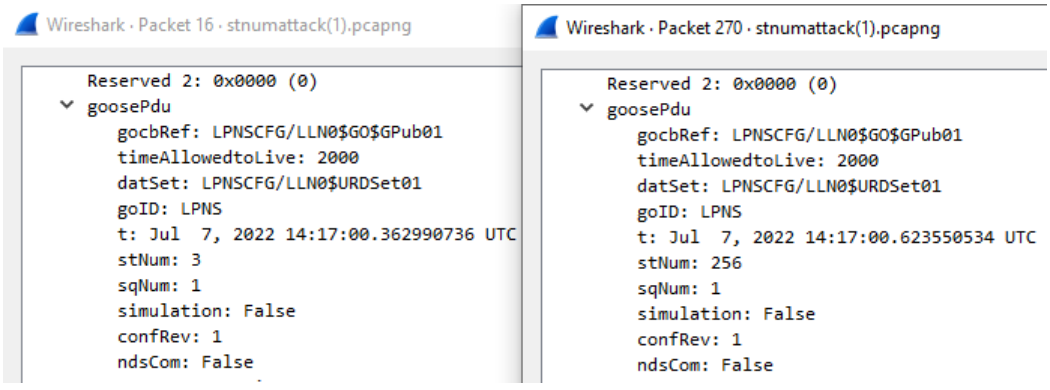

Figure 5.19: *State Number Attack Wireshark Decode*

An example of this is the execution of an FDIA with 1 Mbps data rate. Even at the lowest supported data rate for the testbed, each device was still unable to respond to valid packets. In Fig 5.20, we can see the result of executing the state number attack at 1 Mbps for 1 second results in a state number of 2011. This state number can be increased to a value 2^{32}, which would theoretically prevent standard GOOSE messages from ever being published.

In Chapter 4, the testing outline indicated that the FDIA would take 10k cycles, for traffic levels from 10-100 Mbps. However, when testing was completed with the 100 Mbps traffic, no device was capable of responding. To see if any other attacks would give responses, the 10 Mbps attack was run next. As we anticipated, there were no responses from any devices throughout the test. These results were anticipated since we are using 125 byte packet length. To achieve a data rate of 10 Mbps, we would need to send a total of 10k packets per second to achieve the required data rate. Seeing 10k events on the standard GOOSE protocol traffic would take an extended period of time, and we've tricked the devices to expect that state number in a single second.

GOOSE Receive Status

MultiCastAddr	Ptag:Vlan	AppID	StNum	SqNum	TTL
01-0C-CD-01-00-03	0:1	3	2011	0	2000

Figure 5.20: *State Number Attack Device 1 Statistics*

While there are no expected responses during the attack execution, once the time to live expires on the last packet sent, both Device 1 and Device 2 were able to respond to standard traffic. By implementing the protocol in this way, the devices are able to recover from an attack without external intervention. However, our current implementation on the Raspberry Pi required a restart. This is due to the comparison only checking valid packet time to live and state number. If there was a timeout implemented for the time to live, the Raspberry Pi would be able to respond as well. This is a perk of the open source implementation in libiec61850, as the functionality can be altered to meet application needs.

5.7 Device Comparison of Attack Responses

From the previous sections, the response times for Device 1, Device 2, and the Raspberry Pi were examined for DoS and replay attacks. One aspect that has not been examined yet is the successful response percentage of each device for the replay attacks in Fig 5.21. The DoS was not plotted in this graph because each device successfully responded to every packet.

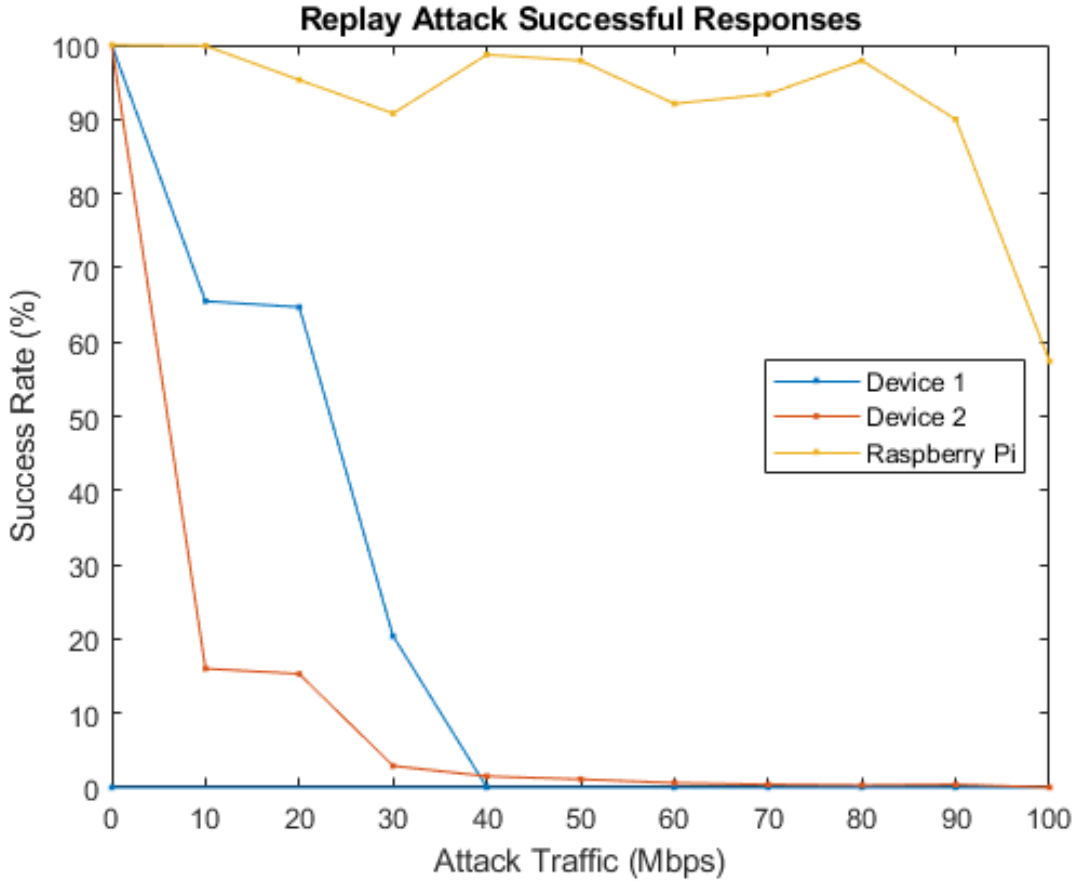

Figure 5.21: *Replay Attack Successful Response Rates*

We can see that the Raspberry Pi has the highest response rate, followed by Device 1 and Device 2. However, the Raspberry Pi did see a large drop in accuracy at 100 Mbps, which could be the limit to raw processing the device can handle. This port was also not limited to the 100 Mbps link like the other devices, so switch queuing could have had an effect on the responses. An interesting note is both Device 1 and Device 2 show significant reduction in response at 10 and 20 Mbps, but the response percentages level out rather than decreasing at

each step as initially expected. This could be common behavior between the two devices that could indicate a cyberattack. For Device 1, the accuracy remains comfortably higher than Device 2, until the relay was disabled from the attack. However, at that point Device 2 was only responding to 10% of events.

5.8 Device GOOSE Implementation Analysis

The three different devices tested presented different protocol implementations that have a variety of benefits and downfalls. In the case of Device 1, the device has fast response times and reasonable successful response rate in the response to attacks. However, the major downfall is encountered when the relay is disabled by cyberattacks. To this point, we were unable to restart the device from an encountered fault without physically power cycling the device.

This is contrasted by Device 2, which has much slower response times and responds to considerably less messages when an attack is present. However, Device 2 was subjected to continuous testing without any errors occurring. Even with resetting of publishing traffic, the device was able to properly respond to valid GOOSE packets. The Raspberry Pi running libiec618520 was a pleasant surprise by providing a perfectly reasonable implementation of the GOOSE protocol.

CHAPTER 6

Conclusion and Future Work

In this thesis, we created an effective, flexible, and practical testbed for OT protocols. Based on two Raspberry Pis, the testbed allows for cyberattack generation, standard traffic generation, and monitoring signal lines of a DUT. As part of the testbed validation, variable traffic rates were tested for cyberattacks, and a reliable traffic generation between 10-120 Mbps was achieved with packets as small as 125 bytes. Through the use of traffic bursts and a timing algorithm, we were able to reliably send 120k packets per second. While this testbed is capable of evaluating multiple protocols, we examined and evaluated different GOOSE protocol implementations on two devices from reputable manufacturers and a Raspberry Pi running libiec61850.

Through creation of this testbed generating valid GOOSE communications and cyberattack traffic, each device's protocol implementation was evaluated. Through the generation of DoS, Replay, and FDIA attacks, the benefits and shortcomings of each implementation's performance were outlined. Different response cases were found for Replay and FDIA attack. Device 1 was reliable in response time, maintaining less than 7ms responses. However, the relay disabled with a CPU error at 58 Mbps replay attack and 53 Mbps state number attack. Due to this higher data rates are extremely dangerous for implementations and require physical intervention to resume normal operation.

Device 2 had opposite results, with much higher response times averaging 15 ms, but a

resilience to all evaluated attacks. The downfall of this implementation was the responses increasing to almost 500ms at attack rates as low as 10 Mbps. The Raspberry Pi running libiec61850 provided solid performance against DoS and replay attacks, but was unable to recover after a state number attack was implemented. However, response times almost tripled and successful responses were halved at 100 Mbps. Considering this device was equipped with a 1 Gigabit Ethernet adapter, there are processing constraints that prevent the device from keeping up with attack traffic.

Our future research will focus on the disabling of the Device 1 and expanding the operation of the hardware agent to perform a large variety of attacks on different protocols. In regards to the Device 1, investigation of more recent firmware releases will determine whether the device vulnerability still exists. This work can be expanded to different devices in the Device 1 architecture to identify whether the vulnerability exists across multiple devices or just the Device 1. The testbed will also implement different OT protocols, expanding from energy sector protocols to protocols in building automation and manufacturing. The implementation of additional protocols will also allow for the testing to be evaluated on different devices.

Funding Support

This research was partially funded by the US. Department of Energy through a subcontract from Oak Ridge National Laboratory, project No. 4000175929. This project was also partially funded by Nebraska Center for Energy Sciences Research Cycle 16 Grant 20-706.

Bibliography

[1] M. Boeding, K. Boswell, M. Hempel, H. Sharif, J. Lopez Jr., and K. Perumalla, "Survey of cybersecurity governance, threats, and countermeasures for the power grid," in *ACM Computing Surveys [Under Review]*, Submitted March 31, 2022.

[2] S. Rao, G. V. Chatrapathi, and T. Yashashwini, "Ethernet/ip + fdi: Value in process automation," in *2017 2nd International Conference On Emerging Computation and Information Technologies (ICECIT)*, 2017, pp. 1–5. DOI: 10.1109/ICECIT.2017.8453324.

[3] H. Jian-Cang, "Research on bacnet building controller based on arm9 and embedded linux," in *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 2318–2324. DOI: 10.1109/CCDC.2018.8407513.

[4] "Modbus application protocol specification v1.1b3 available [online] : Https://modbus.org/docs/modbus_application_protocol_v1_1b3.pdf," pp. 1–50, 2012.

[5] "Ieee standard for electric power systems communications-distributed network protocol (dnp3)," *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, 2012. DOI: 10.1109/IEEESTD.2012.6327578.

[6] A. Yamane, T. Rangineed, L.-A. Gregoire, S. Q. Ali, J.-N. Paquin, and J. Belanger, "Multi-fpga solution for large power systems and microgrids real time simulation," in *2019 IEEE Conference on Power Electronics and Renewable Energy (CPERE)*, 2019, pp. 367–370. DOI: 10.1109/CPERE45374.2019.8980066.

[7] J. Noce, Y. Lopes, N. C. Fernandes, C. V. N. Albuquerque, and D. C. Muchaluat-Saade, "Identifying vulnerabilities in smart gric communication networks of electrical substations using geese 2.0," in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, 2017, pp. 111–116. DOI: 10.1109/ISIE.2017.8001232.

[8] "Communication networks and systems for power utility automation – part 7-4: Basic communication structure – compatible logical node classes and data object classes," International Electrotechnical Commission, Standard, Feb. 2020.

[9] "Communication networks and systems for power utility automation – part 6: Configuration description language for communication in electrical substations related to ieds," International Electrotechnical Commission, Standard, Dec. 2009.

[10] W. Wimmer, A. Baden, and Switzerland, "Iec 61850 scl-more than interoperable data exchange between engineering tools," Jan. 2005.

[11] "Ieee standard for local and metropolitan area network–bridges and bridged networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, 2018. DOI: 10.1109/IEEESTD.2018.8403927.

[12] "Libiec61850: Open source library for iec 61850, [online] available: http://libiec61850.com/libiec61850/," 2016.

[13] "Communication networks and systems for power utility automation - part 7-1: Basic communication structure - principles and models," International Electrotechnical Commission, Standard, Aug. 2011.

[14] R. R. S, R. R, M. Moharir, and S. G, "Scapy- a powerful interactive packet manipulation program," in *2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS)*, 2018, pp. 1–5. DOI: 10.1109/ICNEWS.2018.8903954.

[15] "Wireshark: Network protocol analyzer, [online] available: https://www.wireshark.org/," 2022.

[16] S. Ashraf, M. H. Shawon, H. M. Khalid, and S. M. Muyeen, "Denial-of-service attack on iec 61850-based substation automation system: A crucial cyber threat towards smart substation pathways," *Sensors*, vol. 21, no. 19, 2021, ISSN: 1424-8220. DOI: 10.3390/s21196415. [Online]. Available: https://www.mdpi.com/1424-8220/21/19/6415.

[17] J. Hoyos, M. Dehus, and T. X. Brown, "Exploiting the goose protocol: A practical attack on cyber-infrastructure," in *2012 IEEE Globecom Workshops*, 2012, pp. 1508–1513. DOI: 10.1109/GLOCOMW.2012.6477809.

[18] F. Zhang, M. Mahler, and Q. Li, "Flooding attacks against secure time-critical communications in the power grid," in *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2017, pp. 449–454. DOI: 10.1109/SmartGridComm.2017.8340726.

[19] J. G. Wright and S. D. Wolthusen, "Stealthy injection attacks against iec61850's goose messaging service," *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pp. 1–6, 2018.

[20] S. E. L. Mauricio Gadelha da Silveira Paulo Henrique Franco, "Iec 61850 network cybersecurity: Mitigating goose message vulnerabilities," in *6th Annual PAC World Americas Conference*, 2019.

[21] S. S. M. Reshikeshan and M. S. Illindala, "Systematically encoded polynomial codes to detect and mitigate high-status-number attacks in inter-substation goose communications," in *2020 IEEE Industry Applications Society Annual Meeting*, 2020, pp. 1–7. DOI: 10.1109/IAS44978.2020.9334776.

[22] K. Boakye-Boateng and A. H. Lashkari, "Securing goose: The return of one-time pads," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8. DOI: 10.1109/CCST.2019.8888435.

[23] A. Bohara, J. Ros-Giralt, G. Elbez, A. Valdes, K. Nahrstedt, and W. H. Sanders, "Ed4gap: Efficient detection for goose-based poisoning attacks on iec 61850 substations," in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020, pp. 1–7. DOI: 10.1109/SmartGridComm47815.2020.9303015.

[24] H. Reda, B. Ray, P. Peidaee, *et al.*, "Vulnerability and impact analysis of the iec 61850 goose protocol in the smart grid," *Sensors*, vol. 21, p. 1554, Feb. 2021. DOI: 10.3390/s21041554.

[25] S. M. S. Hussain, T. S. Ustun, and A. Kalam, "A review of iec 62351 security mechanisms for iec 61850 message exchanges," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5643–5654, 2020. DOI: 10.1109/TII.2019.2956734.

[26] A. A. Memon and K. Kauhaniemi, "Real-time hardware-in-the-loop testing of iec 61850 goose-based logically selective adaptive protection of ac microgrid," *IEEE Access*, vol. 9, pp. 154 612–154 639, 2021. DOI: 10.1109/ACCESS.2021.3128370.

[27] P. Jamborsalamati, A. Sadu, F. Ponci, and A. Monti, "A flexible hil testing platform for performance evaluation of iec 61850-based protection schemes," in *2016 IEEE Power and Energy Society General Meeting (PESGM)*, 2016, pp. 1–5. DOI: 10.1109/PESGM.2016.7741721.

[28] E. Piesciorovsky, "Real-time simulator with protection system in-the-loop - instruction manual," Jul. 2017. DOI: 10.13140/RG.2.2.11707.31529.

[29] "Raspberry pi 4b technical specifications available [online] : Https://www.raspberrypi .com/products/raspberry-pi-4-model-b/specifications/," *Raspberry Pi*, 2022.

[30] "Hp spectre x360 laptops, available [online] : Https://www.hp.com/us-en/shop/slp/spectre-family/hp-spectre-x-360," *Hewlett-Packard*, 2022.

[31] "Data sheet | gs716tv3, gs724tv4, gs748tv5, available [online] : Https://www.downloads.netgear.com/files/gdc/datasheet/en/gs716tv3-gs724tv4-gs748tv5.pdf," *Netgear*, 2022.

[32] "Cisco 110 series unmanaged switches data sheet available [online] : Https://www.cisco.com/c/en/us/products/collateral/switches/110-series-unmanaged-switches/datasheet-c78-734450.html," *Cisco*, 2022.

[33] "Wiringpi: Gpio interface library for the raspberry pi available: [online] http://wiringpi.com/," 2019.