



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**THE OPERATION AND CYBER SECURITY OF
INDUSTRIAL CONTROL SYSTEMS AND INDUSTRIAL
COMMUNICATION PROTOCOLS: AN EXPLORATORY
APPROACH**

by

Anastasios Antoniou

September 2016

Thesis Advisor:

Co-Advisor:

Second Reader:

Weilian Su

Christopher Eagle

Tri T. Ha

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2016		3. REPORT TYPE AND DATES COVERED Master's thesis
4. TITLE AND SUBTITLE THE OPERATION AND CYBER SECURITY OF INDUSTRIAL CONTROL SYSTEMS AND INDUSTRIAL COMMUNICATION PROTOCOLS: AN EXPLORATORY APPROACH			5. FUNDING NUMBERS	
6. AUTHOR(S) Anastasios Antoniou				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) In this thesis, an integrated examination of the operational principles and security aspects of industrial control systems and industrial communication protocols to discover exploitable vulnerabilities are investigated. This study primarily focuses on industrial automation equipment that utilizes industrial Ethernet. Of the many protocols developed by commercial vendors, only PROFINET, EtherNet/IP, EtherCAT, and Modbus TCP were researched. The investigation is based on observation of the principles of operation of these protocols and the functionality of industrial control systems. For this investigation, an environment that emulates a small-scale industrial testbed was built utilizing commercial-off-the-shelf equipment. A multi-adaptive software tool, HERMES, was proposed and developed in the C programming language to allow active injection of malformed protocol data units. The result is an integrated platform that can be used for testing several exploitation tools, observing PROFINET communications, and testing the industrial equipment setup under injection of specially crafted packets.				
14. SUBJECT TERMS Industrial communication protocol, industrial control system, SCADA, EtherCAT, PROFINET, Modbus/TCP, EtherNet/IP, cyber security, libnet C library, libpcap C library.			15. NUMBER OF PAGES 139	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**THE OPERATION AND CYBER SECURITY OF INDUSTRIAL CONTROL
SYSTEMS AND INDUSTRIAL COMMUNICATION PROTOCOLS:
AN EXPLORATORY APPROACH**

Anastasios Antoniou
Lieutenant Junior Grade (Eng), Hellenic Navy
B.S., Hellenic Naval Academy, 2008

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL
September 2016**

Approved by:

Weilian Su
Thesis Advisor

Christopher Eagle
Co-Advisor

Tri T. Ha
Second Reader

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, an integrated examination of the operational principles and security aspects of industrial control systems and industrial communication protocols to discover exploitable vulnerabilities are investigated. This study primarily focuses on industrial automation equipment that utilizes industrial Ethernet. Of the many protocols developed by commercial vendors, only PROFINET, EtherNet/IP, EtherCAT, and Modbus TCP were researched.

The investigation is based on observation of the principles of operation of these protocols and the functionality of industrial control systems. For this investigation, an environment that emulates a small-scale industrial testbed was built utilizing commercial off-the-shelf equipment. A multi-adaptive software tool, HERMES, was proposed and developed in the C programming language to allow active injection of malformed protocol data units. The result is an integrated platform that can be used for testing several exploitation tools, observing PROFINET communications, and testing the industrial equipment setup under injection of specially crafted packets.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	BACKGROUND	1
B.	OBJECTIVE	3
C.	RELATED WORK	3
D.	THESIS ORGANIZATION.....	4
II.	INTRODUCTION TO INDUSTRIAL CONTROL SYSTEMS.....	5
III.	COMMUNICATION PRINCIPLES AND REQUIREMENTS IN AN INDUSTRIAL ENVIRONMENT.....	9
A.	EVOLUTION PROCESS AND CHARACTERISTICS	9
B.	THE INDUSTRIAL ETHERNET	13
IV.	ETHERNET-BASED INDUSTRIAL COMMUNICATION PROTOCOLS	19
A.	PROFINET	19
B.	ETHERCAT	25
C.	ETHERNET/IP	32
D.	MODBUS/TCP	36
E.	SECURITY CONCERNS	39
V.	INDUSTRIAL TESTBED IMPLEMENTATION.....	43
A.	THOUGHTS AND CONCERNS.....	43
B.	TESTBED SET UP	45
C.	HARDWARE DISASSEMBLY OF THE PLCs	63
VI.	SOFTWARE TOOL FOR FUZZING AND TESTING	67
A.	SOCKET PROGRAMMING, LIBNET, AND LIBPCAP LIBRARIES.....	67
B.	HERMES: THE INDUSTRIAL ETHERNET PACKET INJECTOR.....	74
C.	HERMES SOFTWARE DESIGN DESCRIPTION	75
D.	HERMES CAPABILITIES.....	79
E.	TESTING HERMES AGAINST TESTBED DEVICES	81
VII.	CONCLUSIONS AND FUTURE WORKS.....	87

APPENDIX. SUPPLEMENTAL MATERIAL	91
LIST OF REFERENCES.....	113
INITIAL DISTRIBUTION LIST	119

LIST OF FIGURES

Figure 1.	ICS Operation Functional Diagram. Source: [9].	6
Figure 2.	Classes of Industrial Communication Traffic. Adapted from [11].	10
Figure 3.	Typical Network Topologies at Fieldbus Level. Source: [11].	11
Figure 4.	Medium Access Control Strategies in Fieldbus and Protocol Examples. Source: [11].	12
Figure 5.	The OSI Conceptual Communication Model. Adapted from [15].	14
Figure 6.	Comparison of Layering between OSI and TCP/IP Suite	14
Figure 7.	Encapsulation Process Illustrated. Source: [17].	15
Figure 8.	Industrial Communication Network Market Share. Source: [20].	17
Figure 9.	PROFINET CBA and PROFINET IO Illustrated. Adapted from [21].	20
Figure 10.	PROFINET Communication Channel Partitioning for RT and IRT. Source: [23].	22
Figure 11.	Performance Comparison between RT, IRT, and non-RT Traffic. Source: [24].	22
Figure 12.	Protocol Implementation Stack in PROFINET. Adapted from [25].	23
Figure 13.	An EtherCAT Frame Encapsulated inside an Ethernet Frame. Adapted from [29].	27
Figure 14.	Memory Mapping of Process Image in the Master Device. Source: [28].	28
Figure 15.	EtherCAT Encapsulation Process for TCP/UDP. Adapted from [31].	30
Figure 16.	EtherCAT Application Layer Encapsulation in Ethernet IEEE 802.3. Source: [28].	31
Figure 17.	Ethernet/IP and CIP Protocol Stack Implementation. Source: [32].	32
Figure 18.	EtherNet/IP and CIP Protocols Organization. Adapted from [35].	35
Figure 19.	Modbus/TCP Master-Slave Communication Model. Source: [19].	37
Figure 20.	The Modbus Object Data Model in Server/Slave Device. Source: [37].	38
Figure 21.	Simple OR Boolean Logic Operation in a LAD Network	47
Figure 22.	Simple AND Boolean Logic Operation in a LAD Network	47
Figure 23.	Function Block for digital-to-analog conversion, followed by Function Block for Signal Scaling	48
Figure 24.	Screen 1 of HMI for a Diesel Generator and its Auxiliary Machinery	49

Figure 25.	Screen 2 of HMI for Monitoring Operational Parameters	50
Figure 26.	Screen 3 of HMI for Monitoring and Control of Fuel System	51
Figure 27.	Network Topology Using TIA Portal	52
Figure 28.	Topology for Capturing and Observation of Network Traffic.....	53
Figure 29.	Topology of Communication Paths	53
Figure 30.	Established Communication between PLC Controllers and HMI	55
Figure 31.	Established Communication between PLCs over PROFINET RT/IRT	55
Figure 32.	Two Subsequent PROFINET PDUs as appeared in Wireshark.....	57
Figure 33.	PDU Sequence for Reestablishment of PROFINET Communication between PLC 1 and PLC 2.....	59
Figure 34.	PLC s7-1200 View with Front Cover Removed.....	65
Figure 35.	System Call socket() Control over PDU Headers. Adapted from [42].....	69
Figure 36.	Process Flow for Custom-made PDU using C Libnet Utility Functions.....	72
Figure 37.	Process Flow for Initiating PDU Capturing with C Libpcap	73
Figure 38.	HERMES Structural and Functionality Description Block Diagram	78
Figure 39.	PROFINET over DCE/RPC Connectionless Protocol Packets	83
Figure 40.	PROFINET-IO Alarm Message Packet.....	84
Figure 41.	Testbed Vertical and Horizontal Expansion	88
Figure 42.	PROFINET IO Encapsulation Process for RT and IRT Traffic on Top of IEEE 802.3 Ethernet Frames.....	92
Figure 43.	PROFINET IO Encapsulation Process for RT and IRT Traffic on Top of IEEE 802.1q VLAN Frame.....	93
Figure 44.	PROFINET IO PTPC Protocol as Part of the Payload Encapsulated in the PROFINET RT Payload	94
Figure 45.	PROFINET IO DCP Protocol as Part of the Payload Encapsulated in the PROFINET RT Payload.....	95
Figure 46.	MRP Protocol as Part of the Payload Incorporated in the IEEE 802.3 Ethernet Frame.....	96
Figure 47.	PROFINET IO Protocol on Top of IP/UDP/DCE-RPC Protocols of RT traffic.....	97
Figure 48.	PROFINET Communication Protocol Encapsulation Process for Non-RT Traffic	98

Figure 49.	PROFINET Communication Protocol Encapsulation Process for Non-RT Traffic	99
Figure 50.	EtherCAT Communication Protocol Encapsulation Process for Non-RT Traffic. Adapted from [28]–[31], [44]–[47].	100
Figure 51.	EtherCAT Communication Protocol Encapsulation Process Non-RT Traffic. Adapted from [28]–[31], [44]–[47].	101
Figure 52.	EtherCAT Communication Protocol Encapsulation Process for EtherCAT Device Protocol. Adapted from [28]–[31], [44]–[47].	102
Figure 53.	EtherCAT Communication Protocol Encapsulation Process for EtherCAT Mailbox. Adapted from [28]–[31], [44]–[47].	103
Figure 54.	EtherCAT Communication Protocol Encapsulation Process for EoE. Adapted from [28]–[31], [44]–[47].....	104
Figure 55.	EtherCAT Communication Protocol Encapsulation Process for CoE. Adapted from [28]–[31], [44]–[47].....	105
Figure 56.	EtherCAT Communication Protocol Encapsulation Process for SoE. Adapted from [28]–[31], [44]–[47].....	106
Figure 57.	EtherCAT Communication Protocol Encapsulation Process for PDO. Adapted from [28]–[31], [44]–[47].....	107
Figure 58.	EtherNet/IP Encapsulation Process.....	108
Figure 59.	Modbus/TCP Communication Protocol Encapsulation Process for Non-RT Traffic	109
Figure 60.	PROFINET IO Sequence of PDUs for Establishment of Communication between PLC 1 and PLC 2	110
Figure 61.	PROFINET-IO Sequence of PDUs for Establishment of Communication between PLC 1 and PLC 2	111

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Transportation Layer Ports and Associated Services in PLC 1 and PLC 2	60
Table 2.	Transportation Layer Ports and Associated Services in the HMI.....	60
Table 3.	Transportation Layer Ports and Associated Services in the Network Switch	61
Table 4.	Equipment's Description Utilized for Testbed	91

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
ARP	Address Resolution Protocol
CIP	Common Industrial Protocol
CoE	CAN over EtherCAT
COTP	Connection Oriented Transfer Protocol
CSMA	Carrier-Sense Multiple Access
CSMA/CA	Carrier-Sense Multiple Access / Collision Avoidance
CSMA/CD	Carrier-Sense Multiple Access / Collision Detection
DCE/RPC	Distributed Control Environment / Remote Procedure Call
DHCP	Dynamic Host Configuration Protocol
EoE	Ethernet over EtherCAT
EtherCAT	Ethernet for Control Automation Technology
ESC	EtherCAT Slave Controller
FBD	Function Block Diagram
FoE	File over EtherCAT
FMMU	Fieldbus Memory Management Unit
FPGA	Field-Programmable Gate Array
GPIO	General Purpose Input-Output
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICS	Industrial Control System
IEC	International Electrotechnical Commission
IECPPI	Industrial Ethernet Communication Protocol Packet Injector
IED	Intelligent Electronic Device
IECPPI	Industrial Ethernet Communication Protocol Packet Injector
IP	Internet Protocol
IRT	Isochronous Real Time
ISO/OSI	Internal Organizing Standardization/Open Systems Interconnection
IT	Information Technology

LAD	Ladder Logic
LLDP	Link Layer Discovery Protocol
MAC	Media Access Control
MBAP	ModBus Application Protocol
MITM	Man-in-the-middle Attack
OB	Organization Block
ODVA	Open DeviceNet Association
OSI	Open Systems Interconnection model
PDU	Protocol Data Unit
PLC	Programmable Logic Controllers
PN-CBA	PROFINET Component Base Automation
PN-DCP	PROFINET Discovery and Configuration Protocol
PN-MRP	PROFINET Media Redundancy Protocol
PN-MRRT	PROFINET Media Redundancy for PROFINET RT
PN-IRT	PROFINET Isochronous Real Time
PN-PTCP	PROFINET Precision Time Control Protocol
PN-RT	PROFINET Real Time
PROFINET	Process Field Net
RTU	Remote Terminal Units
RT	Real Time
RTOS	Real-Time Operating System
SCL	Structured Control Language
SFC	Sequential Function Chart
SNMP	Simple Network Management Protocol
SoE	Servo Drive Profile (IEC 61491-7-204) over EtherCAT (SoE)
STL	Statement List
STP	Spanning Tree Protocol
TDMA	Time-Division Multiple Access
UDP	User Datagram Protocol
VoE	Vendor Specific profile over EtherCAT

ACKNOWLEDGMENTS

I would like to express my deep appreciation to my advisor, Dr. Weilian Su, for the equipment he provided for my experiments, and for his guidance and support throughout the entire thesis research. I would also like to express my thanks to my Co-Advisor, Christopher Eagle, who as an instructor and thesis advisor provided me the required programming skills to meet the difficult programming tasks that I faced during my research. Additionally, I am grateful to all my instructors for their outstanding tutoring.

I wish to express my great gratitude to my country, Hellas, and the Hellenic Navy for providing me the opportunity of this educational experience. Finally, I want to express my infinite and deep love to my parents, Konstantinos and Maria, who have always stood by my side, providing me motivation, help, encouragement, and support before, during, and after my deployment to the Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Supervisory control and data acquisition systems are deployed in a great variety of facilities and infrastructures. These facilities might be heavy industries, electrical power production and distribution plants, a jail monitoring system, a waste and clear water management system, or a liquid fuel distribution system. Maintaining security against modern forms of cyber warfare is of great importance to these facilities, since an interruption in their services as a result of a cyber-attack can cause significant damage to the equipment used and have a severe impact.

Stuxnet, which targeted a nuclear power facility, is a recent example of an attack against an industrial control system. According to [1]–[3], Stuxnet is strongly believed to have been deliberately deployed by someone who was physically present in the facility. Stuxnet took advantage of several vulnerabilities at the same time. These vulnerabilities were located primarily in the operating system and the software platform running on top of the operating system that was used for monitoring the facility and configuring its industrial controllers. After this exploitation, Stuxnet was able to propagate itself as any other worm. Additionally, it was able to hide its presence by using a rootkit component. In the end, Stuxnet managed to hijack the session between the Programmable Logic Controllers (PLC) and the Windows-based computer system, the latter being used for monitoring, control of the process as well as parameterization of PLCs. The resulting man-in-the-middle attack allowed Stuxnet to send compromised data to the monitoring stations; it performed a replay attack, which injected commands into the programmable logic controllers that controlled the speed of some motors.

Of the several vulnerabilities that Stuxnet exploited, the first few resided in the Windows operating system itself. The next vulnerability was on the PLC, which was not able to verify the integrity and authenticity of the received data. Last, but not least, there were the vulnerabilities related to the deployment and the design of the industrial process. Specifically, Stuxnet took advantage of a localized environment, which was assumed to

be sanitized and protected. In this environment, the communication between the devices is based on a trusted relationship (i.e., all devices receive and send data without any encryption scheme or any kind of validation). This trusted relationship, however, proved to be a vulnerability. The communication between devices involved in this attack utilized the Profibus protocol; this protocol was unable to provide the most fundamental principles of information security, which are confidentiality, integrity, and availability. According to [2], Stuxnet acted like a lurking bug. It waited for a specific messaging pattern over Profibus between the Siemens WinCC software and the programmable logic devices. Based on this fact, Stuxnet was able to violate the confidentiality principle, watch and subvert the traffic, create an artificial picture of the process in the nuclear facility, and freely manipulate the data that was exchanged between the operator stations and programmable logic controllers.

Stuxnet is an historic example, marking an important breakthrough in cyber security considerations related to industrial control systems. From this example, we conclude that communication protocols play an important role in operation of industrial control systems. Communication protocols enable operators and engineers to interact with automated systems and are, therefore, the most likely vector for any kind of attack. Industrial communication protocols are the language that machines understand. If we want to make industrial control system (ICSs) secure, we have to enhance and improve the capabilities of these protocols and ensure they meet the requirements related to information security. As a result, it is important to explore the operating principles, the behavior, and the vulnerabilities of protocols that are already deployed. Another related issue is that ICSs are increasingly being transformed from isolated platforms to interconnected systems that are able to communicate over the Internet. To enable this transition, industries have deployed a new family of protocols known as Industrial Ethernet. Many issues related to the security of industrial control systems emerge with this transition, including new attack vectors. Are these protocols able to protect critical and confidential information? Can these protocols resist several kinds of replay, man-in-the-middle, and session hijacking attacks exploited by Stuxnet? How can we validate

their secure operation? These concerns are some of the questions that we attempt to answer and investigate in this thesis.

B. OBJECTIVE

The research issues addressed in this thesis focus on the principles of operation of some communication protocols that belong to the Industrial Ethernet family. The investigation is mainly focused on the structure, type, and classes of messages exchanged by devices. These industrial devices communicate using Industrial Ethernet technologies such as PROFINET, EtherCAT, Ethernet/IP, and Modbus TCP. Additionally, the possibility of developing a multi-adaptive platform that can test security capabilities of industrial communication protocols is examined. Finally, a testbed based on PROFINET communication and Siemens industrial equipment is built, and a software tool (a crafted packet injector) is proposed and developed.

C. RELATED WORK

Several Ethernet-based industrial communication protocol vulnerabilities have been discovered [4]–[7]. Most of them are closely related to injection of malformed Protocol Data Units (PDU). Many of them cause a denial-of-service against the industrial control device, while many others result in remote execution of unauthorized commands. In both cases, it is obvious that each new device that is designed and deployed in an industrial environment is vulnerable due to misconfiguration and incorrect design and implementation of its associated communications protocols.

It seems that no sure technique exists to discover vulnerabilities like those previously described. The proposed solution in this thesis is that early detection and mitigation of these vulnerabilities depends heavily on the observation of the behavior of the protocol through actual deployment. Additionally, to verify the system behavior, the user must be able to inject specific inputs into the deployed system. This input might be a sequence of crafted PDUs that results in an unexpected behavior of the system. The term that describes this approach is known as fuzzing. Fuzzers are tools used to create and inject a variety of inputs to a device. The expectation is that one or many of these inputs will have a negative impact on the device under test.

The rationale behind fuzzing and excitation of protocols with malformed inputs is to reveal the unintended behaviors in devices destined for deployment in the industrial automation process. Being able to observe this behavior and properly configure the devices to avoid and handle these cases might help designers and system administrators eliminate security flaws before implementation and deployment of these protocols.

There are several fuzzing frameworks and tools available for testing a large variety of applications and protocols. Many of the most common fuzzer frameworks are discussed in [8]. In this research, we attempt to build a multi-adaptive and multi-purpose crafted packet injector written in the C programming language. This software is intended to craft packets related to messages that are generated by Ethernet-based industrial communication protocols.

D. THESIS ORGANIZATION

A basic description of the structure and the components of any industrial control system is presented in Chapter II. An overview of the communication principles, the requirements of industrial networks, and the evolution process of communication principles in an industrial environment is given in Chapter III. A description of the fundamental concepts of some Ethernet-based industrial communication protocols, such as PROFINET, EtherCAT, EtherNet/IP and Modbus/TCP, is provided in Chapter IV. The steps in creating a testbed that simulates a hypothetical control process utilizing Siemens industrial automation equipment and PROFINET is described in Chapter V, which contains the experimental results. In Chapter VI, the fundamental principles of socket programming in a Unix-based operating systems are presented. Additionally, in Chapter VI the C libraries libnet and libpcap, which are utilized for creating a software tool capable of injecting crafted packets related to the industrial Ethernet-based communication protocols, are described. Furthermore, a description of the software design of the proposed packet injector is provided. The thesis conclusion is given in Chapter VII, where research conclusions and recommendations for future work are offered.

II. INTRODUCTION TO INDUSTRIAL CONTROL SYSTEMS

The term Industrial Control System (ICS) refers to control systems deployed in critical infrastructure and several sectors of an industrial environment for monitoring and control of automated systems. ICSs are responsible for acquisition of analog or digital signals at each level of an industrial process, communication of these signals between automated devices at the same or higher levels, and for interpreting and representing them in human-readable form. In addition to this, an ICS controls and issues commands to industrial devices based on current values of the monitoring parameters. Once installed and implemented successfully, an ICS acts autonomously.

Engineers carefully design this automated behavior. They predetermine the parameters and the kind of operations that an industrial process should perform. Nevertheless, an ICS, despite its automated nature, always interacts with human operators. The operators are ultimately responsible for monitoring the ICS operation so that they can confirm that the process and related equipment are operating within the design limits of the system.

Some typical applications of ICSs include electricity generation plants, the chemical and oil industry, water and waste water management, nuclear power stations, and the manufacturing industry (e.g., automotive, integrated circuits). Moreover, ICSs are deployed throughout the transportation industry, such as for management of traffic in train systems, management of traffic lights, and even in management of prison security. We also find examples of ICSs inside commercial or military ships and aircraft.

ICSs have gone through several stages of evolution over the years. In their very early stages, ICSs resided on isolated platforms. The data communication managed by the ICS between the interconnected devices was based on proprietary protocols and application-specific devices. In addition, the ICS services and data representation were accessible at a local level to the operators. As a result, networking was not common, and operators could monitor the operation of the infrastructure only in cases where they were physically present at the control system. Later on, the need for communicating

operational data between geographically separated areas and the increasing size of industrial environments made secure network device communication imperative for the safe operation of industrial equipment. At this point, a huge transformation occurred, during which these isolated platforms converted to open architectures composed of highly interconnected devices. There was also the necessary transition from proprietary to standardized designs that allows different devices under different control systems to interact by exchanging data through public networks like the Internet.

As previously noted, the operation of an ICS is complex, supporting the interaction of several subsystems and devices. A basic functional diagram of this operation is shown in Figure 1. The ICS, from a theoretical point of view, consists of the controlled process, controllers, sensors, actuators, and human-machine interfaces or engineering stations [9]. The controlled process is the entire infrastructure, and the design of the related ICS is tailored to serve the needs of the industry in which it operates (e.g., manufacturing, transportation management, energy production).

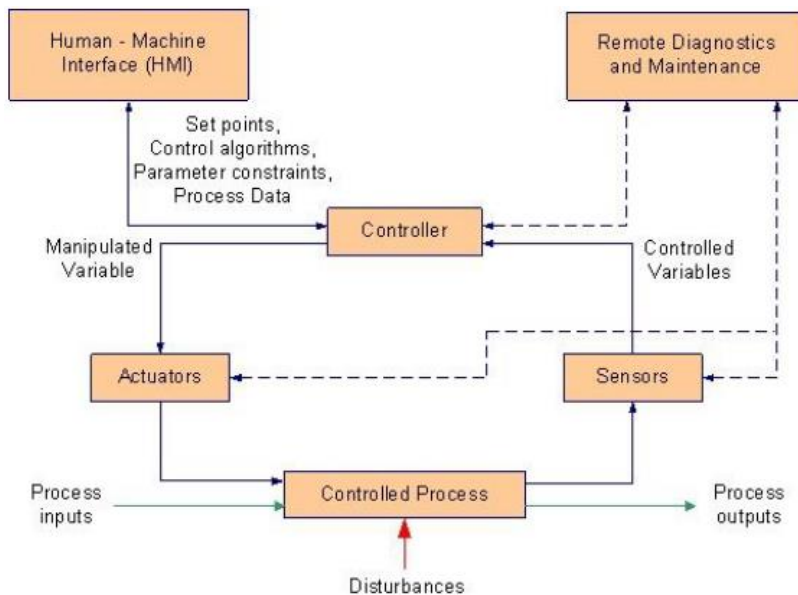


Figure 1. ICS Operation Functional Diagram. Source: [9].

The operation of an ICS is based on several control loops [9]. These loops use signals that are generated by sensors, controllers, actuators, and human operators to

perform dynamic adjustments to the whole process. Sensors are devices that generate signals in a digital or analog form; they are responsible for converting physical, chemical, or mechanical properties to electrical signals. Sensors are considered producer devices that actively generate signals. After processing, these signals can be used by other devices and can be transmitted to remote stations for further processing. Ultimately, these signals are converted to a human readable form.

Actuators are both passive and active devices which consume and produce data. In other words, they accept commands in order to perform a mechanical function and generate data indicating their actual state. For example, if the actuator is a control valve in a pipeline system, it usually receives position commands while providing feedback of its state indicating whether it is opened, closed, or has malfunctioned.

Controllers are mainly programmable devices that passively receive data from sensors, actuators, and engineering stations; they interpret these data and issue commands to controlled devices in the ICS. Human-machine interfaces, such as optical displays and panels, are used by an operator to monitor the ICS and apply changes to the system if necessary. The interfaces also display the interconnection of different devices in the ICS.

The components making up an ICS system are as follows: Intelligent Electronic Devices (IED), Remote Terminal Units (RTU), Programmable Logic Controllers (PLC), Human Machine Interface (HMI), supervisory management workstations, data historians, and business information consoles [10].

IEDs are devices that perform a specific function; these are sensors, actuators, pumps, circuit breakers, signal transformers, and switches. They transmit data to and receive commands from the PLCs and RTUs. RTUs gather data from IEDs and transmit them to PLCs or the HMI. Enhanced RTUs have more capability and sometimes perform the functionalities of PLCs.

PLCs are small-scale specialized computers running specific protocol stacks equipped with general purpose input-output ports. Recent advances have enabled a web server in a PLC. In addition, several vendors have developed PC-based PLCs. The PLCs are programmed with high-level programming languages, such as Ladder logic (LAD),

Statement List (STL), Structure Control Language (SCL), Function Block Diagram (FBD), Sequential Function Chart (SFC), and the GRAPH programming language.

HMI's are computers that provide a visual representation of the ICS. They are considered the “eyes” of the operator inside the complex system of the entire industrial process. Operators can issue commands, reconfigure the ICS, and monitor alarms and malfunctions. Supervisory management workstations, data historians, and business information consoles are similar to HMI's except that they cannot issue commands to the ICS and can be geographically dispersed and network connected.

It is important to understand the dependency and connectivity of components that constitute the ICS. Any disruption in one component's operation can affect the overall ICS performance, security, and safety.

III. COMMUNICATION PRINCIPLES AND REQUIREMENTS IN AN INDUSTRIAL ENVIRONMENT

The ICS utilizes different communication protocols and can be organized into different topologies. In the following sections, we focus on the fieldbus protocols and present fundamental concepts, evolution of the protocols, network topologies and traffic classifications. Additionally, we discuss the current trend in fieldbus networking, which is based on the Industrial Ethernet.

A. EVOLUTION PROCESS AND CHARACTERISTICS

Different manufacturers and vendors develop propriety solutions for providing distributed monitoring and control. These technical solutions are categorized as fieldbus technologies, which are described by specification IEC 61158. Since fieldbus technologies are proprietary, interoperability between them is difficult.

A fieldbus is defined as “a digital, serial, multi-drop, data bus for communication with industrial control and instrumentation devices such as—but not limited to—transducers, actuators, and local controllers” [11], [12].

One of the incentives behind the development of fieldbus technologies was the replacement of “simple signaling techniques” [11]. Examples of these simple signaling techniques are digital and analog signals. These might be current signals of 4.0 to 20.0 mA, voltage signals in the range of 0.0 to 10.0 V, and general purpose digital inputs/outputs of 0 (0.0 V) or 1 (5.0 V). These simple techniques have drawbacks, such as signal attenuation and noise immunity, making them unsuitable for large scale deployment.

The use of fieldbus to replace simple signaling techniques allows signals to be encapsulated into packets and transmitted using communication protocols. It also enables the ICS to be decentralized (i.e., pushing the processing of a large quantity of data to many distributed devices), thus avoiding the single point-of-failure issue.

The fieldbus protocol must also handle time-critical data, such as motion control and transmission/reception of emergency control signals. In addition, it handles real-time cyclic or acyclic operations. Cyclic operations are repeating, such as updating variables and parameters like temperature and pressure measurements, in a periodic fashion. Acyclic operations are functions that do not repeat periodically, such as firmware updates and device parameterization.

The characteristics of the ICS traffic are depicted in Figure 2. Two types of traffic are process data or management data. Both traffic types can be cyclic and acyclic, while process data is real-time and management data is non-real-time.

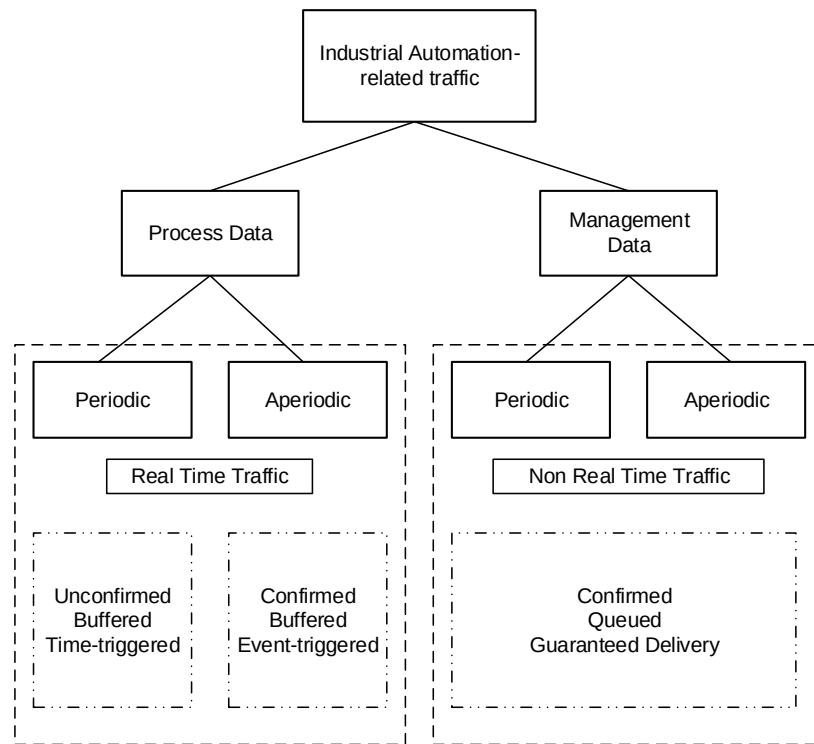


Figure 2. Classes of Industrial Communication Traffic. Adapted from [11].

Like many other protocols, fieldbus communication protocols utilize the International Organization for Standardization/Open Systems Interconnection (ISO/OSI) model. This model separates a communication system into seven abstract layers, known as the application, presentation, session, transport, network, data link and physical layers.

For each layer, data is encapsulated with a header and possibly a trailer. The encapsulation process introduces more delay and lowers the throughput of the system. Due to time-constraint requirements, real-time data usually utilize only three layers, while non-real-time operations may use all the layers of the ISO/OSI model.

As mentioned previously, the fieldbus communication networks must be able to support time-critical operations; hence, appropriate hardware with enough processing and memory resources must be used in these networks. Another important factor that needs to be considered is the network topology itself. The number of nodes within a network segment must be carefully selected, so that the segment does not overload the fieldbus communication network and impact the deterministic behavior of the fieldbus network (e.g., a cyclic operation of updating a temperature value every 2.0 ms). In addition, the network topology must offer maximum redundancy and survivability.

The most commonly used topologies are shown in Figure 3. A fieldbus network can be composed of multiples of these topologies. An example is a tree topology where every node is an individual segment of several automation devices. Devices within the segment might be further connected as a star or as a ring topology.

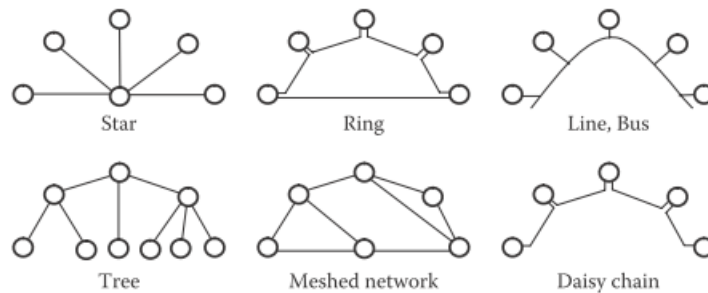


Figure 3. Typical Network Topologies at Fieldbus Level. Source: [11].

Another important mechanism that describes the behavior of a protocol is the medium access strategy. The devices in the fieldbus communication networks use time-division multiple access (TDMA) to communicate with different methods shown in Figure 4 [11]. The first method is polling, which has two models, master-slave and

producer-consumer. The second method is token passing, in which a device needs to acquire a token prior to transmission.

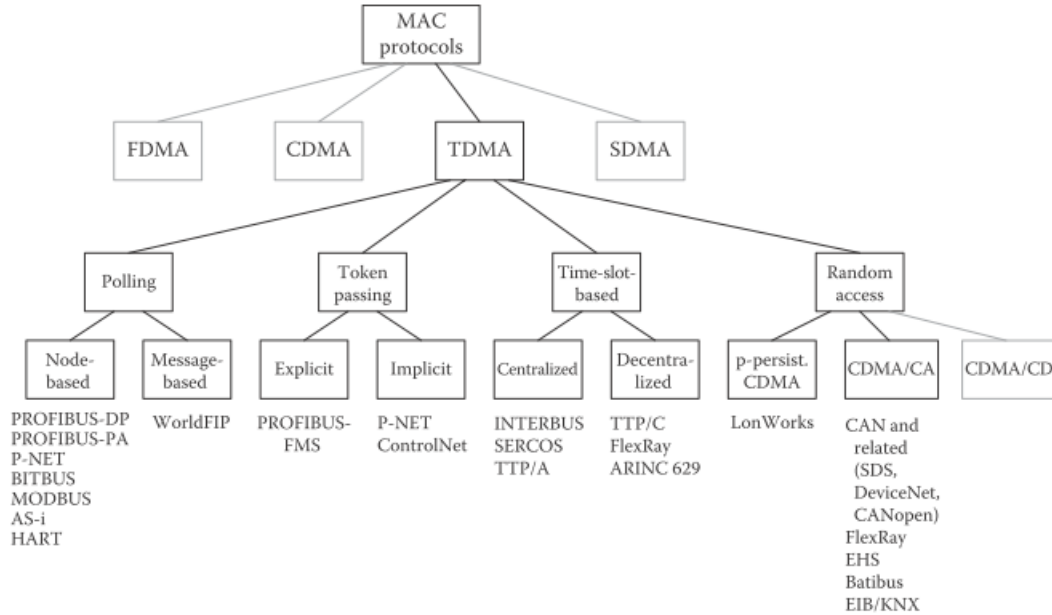


Figure 4. Medium Access Control Strategies in Fieldbus and Protocol Examples. Source: [11].

A third method is the time-slot-based method. In this method, the time domain is divided into time slots of equal or different sizes, depending on the operation that is performed by the device. One device can provide a synchronization message to other devices, or the devices can synchronize themselves using internally implemented timers.

The last method is the random access method. Devices can access the network to communicate at any time, increasing the redundancy of the network. On the other hand, this method can result in many packet collisions in the fieldbus network, causing non-deterministic behavior. Since industrial networks typically have strict timing requirements, the random access method must be implemented with care. Specifically, Ethernet uses an improvement to random access technology called carrier-sense multiple access with collision detection (CSMA/CD). This improvement has enabled Ethernet to become a promising technology for fieldbus network implementation. Some of the

features of an Ethernet network enabled by CSMA are collision detection, high speed switches, and full-duplex capability with gigabit-per-second transmission rates.

All the aforementioned features set up the foundation for a new standard known as the Industrial Ethernet, which is what is investigated in this research. It is becoming more common throughout the industry environment. Some of the main reasons that industry is transitioning from the serial fieldbus protocols to Industrial Ethernet are as follows [13]:

- a) The establishment of interoperability in communication between industrial equipment,
- b) The decreased costs of industrial equipment by utilizing standard equipment that have been widely deployed in IT networks such as switches and routers, and
- c) The advantage of increased speeds supported by the current version of Ethernet technology for transmission of data over copper wire or fiber optics.

B. THE INDUSTRIAL ETHERNET

Industrial Ethernet technologies were introduced as a replacement to the fieldbus serial communication protocols. They include the Ethernet protocol, rugged connectors, and hardened and high performance devices that are able to handle real-time requirements and harsh environmental conditions like humidity, mechanical vibrations, and high temperatures [14]. The Ethernet standard and protocol, with few adaptations, is used as the carrier for other proprietary industrial protocols operating in higher layers of the OSI stack.

The OSI model, according to [15], is a conceptual model. Its purpose is to define the communication principles and functions that characterizes the interaction of two devices within computer-based networks. As shown in Figure 5, the OSI model specifies seven abstract layers along with the related protocol data unit per layer.

Implementation	Layer Name	Name of Protocol Data Unit per Layer
Host related Layers	Application	Data
	Presentation	
	Session	
	Transport	Segments/ Datagrams
Media related Layers	Network	Packets
	Data Link	Frames
	Physical	Bits

Figure 5. The OSI Conceptual Communication Model. Adapted from [15].

Networking within the Industrial Ethernet is provided by the Transmission Control Protocol and Internet Protocol (TCP/IP) which, according to [16], “provides end-to-end data communication and specifies how data should be packetized, addressed, and transmitted, routed and received.” The mapping of the TCP/IP suite with respect to the OSI model is shown in Figure 6.

OSI Model	TCP/IP suite
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data Link	Network interface
Physical	Physical

Figure 6. Comparison of Layering between OSI and TCP/IP Suite

In a packet-switched network, application data is encapsulated layer-by-layer. As shown in Figure 7, data is encapsulated by layer 7 first. Afterward, the encapsulated data is passed to the lower layers, and their associated headers are added. When the receiving device receives the packet, the reverse happens, where each layer removes the

appropriate header. This fundamental concept of layering and encapsulation is used by industrial protocols that are based on the Ethernet standard. Encapsulation is an effective concept overall but has a drawback that affects industrial devices. Since traffic in industrial networks is time constrained, each layer introduces more latency, which affects the required deterministic behavior of fieldbus networks.

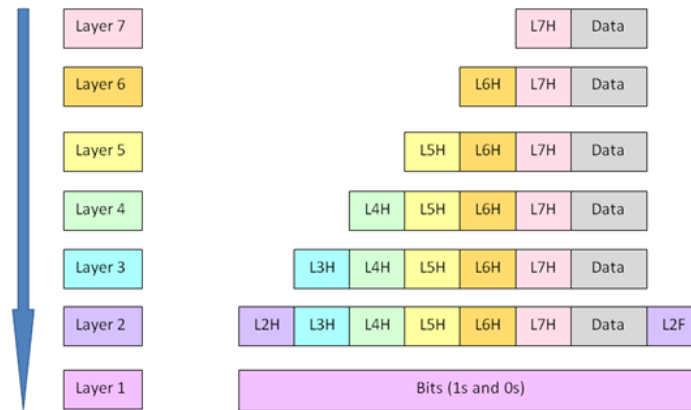


Figure 7. Encapsulation Process Illustrated. Source: [17].

In addition to latency, the jitter and bus cycle need to be considered in the design of an industrial network. Some device parameters and variables need to be updated in a periodic fashion. Jitter describes the variation from the expected arrival time of this periodic update. The time required for the periodic update is known as the bus cycle. The bus cycle is defined by “the time required to provide all nodes with new set points and read the current actual values of all nodes” [18]. The last concept worth mentioning relates to the prioritization of the network traffic in an industrial environment. For example, it is important to route alarm signals prior to other device traffic.

The network traffic is classified based on the industrial network’s usage [19]:

- a) Class A: Information technology (IT) and real-time (RT) traffic that is entirely implemented with protocols that are included in the TCP/IP suite. The stack might be slightly modified or reduced. The devices that are used for networking, i.e., Ethernet controllers and switches are ordinary with no special features related to processing power;
- b) Class B: This class is used only for RT traffic, and the network and transportation layers of the TCP/IP stack are bypassed. Only the Ethernet

frame of IEEE 802.3 is used from the TCP/IP stack. There is also the option of prioritization of traffic with the usage of the Ethernet frame described by the IEEE 802.1q standard. The devices again do not have special features related to processing power and performance. Any common Ethernet controller and switch available for the IT networks can be deployed here as well; and

- c) Class C: This class of traffic is the most demanding. It refers to both isochronous RT and RT traffic. Isochronous RT traffic is a transmission method by which devices are synchronized with very high accuracy. It requires special configuration and design of the segments. This requirement places certain constraints in the number of the devices per segment. Additionally, it introduces the concept of a synchronization domain, where all devices are synchronized under supervision of a dedicated master device. The TCP/IP stack is again completely bypassed and only typical Ethernet frames are used, with or without prioritization of traffic, to carry application layer industrial protocols. The hardware networking equipment must be optimized to perform quickly. Application specific integrated circuits or field programmable gate arrays (FPGA) might be used for handling the strict requirements of the synchronization domain.

The conclusions we draw from the previous classifications is that hardware and network configuration play an important role in industrial networks. They affect the performance of the communication, especially in the real-time and isochronous real-time applications. Additionally, whenever we have to deal with real-time traffic, isochronous or not, the time overhead of encapsulation introduced by several layers must be minimized by bypassing one or more layers of the TCP/IP stack. A last concept that we introduce is the meaning of connection-oriented and connectionless communications. Usually in the IT networks, we find both communication methods in use; however, as mentioned in [11], this is not always the case for industrial networks. The connection-oriented concept requires that both end devices in a network have previously established a connection based on the protocol that they are using for communication. It also requires that when the destination device receives a message, the device provides a confirmation signal back to the transmitter, indicating explicitly the successful reception of the signal. In industrial control systems, this three-way interaction consumes a lot of time and bandwidth in the communication channel. As a result, most of the protocols considered in this research support real-time communications that are connectionless, and the receiver

does not confirm the receipt of data. Because variables and parameters are constantly updated, there is also no requirement for connection-oriented interaction, and as a result, connection-oriented interaction is avoided when real-time traffic is taking place. The received data are either buffered or they overwrite previously saved values.

In the next chapter, we investigate and review some of the most deployed communication protocols in ICS. In addition, we look into the encapsulation process and messages of these protocols, which forms the basis of our proposed fuzzing framework. The protocols considered are PROFINET, EtherCAT, EtherNet/IP, and Modbus/TCP. Different manufactures and vendors support each of these protocols. The current percentages of traditional fieldbus and Industrial Ethernet deployments are shown in Figure 8, along with the percentages of their related communication protocols. It is clear that traditional fieldbus protocols are being abandoned for Ethernet-based solutions. The most popular solutions thus far appear to be PROFINET, EtherNet/IP, EtherCAT, and Modbus/TCP.

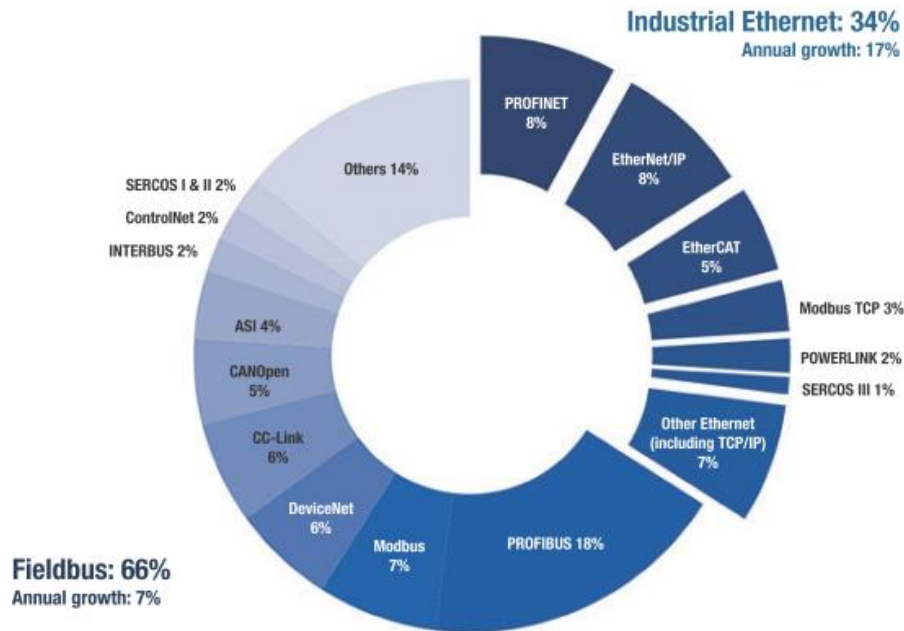


Figure 8. Industrial Communication Network Market Share. Source: [20].

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ETHERNET-BASED INDUSTRIAL COMMUNICATION PROTOCOLS

The following sections are a result of an extensive reverse-engineering effort in order to present the fundamental concepts related to the operation of Ethernet-based industrial communication protocols. Due to limited access to specifications or source code that describe the models, the messages, and the behavior of the protocols, many references were consulted to provide a condensed summary of the most important elements in the operation of these protocols. As a helpful reference for understanding their structure, the messages supported by these protocols are diagrammed and provided in the Appendix. These diagrams contribute to visualizing how the encapsulation process is applied in each case. Finally, the diagrams help in creating the required data structures for the proposed software tool.

A. PROFINET

PROFINET is an Ethernet-based protocol and automation standard introduced by SIEMENS AG. It supports 100 Mbps data communication, full duplex transmission, auto-negotiation of transmission parameters and auto-crossover [21]. As depicted in Figure 9, PROFINET incorporates two separate entities with different applications and functionality. These entities are known as PROFINET-IO (Input-Output) and PROFINET-CBA (Component Based Automation). PROFINET can use copper wire, fiber optics, and wireless as a physical medium.

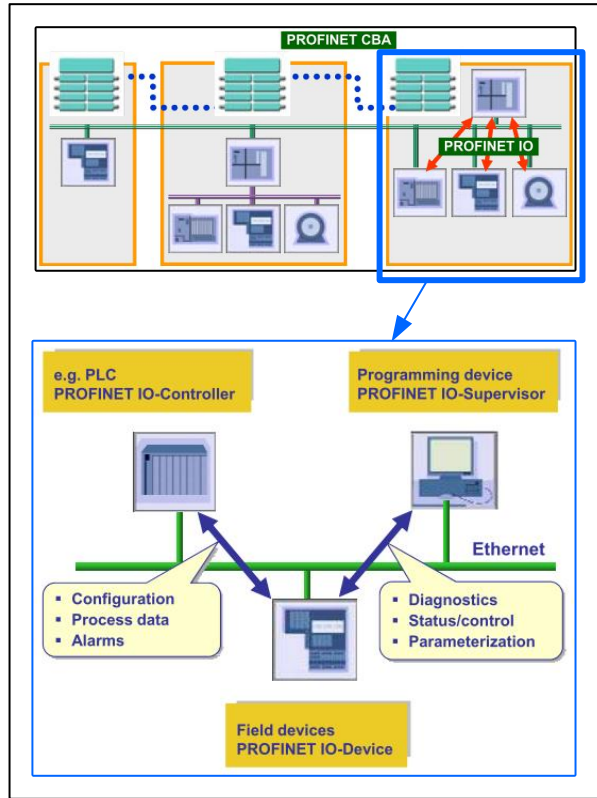


Figure 9. PROFINET CBA and PROFINET IO Illustrated. Adapted from [21].

As illustrated at the top edge of Figure 9, the CBA component of PROFINET partitions the industrial process into separate and independent cells that must establish communication for data exchange related to the monitored industrial process; therefore, different zones of a distributed industrial environment can communicate and exchange data. Each cell is considered a component or a module and contains several field devices, such as PLCs, HMI, sensors, general purpose IO devices, motion controllers, and many other similar devices. As can be inferred from [21], the CBA is used to create a distributed automation environment, where each cell is responsible only for an automation task. In the top right of Figure 9, a blue cell is shown, which sets the bounds for PROFINET IO communication.

PROFINET IO is applied within the module for communication between the devices that are considered members of the automation cell. PROFINET is based on a “Provider/Consumer” model and categorizes devices into three different entities: the IO

controller, the IO device, and the IO supervisor [21], [22]. The IO controller is usually a PLC that controls an automation task based on the input that it receives from peripheral IO devices. The IO devices are general input or output devices that transfer signals from the actual industrial process or receive commands by a controller; thus, the IO controller closely monitors and controls the IO field devices. The IO supervisor is usually a PC-based solution or HMI screen used for monitoring the process and for configuration or diagnostic monitoring of the devices. Together, the IO field devices, controller, and supervisor form parts of the PROFINET IO cell.

Both PROFINET CBA and PROFINET IO establish the same requirements for network traffic and communication. PROFINET incorporates several subcomponents in terms of sub-protocol variations to support all the classifications and requirements discussed in Chapter III under Section B. Both PROFINET CBA and PROFINET IO categorize traffic into classes A, B, and C. Classes A, B, and C can support real-time communication and a cyclic exchange of general purpose inputs/outputs, parameters, and alarms [22]. Class B differs from class A because it supports extra functionality, including network diagnostics and system redundancy. Finally, class C has a feature not found in class B; it reserves bandwidth in the time domain to support isochronous real-time traffic and synchronization of the devices that belong to the same synchronization domain. All of the aforementioned classes are also known as PROFINET RT for cyclic or acyclic data (RTC, RTA) and PROFINET IRT.

As discussed in Chapter III under Section A, the TDMA principle is applied at the fieldbus level. The same principle is applied to allow the PROFINET network domain to handle in a single time slot RTA, RTC, IRT, and non-RT traffic. This principle is illustrated in Figure 10, where it can be seen that the time domain is divided into several time slots of equal size. Each time slot corresponds to a bus cycle. The bus cycle is further divided into elementary time units of a certain size. Each of these units is allocated for the transmission of packets related to RT traffic and non-RT traffic. These units are shaded green in Figure 10. The cyan rectangles correspond to the time interval that is allocated for the isochronous RT traffic. This type of traffic sets the strictest requirements for accuracy, response time, and low jitter.

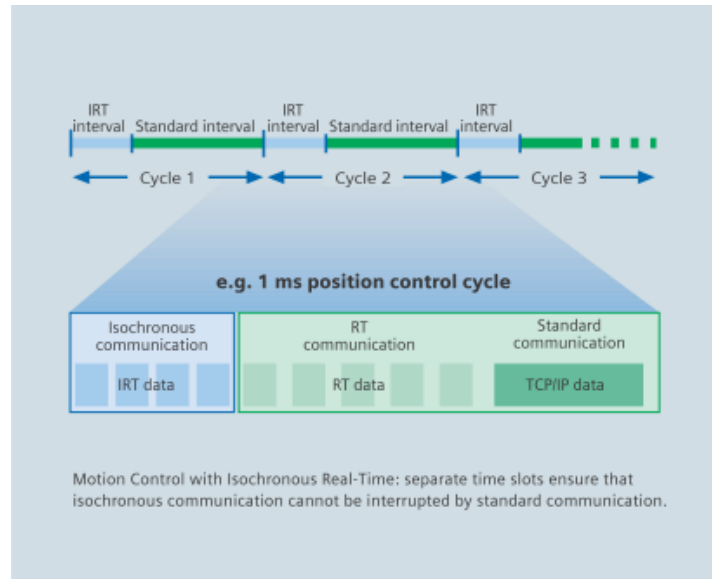


Figure 10. PROFINET Communication Channel Partitioning for RT and IRT.
Source: [23].

A rough representation of the requirements for RT, IRT, and non-RT traffic inside a PROFINET communication domain is presented in Figure 11. Non-real-time traffic is illustrated with the TCP/IP traffic. Also emphasized are the strict time constraints that the communication is subjected to inside an industrial communication network. In the case of the IT-related protocols, which are generally closer to non-RT traffic, jitter is not a serious concern; however, in RT and IRT, jitter should be kept as low as possible.

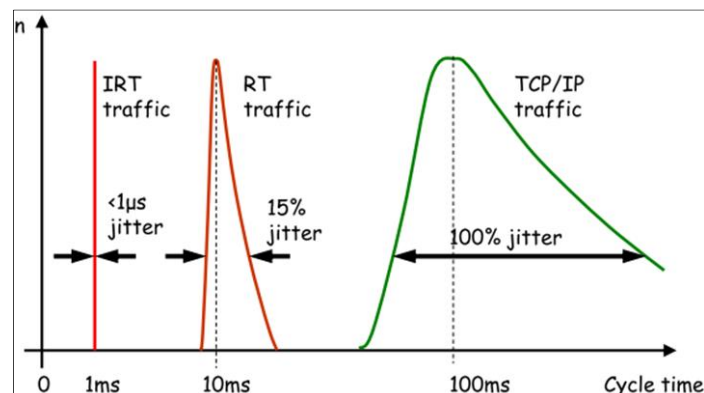


Figure 11. Performance Comparison between RT, IRT, and non-RT Traffic.
Source: [24].

In order to achieve the aforementioned performance, the PROFINET network must be designed very carefully. In practice, this means that the number and types of devices inside a PROFINET network must be taken into account before deployment. In TDMA, the bandwidth is interpreted in time metrics. As more devices are added into the network, each device transmits less since time is divided between more devices. In addition to this, as the bus cycle shortens, the available bandwidth for each device inside the network also shrinks. As a rule of thumb, the maximum available bandwidth in terms of time allocation for a device is less than half of the bus clock cycle. For instance, if the clock cycle in a domain is 1.0 ms, then the maximum available bandwidth for cyclic IO data, in terms of time allocation, is less than 500.0 μ s. The domain sometimes is also referred to as the synchronization domain because the IO controller device operates as a master device and sends a synchronization signal to all the slave devices and specifies the time basis so that all of them can communicate synchronously.

Many of the protocols that have been deployed so far in IT networks are incorporated for the non-RT traffic in the PROFINET domain. Currently available devices, like PLCs, HMIs, PROFINET networking devices, and general purpose distributed I/O devices typically include in their internal software implementation many of these protocols. In other words, the PROFINET devices usually utilize protocols like IP, TCP, and COTP but mainly for diagnostics, firmware updates, or configuration setup. This implementation is shown in Figure 12.

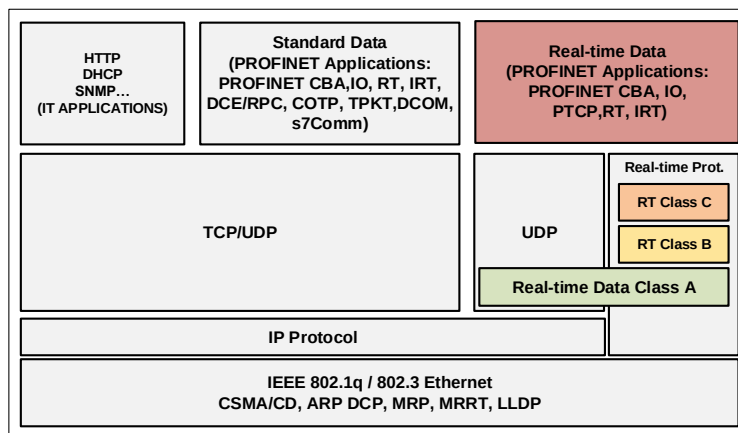


Figure 12. Protocol Implementation Stack in PROFINET. Adapted from [25].

Additionally, the application of the PROFINET communication protocol for handling real-time data of classes A, B, and C is also shown in Figure 12. Although the PROFINET sub-protocols are application layer protocols, the most demanding traffic is represented by RT class B and RT class C, which transfer their data on top of the Ethernet 802.3 or 802.1q. This reinforces the view that whenever devices have to deal with RT and IRT traffic, the TCP/IP protocol stack is bypassed. This helps to avoid multiple encapsulation levels and other stack inefficiencies. As a result, this practice saves clock cycles in the synchronization domain and clock cycles for the embedded microcontrollers of the industrial devices; thus, it is possible for the communication to meet strict time-synchronization requirements that are required amongst the several devices.

The green shaded area in Figure 12 demonstrates that some of the real-time traffic can be transferred on top of the connectionless user datagram (UDP) protocol, which is common in the Internet. Similarly, it is possible to route traffic related to HTTP, HTTPS, or SNMP protocols on top of the IP and TCP protocols. Finally, it can be inferred from Figure 12 that a mechanism is utilized for addressing the devices in a PROFINET-based communication network. This is done by utilizing IPv4 (Layer 3) addresses and media access control (MAC) addresses (Layer 2). Every device in a PROFINET network is assigned a symbolic name, and this symbolic name is associated with a hardware MAC address. PROFINET devices also utilize a Discovery and Configuration protocol, which is used by the IO controllers to discover and assign IP address to the IO devices. Additionally, PROFINET devices use the dynamic host configuration protocol (DHCP) to automatically assign IP addresses to the IO devices.

Since PROFINET and its variations are proprietary, it was not possible to access the original specifications or source code of PROFINET IO and PROFINET CBA for this research. As a result, information about PROFINET was gathered from open source intelligence and reverse engineering. In addition to this, in case of PROFINET we created a testbed described in following chapters. The testbed was created using Siemens equipment and provided the capability to see PROFINET “in action.” Using appropriate software for capturing and observing network traffic, we were able to observe

PROFINET behavior and its variations according to the task being performed by the devices.

Using references [25]–[27], the testbed, Wireshark (a network traffic capturing software), and reverse engineering, we were able to reproduce and outline some PROFINET messages. Since we do not have the original description of the protocol, complete interpretation of the payloads, headers, and data structures is not possible. All reverse-engineered packets are diagrammed and presented in Figures 42 through 49 in the Appendix. For this research, only the headers and the observed values were utilized so that packets could be reproduced and crafted using the software tool, which is presented later.

B. ETHERCAT

EtherCAT is an Ethernet-based industrial communication protocol initially developed by Beckhoff Automation GmbH; it is maintained and further developed by EtherCAT Technology Group. EtherCAT is standardized as a fieldbus level communication protocol that is described in IEC 61158 and IEC 61784. It is fundamentally an automation control protocol at the fieldbus level. In addition to this, EtherCAT supports process-data communication, which is mainly comprised of cyclic real-time traffic and transfer of parameters related to the industrial process. It also supports communication between devices for configuration, parameterization, and diagnostics. EtherCAT utilizes fast Ethernet 100BASE-TX for copper media and 100BASE-FX for optical fiber.

EtherCAT implements a master-slave model and separates the entire industrial process at the fieldbus level into several segments. Each segment includes a master and several slave nodes. The maximum number of addressable nodes in a segment is 65536. From the master device perspective, the entire segment with its sub-nodes appears as a single Ethernet device. In this way, the master device is required to generate only one Ethernet frame for each segment; thus, EtherCAT provides a collision-less network, where only one frame is transmitted at a time. The advantages of a collision-less network are many. One of them is that no switching equipment is required in the segment. This

lowers the total cost of the network infrastructure and, most importantly, eliminates any latencies introduced by the switching equipment. Each device receives a single Ethernet frame, reads the data that is applicable, and retransmits the frame to the next slave node in the segment. Besides communicating between the master and slave nodes, EtherCAT is also used to communicate between master devices. This functionality appears to be similar to the PROFINET CBA, where the industrial components are split into segments, and the master devices of each segment can communicate with each other. EtherCAT is also used to transfer the data and parameters to the HMI and operator's terminal for process visualization.

Only the master device can issue commands to the slave nodes in a segment. The commands are contained within an EtherCAT telegram, which is encapsulated inside an Ethernet frame. This telegram contains several smaller datagrams, where each datagram is intended for each slave device in the EtherCAT segment. This concept is illustrated in Figure 13. Each slave node receives the entire Ethernet frame but is only concerned about the datagram that references its own address. The slave node extracts and processes the appropriate datagram. Afterwards, it inserts new data into the datagram and forwards the whole telegram with the updated datagram to the next node of the segment. The last addressable node forwards the telegram back to the master device.

EtherCAT can be used over different network topologies such as line topology, daisy chain with or without drop line, tree structure, and star topology [28]. For real-time traffic, a star topology is typically used. The novelty of EtherCAT is that data is processed “on-the-fly” at the data link layer instead of “store-and-forward” at upper layers, reducing the total delay necessary for each slave device to receive, extract, process, insert, and forward the received packet [28], [29].

Another advantage of using the EtherCAT protocol is that there is no special requirement for hardware. The master device can be any common commercial-off-the-shelf computer system equipped with a 100BASE-X network interface card. The Beckhoff company provides TwinCAT, a development platform, for EtherCAT. Furthermore, there are open source implementations of the master device running on Unix/Linux based operating systems. One of these implementations is developed by the

EtherLab Organization and is available on [30]. For slave devices, an embedded system is required whose performance must satisfy the specification of the EtherCAT communication protocol. This specification is dictated roughly by the requirements of the class C industrial network traffic, as discussed in Chapter III under Section B. A slave device in an industrial environment can be a drive, a sensor, an actuator, or a decentralized peripheral embedded system with analog or digital input/output capabilities. The slave devices are usually implemented using an application-specific integrated circuit (ASIC) or an FPGA controller.

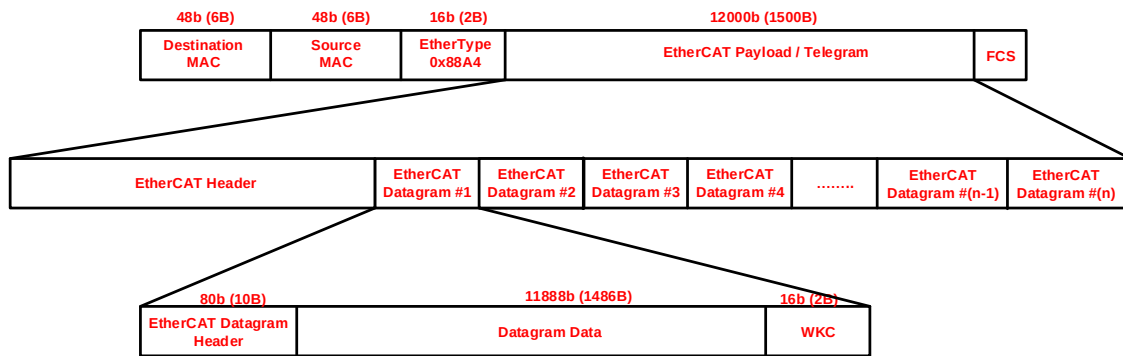


Figure 13. An EtherCAT Frame Encapsulated inside an Ethernet Frame.
Adapted from [29].

An important and novel feature of an EtherCAT-based network is the Fieldbus Memory Management Unit (FMMU). The organizational concept of the field bus level is based on the idea that the entire process, which constitutes the segment, is mapped in a shared logical memory space inside the master device. This logical memory space is mapped and mirrored inside every Ethernet frame. The datagrams concept discussed previously is based on the FMMU. The master device has a memory space allocated that mirrors the logical arrangement of the network in the segment. This memory space is split into smaller segments, and each segment of memory is assigned to a slave node. This space includes information about the state of any slave node, and the allocated memory space is used for tracking the entire process. It may also contain commands that are issued to the slave devices. In this way, the data that is transferred between the master and the slaves in a network segment are temporarily stored in this process image. The

master device logically creates this partition of the image, which is continuously updated by collecting the data that is transferred with the datagrams over the EtherCAT frame (which is encapsulated in the Ethernet frame). A rough representation of the mapping of the industrial process inside a segment that consists of a master and several kinds of slave nodes is provided in Figure 14. The FMMU concept is responsible for the logical addressing of the contents of the EtherCAT datagrams that are contained in a single Ethernet frame between the master and the slave devices.

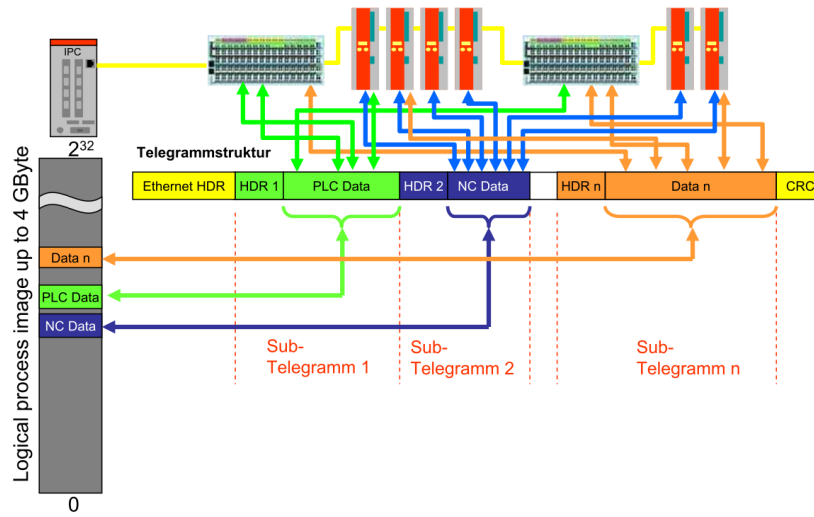


Figure 14. Memory Mapping of Process Image in the Master Device.

Source: [28].

In addition to the FMMU method, there are two other important addressing schemes supported by EtherCAT. These addressing schemes, are the positional and node addressing methods [29]. The positional addressing scheme uses a 16-bit field in the datagram header to identify the physical position of a device in the segment. The address of the master device is always zero, and any other slave device has a value set in increments by one depending on the order of electrical connections of the devices in the network. The positional addressing scheme is usually applied during the startup and configuration of the segment to detect the number of the newly attached devices in the network. On the other hand, in the node-addressing scheme, the master assigns to each device a unique address during the data-link initialization phase, and this is used to send

EtherCAT frames in an explicit manner to the nodes in the segments. Node addressing does not change and remains constant even if the configuration of the segment changes. By contrast, a positional addressing scheme might change (e.g., due to a failure of a network link or newly attached devices).

EtherCAT relies heavily on strict synchronization, which is very important for performance and safe operation of the equipment. EtherCAT utilizes a distributed clock mechanism [11], [28], [29]. This mechanism is responsible for synchronizing the devices to the same system time with a precision tolerance of 1.0 μ s or less. Due to this tight tolerance, it is important to consider issues such as clock drifts and time offsets of the devices as well as the network propagation delays. The distributed clock mechanism uses a reference clock, which is provided by one of the slave nodes in the segment. The node that is physically closer and electrically connected first to the master provides the reference clock. The reference clock in this particular slave device is also the reference for every other slave device as well as for the master controller. All other devices in the segment maintain an internal clock, which is not used as a reference. In addition to the use of a reference and internal clocks, three additional synchronization actions are performed [29]. First, the master device measures the round trip propagation delays to all the slave devices. Second, the offset of the slave devices' clocks with respect to the reference clock are calculated. The offset is sent to all the slave devices, which use the offset to adjust their clocks. Third, the master device periodically determines the time differences between the devices and sends out the difference to slave devices for drift compensation.

EtherCAT, like PROFINET, is an application layer protocol. It handles real-time traffic by using IEEE Ethernet 802.3. For non-RT requirements, EtherCAT can be encapsulated inside the TCP or the UDP protocols as payload. As shown in Figure 15, EtherCAT frames can be encapsulated either by TCP, UDP or Ethernet 802.3.

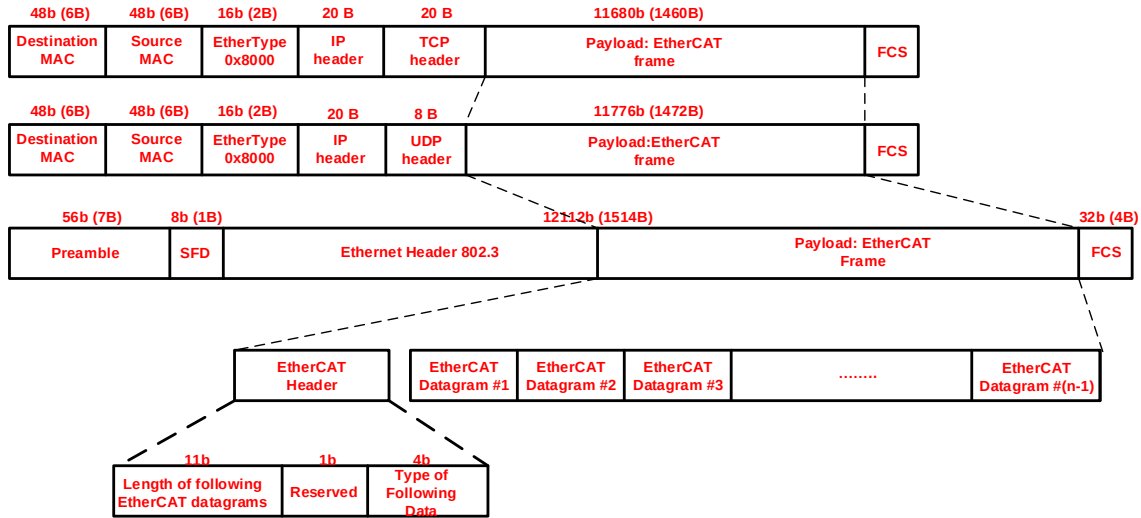


Figure 15. EtherCAT Encapsulation Process for TCP/UDP. Adapted from [31].

To achieve the assigned tasks, the application layer utilizes three classes of PDUs that correspond to different EtherCAT family communication protocols. Each of the communication protocols is identified by the “Type” field of the EtherCAT frame header as shown in Figure 15. These protocols are the EtherCAT Device Protocol, the EtherCAT for Process Data Protocol, and the EtherCAT for Mailbox Data Protocol. The EtherCAT device protocol is mainly utilized at the fieldbus level for isochronous and real-time communication with strongest requirements related to synchronization and fast bus cycles. It is used for data exchange between the master devices and the slaves. It is also used for reading values of distributed Input-Output devices and for issuing commands to motion control devices, actuators, and robotic devices.

The EtherCAT process data protocol is used mainly for communication between master devices or for transferring data from the lower layers of the fieldbus to the upper layers of the industrial environment; therefore, it is utilized for process visualization at the operators’ HMI panels. The operators’ computers cyclically poll the master devices using the EtherCAT process data protocol to obtain status data of the industrial process.

The mailbox protocol is used for two types of communication, asynchronous access to devices and incorporation of other protocols that do not belong to the EtherCAT family. Some asynchronous access examples are firmware updates, device

configurations, and remote access to devices. Protocols that can be encapsulated into EtherCAT datagrams are Ethernet over EtherCAT (EoE), CANopen over EtherCAT (CoE), File access over EtherCAT (FoE), Vendor Specific profile over EtherCAT (VoE), and Servo drive over EtherCAT (SoE).

EoE and FoE are used to tunnel IP/TCP/UDP traffic containing the EtherCAT datagrams. Another usage is for firmware updates. CoE is used to transfer another automation protocol standard known as CANopen. SoE provides real-time capabilities for motion control. Finally, VoE allows EtherCAT to transfer another vendor protocol, such as PROFINET. Within the EtherCAT payload, every datagram may contain data that are related to completely different services and protocols as shown in Figure 16. The header of each datagram contains a field named “type,” and the numeric value of this field specifies which protocol variation is contained in each datagram. The advantage is that data of different protocols coexist and are transferred simultaneously within the payload portion of the Ethernet frame.

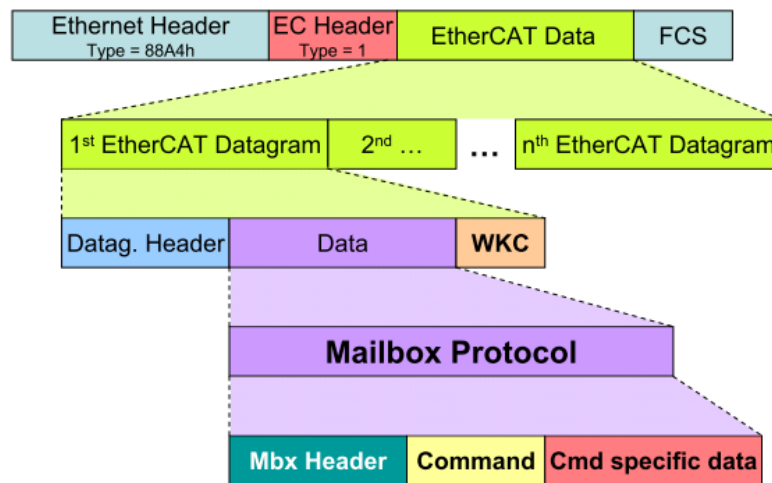


Figure 16. EtherCAT Application Layer Encapsulation in Ethernet IEEE 802.3.
Source: [28].

In the Appendix, the encapsulation process of several EtherCAT protocol variations is illustrated in Figures 50 through 57.

C. ETHERNET/IP

The Ethernet Industrial Protocol (EtherNet/IP) is another popular solution that was proposed and developed by the organizations OpenDeviceNet Vendor Association (ODVA), ControlNet International (CI), and the Industrial Ethernet Association (IEA). Although it is solely an application layer protocol like PROFINET and EtherCAT, EtherNet/IP has one fundamental difference. As discussed in previous sections, EtherCAT and PROFINET do not use the network and transport layers to meet real-time requirements but instead use the data link layer to encapsulate the data. As shown in Figure 17, the Ethernet/IP uses protocol layers up to the transport layer to carry a family of industrial protocols known as the Common Industrial Protocol (CIP). The CIP suite of protocols is outlined by the red border and utilizes TCP/UDP at the transport layer.

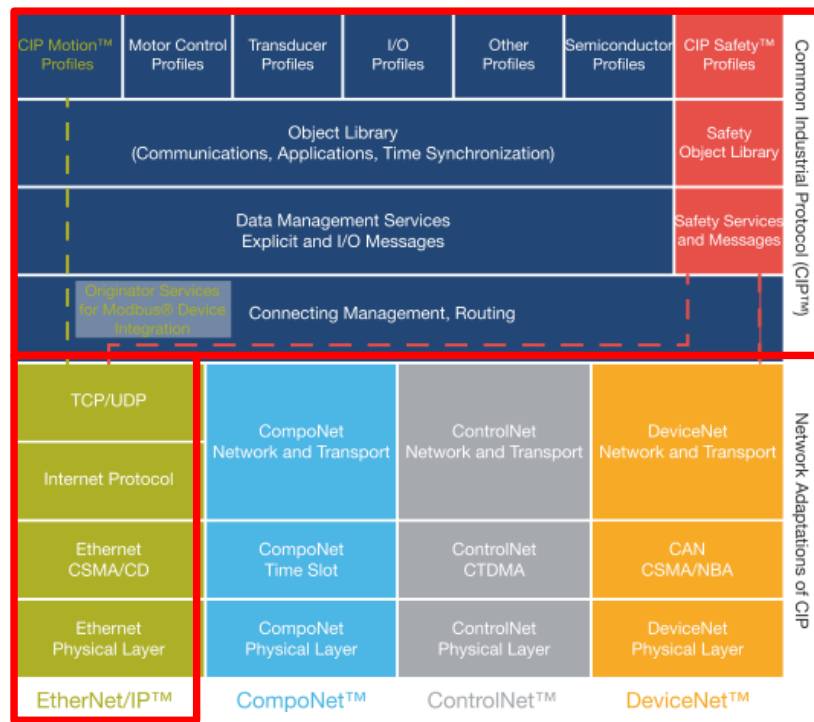


Figure 17. Ethernet/IP and CIP Protocol Stack Implementation. Source: [32].

As shown in Figure 17, the CIP is preceded by the EtherNet/IP. The main reason for the CIP development was the need for the creation of a series of industrial application layer protocols for industry that would be totally “wire-independent” and widely

compatible with any physical or data link layer protocol [33]. Ethernet/IP and CIP use a consumer/producer messaging model. The consumer/producer model is by default a multicast model. Producer devices generate data that are encapsulated in PDUs and then broadcast their messages inside the medium. The consumer devices, upon receipt of the packets and based on the connection identity that describes the PDUs, either accept or drop the packets. Any other devices in a network segment that need to consume the same type of data first need to ask for the connection ID and then are able to listen for the broadcast traffic, receive data, and process the data. Apart from the consumer/producer model, EtherNet/IP and CIP are still able to support point-to-point communication or client-to-server communication. All of these activities are performed using the TCP/IP or UDP/IP protocols.

The model just described uses two kinds of messaging schemes: the explicit and the implicit. In the implicit messaging scheme, the nodes that consume the data are allowed to imply the way that the data is going to be processed based upon the connection ID. The connection ID is usually obtained during a special request by a consumer node to join a channel that has already been established by a producer device. By contrast, in the explicit messaging scheme, the message includes a service identifier and an address that specify explicitly which consumers should receive the data and how they should treat the received data or how they should perform a certain action.

In the EtherNet/IP, the explicit messaging scheme is used for non-time-critical data transfers [33]. Explicit messages are utilized for point-to-point communication, and the TCP and IP protocols are used. As stated previously, explicit messaging is used to command devices in a segment to perform an action that is described in the messages that are sent. Reliable delivery and acknowledgment of the messages is required in this case; thus, the TCP protocol is employed. On the other hand, implicit messages are usually sent utilizing the UDP and IP protocol. Implicit messages contain time-critical data for transfer. In addition, messages might contain data that are required for synchronization between a producer device and many consumer devices. In this case, UDP is preferable as it is a connectionless service, generally utilizes smaller packets, has a higher speed, and achieves a better throughput.

Another fundamental concept is the object-oriented nature of the CIP protocol. The devices in an EtherNet/IP segment are called CIP nodes, and each CIP node consists of a collection of objects [34]. CIP objects can be structured either in instances or in classes. Classes consist of many objects that belong to the same system component. Each individual object is an instance of a particular class. Objects and classes have attributes. Attributes usually refer to the kind of data that are contained in an object or a class, and usually, the data define the current state of the object or class. The CIP objects are used to define the communication services, the behavior of the CIP devices, and the way in which information is accessed and exchanged within a network segment consisting of CIP objects [34].

The preceding description of objects is fundamental for the task assignment and addressing of devices in a network segment. Every node, has a Node Address, a Class Identifier, an Instance Identifier, an Attribute Identifier, and a Service Code [34]. The node address from the EtherNet/IP perspective is the IP address that is assigned to every device in the network segment. The class identifier specifies which classes or objects included in a node are accessible from the network. The attribute identifier is an integer that corresponds to the attributes of the node, which are determined by the objects or classes of objects that they contain. The service code specifies what action the node should take upon reception of an explicit message.

The classes and objects are software implementations within the nodes and devices that comprise the industrial process. Each object or class that is added to a device adds an explicitly specified functionality to the device. The advantage of this approach is that every device has a limited number of functions and actions to perform; therefore, devices are not overwhelmed by other unnecessary tasks and the total performance and throughput per device is high. The CIP suite specifies, at a minimum, a certain number of objects that every device within an EtherNet/IP network should have. To increase the interoperability of the devices and flexibility for engineers in selecting hardware components for the design of the industrial process, the CIP suite intentionally maintains object identifiers whose functionality can be specified by different vendors depending on the special needs of the application.

As illustrated in Figure 18, the EtherNet/IP and CIP industrial protocol suite is incorporated on top of the TCP/IP protocol suite. The EtherNet/IP payload is encapsulated by TCP or UDP and provides the required headers that specify commands related to the objects that a device requires for access. The CIP is implemented on top of the EtherNet/IP and contains the objects implemented in each individual device, such as PLCs and HMIs.

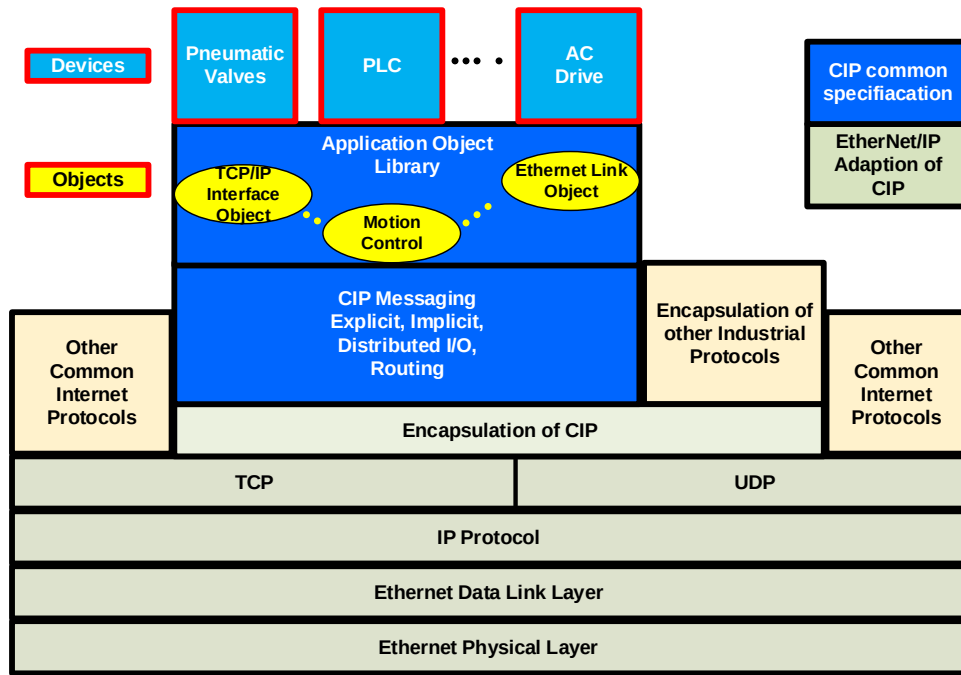


Figure 18. EtherNet/IP and CIP Protocols Organization. Adapted from [35].

EtherNet/IP categorizes devices into three different classes [32]. These are the Messaging Class, the Adapter Class, and the Scanner Class. The Messaging Class devices support only the explicit messaging scheme and rely on connection-oriented data exchange to communicate with other devices. These devices are unable to support real-time communication, and they are primarily used for the configuration and programming of devices and network segments or for monitoring diagnostics in the distributed industrial devices. The Adapter Class devices are receivers or transmitters with real-time capabilities, but in order to initiate this type of communication, must specifically be tasked to do so with an explicit request in a connection-oriented or connectionless fashion

that is usually initiated by a scanner class device [32]. Examples of Adapter Class devices are usually distributed, general purpose, Input/output devices that receive signals from sensors, controllers, or HMI screens that receive or send real-time data to or from PLCs. Scanner Class devices are the originators of the real-time data exchange connections and are usually PC-based controllers or PLCs.

D. MODBUS/TCP

Modbus/TCP is another Ethernet-based communication protocol. Like PROFINET, EtherNet/IP, and EtherCAT, it is solely an application layer protocol. Modbus was originally developed by Modicon and initially used a serial protocol in the physical and data link layers based on RS485. Its usage was primarily for data exchange, like digital or analog inputs between PLCs and HMIs. Since Modbus was totally independent from the physical and data link layers, it later adopted Ethernet technology, which led to the Modbus/TCP implementation. Modbus/TCP in contrast to the PROFINET and EtherCAT protocols, uses the entire protocol stack of the TCP/IP suite to establish communication in the industrial network segment [36], [37]. Additionally, Modbus/TCP can use any kind of available media, like copper wire, ether, and fiber optics.

Modbus/TCP has a relatively simple operation and implementation protocol, and its model does not differ much from common IT networks. Modbus/TCP is based on a polling method, server-client and master-slave model. In the master-slave model, each Modbus/TCP network segment contains a device configured as a master and is the only device that can initiate transactions or queries for the slave devices; therefore, the master device usually implements the role of a client that needs to establish a connection with a slave device. The slave device operates as a server that provides the requested data by the master device. For the master device to obtain a complete picture of the process, the device needs to cyclically poll all slave devices to obtain values of distributed-IO devices, sensors, actuators, or any other kind of peripheral devices. The slave devices return queries explicitly addressed to them and do not reply to broadcast queries or initiate communication with a master device [36]. This behavior limits the traffic generated in the

network to the minimum required in order to achieve what is necessary for the monitoring and control of the industrial process.

Because of the connection-oriented nature of the Modbus protocol implementation, the capability to handle highly demanding real-time traffic is doubtful. This assumption is based on the operational model of Modbus/TCP as presented in Figure 19. In Figure 19 and as mentioned in [19], we can see that every request sent by the master device needs to traverse the TCP/IP stack at least four times. Additionally, the establishment of a connection introduces extra delays. Apart from this, some network switching equipment, as shown in Figure 19, also introduces additional delays. Finally, the traverse of the entire TCP/IP stack in a device might introduce further delays related to stack implementation inefficiencies. These inefficiencies are mainly determined by the operating system and hardware design of the master and slave devices. In the case of Modbus/TCP, low jitter and deterministic transmission cycles are difficult, if not impossible, to achieve. The TCP/IP implementation imparts to the Modbus all the disadvantages and delays common to IT networks. On the other hand, the Modbus TCP/IP model is extremely simple and easy to implement in any device that supports the TCP/IP suite of protocols.

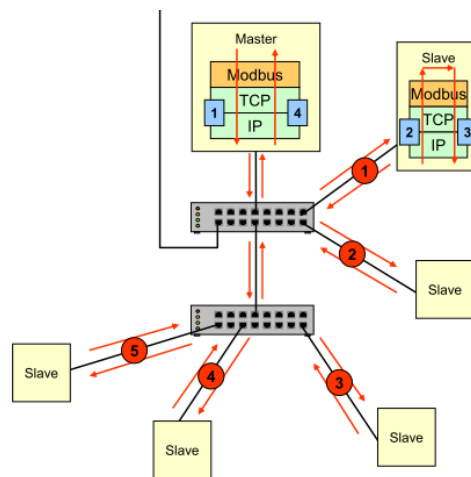


Figure 19. Modbus/TCP Master-Slave Communication Model. Source: [19].

As stated previously, Modbus/TCP is an application layer protocol responsible for interpreting the data encapsulated inside the payload portion of a TCP datagram. The Modbus Application Protocol Data Unit (MBPDU) includes a typical header name MBAP, a Function Code, and the Modbus data payload [37]. An accurate description of the encapsulation process for the Modbus/TCP datagram is shown in Figure 59 of the Appendix. The builder and initiators of Modbus/TCP datagrams are always master devices which can issue commands to the slave devices through their Modbus datagrams. At this point, the function code of the PDU is the most crucial parameter as it determines what kind of operation is required in the slave device. Moreover, the function code determines how the slave device interprets the data in the payload portion. The function code usually determines what operation the client wants to perform in the Modbus data object, as shown in Figure 20. These are usually read and write operations in the slave device's application memory. This might include discrete bits and bytes related to actual digital or analog inputs/outputs or several values that are stored in registers or entire files. Apart from these, the Modbus communication model offers some more advanced functionality that allows client devices to configure the application objects' attributes or enable other application services [38].

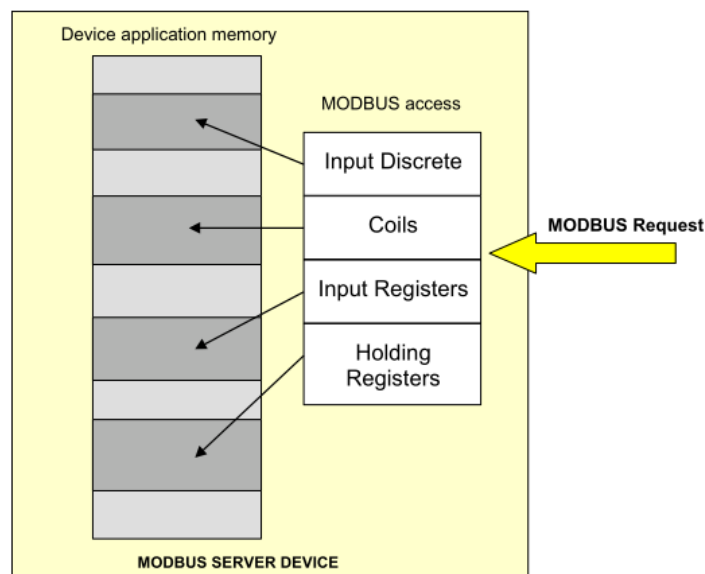


Figure 20. The Modbus Object Data Model in Server/Slave Device.

Source: [37].

The object model implemented in the slave device typically processes a request that has been received from a master device [37]. The processing includes a series of input validation functions. These functions confirm that the PDU contains a valid protocol type and a header with a legitimate function code operation. If the function code is not supported by the slave device, or if the code contained in the PDU protocol is not valid, then the slave responds to the master with an exception message indicating an error. The process continues checking the validity of the address and the size of the requested data. The same process occurs if a write operation needs to be done in the slave device. It might seem that Modbus input PDU validation could easily handle the case of a malformed input upon reception of a request; however, the Modbus/TCP protocol description does not seem to contain a mechanism for validation of the authenticity of the source of the request. That said, the model and implementation of Modbus/TCP does provide an access control mechanism. Through this mechanism, only specific requests are executed, those initiated by devices with certain IP addresses and an appropriate unit identification code.

E. SECURITY CONCERNS

Cyber security of industrial control systems is becoming more important. The communication protocols that were presented in the previous sections play an important role in every approach for investigating security flaws and vulnerabilities in industrial equipment.

The communication protocols of the devices allow industrial devices to interact and communicate with operators and other cooperating devices. As a result, any approach to investigate the security of industrial control systems must build on a complete understanding of the communication protocols. It is also important to understand the models of the finite-state machines implemented inside the control system relative to the communication protocol stack implementation. This model and protocol implementation might be described in a high or low level programming language like C or assembly. If there is a vulnerability in this protocol implementation, then it is possible, for example, for an industrial device to fail upon receipt of a malformed input. Failure of a device

might result in loss or damage to the physical infrastructure of the industrial environment or other unpredictable side effects.

As mentioned previously, the integration of industrial networks with IT networks has created additional opportunities for potential attackers to exploit ICSs; however, this network integration itself is not always the cause of vulnerabilities. As we learned in the case of Stuxnet, the enterprise network and the Internet were not the starting point for an attack. In practical terms, this means that an ICS might still be insecure, even if it resides behind numerous firewalls, demilitarized zones, intrusion detections systems, intrusion prevention systems, and “air gaps.”

The vulnerability present in the Ethernet-based industrial communication protocols is the lack of any mechanism embedded within the communication protocols themselves that protects their confidentiality, integrity, and availability. Confidentiality is not protected because a simple capture of network traffic related to the described protocols can reveal many clues related to the role of the connected devices in the industrial process, the parameters that are monitored, the number of the devices in the network, and the established communication channels (i.e., which device is “talking” to another device). Their availability is vulnerable because network bandwidth exhaustion, especially for real-time applications, can totally corrupt their operation; small bus cycles and low jitter are crucial elements for the guaranteed delivery and reception of critical parameters that are monitored in an industrial environment. Finally, the integrity is not protected. In industrial communication networks, this is of high importance if we consider the fact that the PDUs are actually carrying not just data but instructions for execution by the devices in real-time. The impact of an intrusion can be significant if any of the devices starts executing commands without first validating the authenticity of the source of the instructions.

The lack of authentication and encryption mechanisms in the protocols discussed so far is emphasized in [39]. The application of any encryption scheme at the fieldbus level seems to be partially an unrealistic scenario. Encryption and authentication are operations that require consumption of processor resources and multiple interactions between the devices; thus, they introduce time overhead, which is undesirable for the

fast pace of industrial process. The fieldbus level relies heavily on the RT and IRT capabilities of the protocols to perform the tasks of monitoring and control of industrial devices.

The RT and IRT specifications of the protocols presented previously are transferred over typical Ethernet frames. Most of the encryption schemes developed so far are applied either on the application layer or in the transport and network layers of the OSI protocol stack. Consequently, the capability for the deployment of any encryption scheme at the link layer is something that needs further investigation. As mentioned in [39] and as suggested by the vendors and developers of the industrial protocols, the security of the ICS can be achieved only by the partition of the ICS into autonomous and isolated cells. This kind of segmentation can limit the impact of an infection or an attack and creates secured cells, where the automation devices can communicate based on a trusted relationship over the real-time variations of the protocols.

All of the aforementioned security concerns are also an attacker's objectives. The primary objectives of an attacker can be summarized in the following tasks against the operators or the industrial devices [40]:

- a) Loss of monitoring and control capability of industrial equipment. This is similar to a denial-of-service, denial-of-view, and denial-of-control of the equipment, and
- b) Manipulation of the monitoring, view, and control of programmable controllers, sensors, actuators, and HMI.

In the next chapters, we present considerations related to the development of a testbed and a software tool that help examine the aforementioned security concerns. We also develop and present the stages for creating the testbed. Finally, we treat the testbed as a “black box” and try to use any available means to perform an integrated reconnaissance. This includes the use of several available exploitation tools and the proposed software tool. The use of software tools in the reconnaissance phase can help detect any exploitable vulnerabilities that might be present in the system.

THIS PAGE INTENTIONALLY LEFT BLANK

V. INDUSTRIAL TESTBED IMPLEMENTATION

Apart from the theoretical approach of the communication protocols and related security issues discussed in Chapter IV, we considered that engagement and practice with actual industrial equipment would give better insight. The thoughts and stages we went through in our effort to build a realistic industrial process are presented in the next sections. We describe the equipment that we utilized for building a testbed as well as the programming we performed for enabling the full functionality and communication between the automation devices. Moreover, we present a series of useful observations that we made after programming the industrial devices and testing some publicly available exploitation tools. Finally, we perform a reconnaissance against the internal hardware architecture of an industrial controller that was part of our testbed.

A. THOUGHTS AND CONCERNS

There are many manufacturers of ICSs and industrial communication protocols and it is difficult to test and evaluate security concerns for all of them. In addition, the cost for these systems is not negligible, making it difficult for researchers to procure and evaluate them. Finally, open source intelligence gathering for these proprietary devices is difficult. Consequently, researchers cannot collect many clues about the software or hardware design and circuits of these devices. Similarly, there are few details about the processors, their instruction sets, and the memory arrangement or the Ethernet controllers that are running inside these devices.

The situation is almost the same for finding the protocols' standards, specifications, and description. Many protocols are maintained by developers who work for industrial equipment companies. Other protocols are maintained by scientific groups and consortiums. For instance, Siemens AG is the developer for the PROFINET communication protocol. Open DeviceNet Vendors Association (ODVA) is another organization that maintains and develops Ethernet/IP. As a result, access to the source code that describes the behavior of the protocol and the classes of messages is not available. In this case, open source intelligence seems to be the only option for obtaining

detailed knowledge of the industrial communication protocols outside of buying the specifications. Furthermore, no virtualization technology has been developed for industrial devices (e.g., PLCs and HMI devices). As a result, it is not possible to deploy many of them in a virtual environment for experimental trials. In the IT world, it is easy to deploy any number of operating systems as virtual machines and test them; this is not the case for industrial control systems.

As a result the following questions must be considered: is it possible to create a testbed environment that could validate secure operation of a system efficiently before its deployment in the industrial process? If we assume that this is feasible, which software or hardware tools should be used? Which method or approach can provide the assurance—before its deployment and use in the industry—that an embedded device is safe and can resist most types of attacks?

In this thesis research, we approach these questions by performing an integrated investigation, reverse engineering, and fuzzing of industrial equipment and its associated communication protocol. Some fundamental properties and functional principles of industrial protocols based on Ethernet were discussed and presented in Chapter IV. Fuzzing and reverse engineering are approaches that may yield interesting results and are also likely to be employed by malicious attackers. To begin the process of reverse engineering, we must purchase and program some of the industrial devices. After programming the devices, we observe the behavior of the devices and the communication protocol. Moreover, we can disassemble the hardware and the components installed inside the devices. If this is possible, then we can perform information gathering related to the processor architecture and related instruction set utilized by the device. Furthermore, if we properly set up a network of similar devices, we can test the devices with common exploitation tools and see which tools may or may not achieve a successful attack. At the end, we can fuzz the devices with appropriate or malformed inputs and observe their behavior.

To test an industrial network for vulnerabilities, we created two separate entities: a testbed that emulates a small-scale industrial process and a fuzzing framework. For the former, we utilized Siemens AG equipment and the corresponding Application

Programming Interface (API) software utility. For the latter, we implemented a fuzzing framework using the C programming language and network libraries to craft and inject packets into the Siemens industrial testbed network.

B. TESTBED SET UP

As emphasized so far, engagement with actual industrial equipment is required to observe the operation of industrial devices and communication protocols and detect security flaws or vulnerabilities if present. There was no particular reason for selecting Siemens equipment for setting up the testbed. We could have easily set up a similar environment using Allen Bradley/Rockwell industrial automation equipment.

The testbed includes two of the most recently developed industrial controllers, an HMI screen, and an industrial switch. The equipment type, their part numbers, the functionality, and other more general or technical features are presented in Table 4 of the Appendix.

To make the testbed more realistic, the devices were programmed to simulate a small monitoring and control application. In this way, we set up the devices as they were intended to operate and, therefore, could attempt to disrupt their operation. In addition to this approach, we needed to test each device individually by fuzzing it separately through its communication interface. For fuzzing of the industrial devices, as discussed later, we utilized a software tool developed specifically for this thesis which is discussed later at length.

The industrial process was built utilizing software provided by Siemens and known as the TIA Portal. The TIA Portal is a multipurpose platform which allows the user to program the programmable controllers and other peripheral devices. It allows fully functional simulation and emulation of the programmed industrial environment. In this way, the user can validate the correct operation of his/her design before actually downloading to the devices. TIA Portal allows the user to access the devices on-line while they are under operation. The advantage of this feature is that we can dynamically change some parameters of the programmable logic controllers. It also allows the user to access the diagnostics functionality embedded in the devices in real time. This is a very

important functionality, as it enables a user to see any errors that occur during run time and examine the on-board memory utilization of the PLCs. In addition, we can see the bus cycle diagnostics. As mentioned earlier, the bus cycle is the total time required to provide all nodes with new set points and read the current values from all nodes in the same network segment. All these diagnostics are used later to examine whether our fuzzing operations, or other types of attacks affect the PLCs, the resources, or other peripheral devices.

The TIA Portal supports several programming languages for PLC controllers. For simplicity, we developed the monitoring and control operation using the LAD programming language. The LAD programming language is a graphical language, and the basic principle of this programming language imitates the behavior of electromechanical relay contacts that were heavily utilized in early automation designs. In other words, in LAD, we expect to see simple designs with electrical relay contacts that indicate the presence or the absence of a Boolean logic value in one of the digital input modules. LAD is also enhanced with some advanced features. These are function blocks which, when enabled by a digital signal, can perform more complicated tasks. Some of these tasks are implementing timers, counters, comparators, mathematic functions, move operations of memory segments, or signal conversions.

For Siemens equipment, one of the most fundamental concepts in PLC programming is Organization Blocks (OB). The OBs are independent entities of instructions that are executed by the PLC processor either on a cyclic basis or when triggered by certain events. Examples of such events might be the startup of the PLC, a hardware or software interrupt or an error. The OBs are very similar to system calls of the Unix operating system or function calls in the C programming language. The OB that is always present in any Siemens PLC program is OB1. This is a block similar to the main function we can find in any C program; there is, however, a fundamental difference. The OB1 is always executed cyclically, while the main function in a C program might be executed only once. The OB1 is also responsible for calling other organization blocks or other function blocks. The reason that OB1 is executed cyclically is that all values of the signals applied in the General Purpose Input-Output (GPIO) of the PLC must always be

updated. During cyclic execution, the OB1 always validates and checks the status of the inputs, outputs, or assigned memory areas, or makes function calls defined by the user. The cycle time that was mentioned earlier for Siemens equipment corresponds to the time required for one complete execution of OB1. Any sufficiently large deviation from preconfigured time margins generates an alarm and is considered a high importance event that it is logged.

Some examples of LAD networks are presented in Figures 21 through 23. In these figures, the symbols of open contacts correspond to digital inputs at the input ports of the programmable logic controller. If a high input voltage is applied to the input of the PLC, then the open contact is converted to a closed contact, and the high-level signal is transferred to the output indicated by %Q0.6 (i.e., absolute addressing) or “Tag_14” (i.e., symbolic addressing). Shown in Figure 21 is an OR Boolean logic function, and an AND Boolean logic is illustrated in Figure 22. All these networks can be placed either inside the OB1 or in function blocks specified by the user, which are called from the OB1.



Figure 21. Simple OR Boolean Logic Operation in a LAD Network

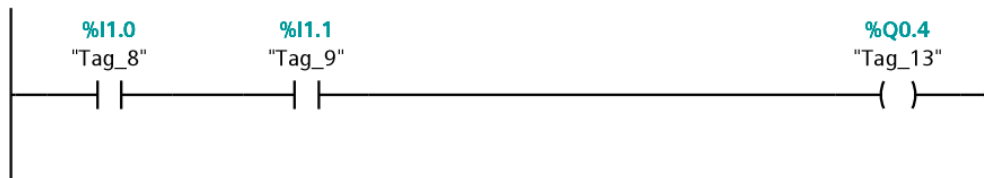


Figure 22. Simple AND Boolean Logic Operation in a LAD Network

In Figure 23, we see some of the more advanced capabilities provided by the LAD language. The boxes in the figure are function blocks that are used for analog signal conversions and signal conditioning. The first block takes a 16-bit input value generated by the embedded analog-to-digital converter of the PLC and converts it to a real number that is stored in memory. Afterward, this value is scaled appropriately by the second box, so it represents the actual physical property that we want to measure (e.g., temperatures from 0 to 200 degrees Celsius).

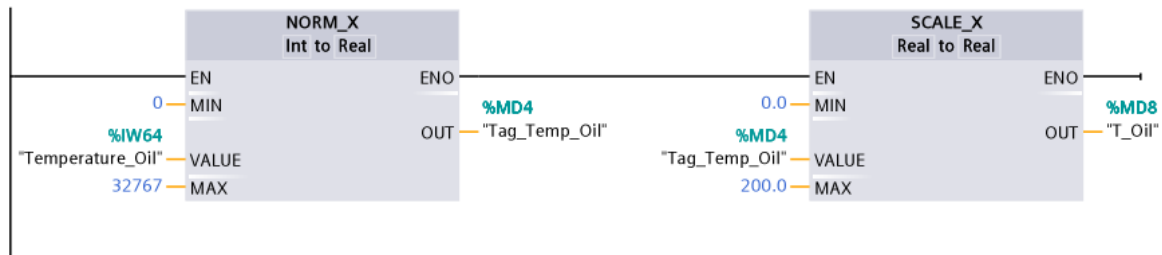


Figure 23. Function Block for digital-to-analog conversion, followed by Function Block for Signal Scaling

Monitoring and control is the underlying purpose of any ICS; thus, we decided to create the monitoring and control system of a hypothetical diesel-engine-driven electric generator. We combined blocks created by the LAD language with other programming utilities. In addition, some auxiliary machineries needed for the generator were incorporated as well. The TIA Portal provided the required features to graphically represent our design. We created or drew graphical objects and associated these with the signals and parameters. In addition, we created action buttons; for example, we can create a button to start or stop a machinery process. After the design is finished, we downloaded the configuration into the PLCs and HMI screen, creating the simulation platform for our security testing testbed.

The designed environment for the monitoring and control of the hypothetical generator is illustrated in Figures 24 through 26. More precisely, in Figure 24, we see that the diesel generator is surrounded by some auxiliary pumps, a lubrication oil pump, and a coolant water pump for cooling water circulation. We also see the heat exchangers for the

closed circuits of the lubrication oil and a coolant water. The “start” and “stop” buttons are used to start and stop the electrically driven pumps or the diesel generator. The boxes with the pound symbols indicate parameters like temperatures. These indications are derived hypothetically from analog sensors, whose analog signals are converted into digital values. The digital values are scaled so they properly represent the monitored parameter. The red circles in Figure 24 correspond to objects that indicate the status of the machinery. The color red indicates the machinery is not operational, while a green color indicates the machinery is in “running” status.

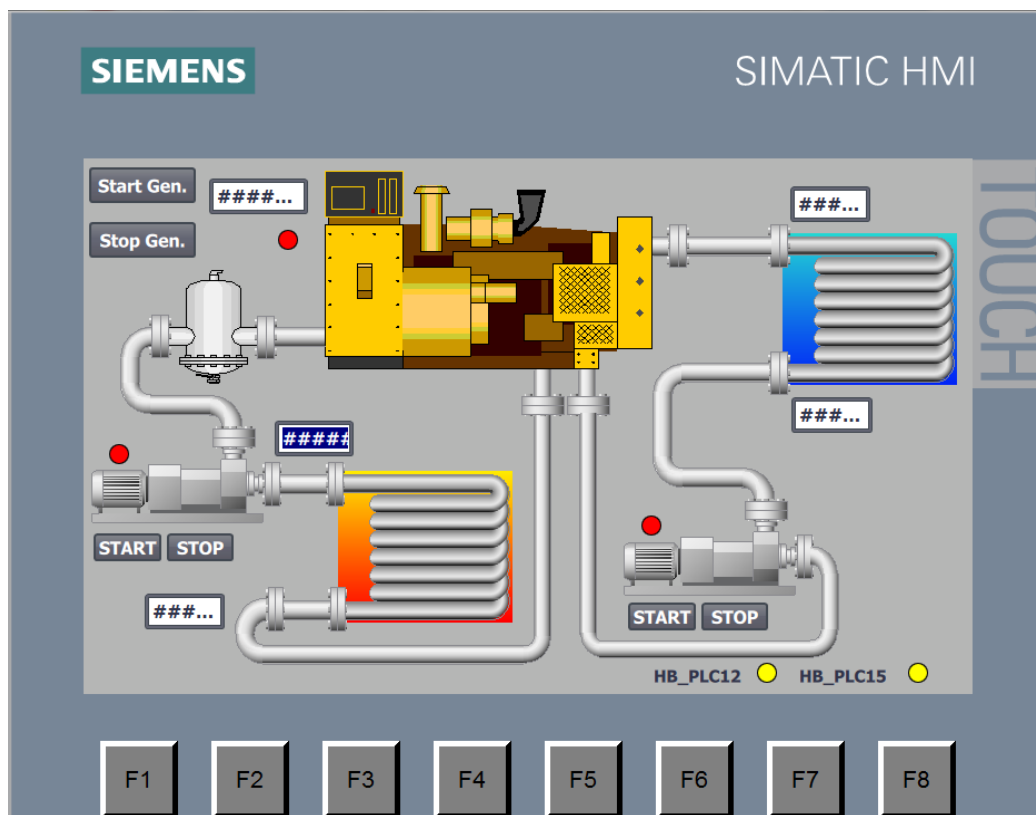


Figure 24. Screen 1 of HMI for a Diesel Generator and its Auxiliary Machinery

In Figure 25, some bar graph indicators are shown; they indicate parameters such as temperature, fuel level and oil pressure. These parameters are obtained from the embedded analog sensors; the analog readings are converted to digital signals for easy presentation at the HMI screen. The monitoring of these parameters is considered crucial

for inferring the “health” status of the machinery. If the values of the parameters are beyond the operating limit, alarms are generated.

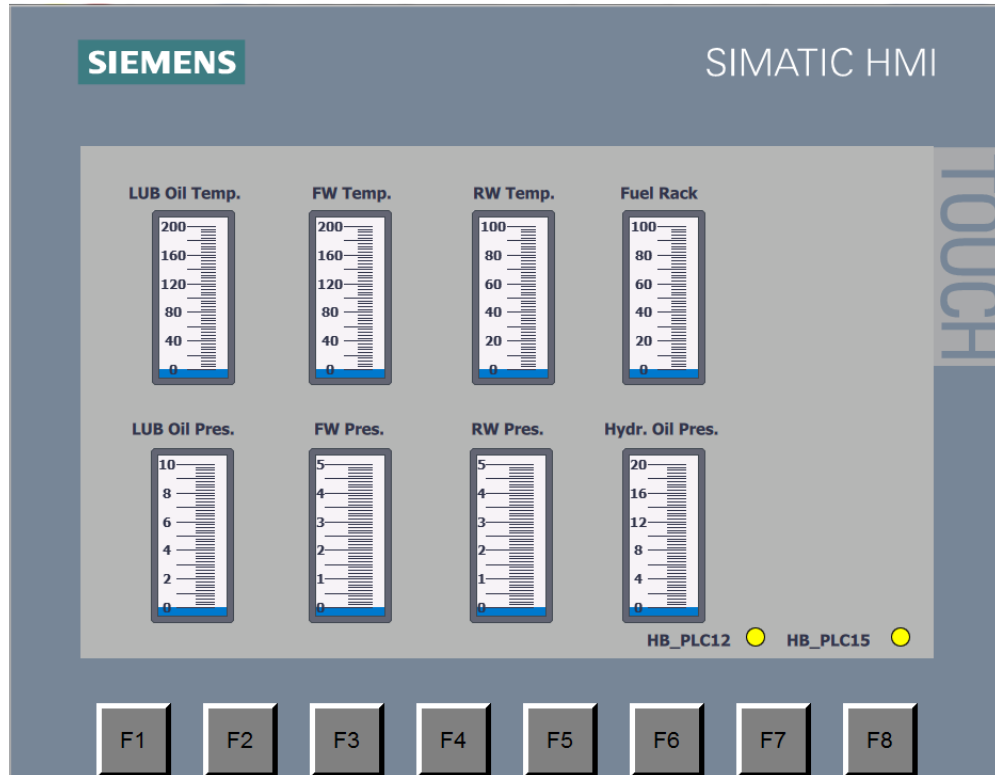


Figure 25. Screen 2 of HMI for Monitoring Operational Parameters

The auxiliary machinery, which is part of the fuel system that supplies the diesel generator, is depicted in Figure 26. The bar graphs indicate the level of the fuel within the tanks. The “O” and “C” buttons are used to control the fuel valves, either opening or closing them, respectively. The status of the values is indicated by the circular indicators that are right next to the “O” and “C” buttons; Red means closed while green means open. In addition, the “start” and “stop” buttons are used to start and stop the fuel pump.

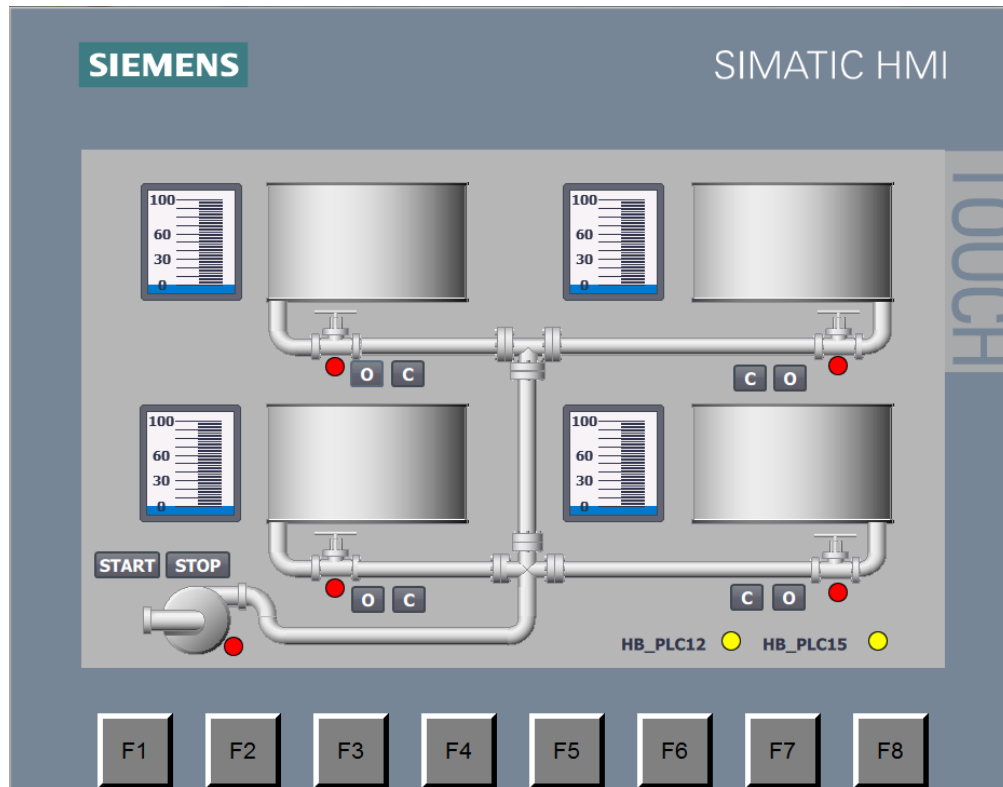


Figure 26. Screen 3 of HMI for Monitoring and Control of Fuel System

Besides designing and building the industrial network, it is important to establish the communication channels between devices within the network. Communication between devices is driven by the operational needs of the system. Once communication is established, we can observe and evaluate the underlying protocols that are employed. The topology of our small-scale industrial network is shown in Figure 27; it consists of one HMI and two PLCs. Communication between the devices is dictated by the requirements of the equipment and parameters that need to be monitored. The following points describe the specification:

- a) PLC 1 is responsible for monitoring the operation of the diesel generator and cyclically updates the monitored parameters;
- b) PLC 2 is responsible for monitoring the fuel replenishment and supply system and sending to PLC 1 the status of the fuel valves and the operating status of the electrically driven fuel pump; and
- c) HMI 1 establishes communication with PLC 1 and PLC 2, so that the operator is continuously informed about the status of the machinery. HMI

1 is also capable of issuing commands to PLC 1 and PLC 2 so that the operator can remotely control the devices (e.g., starting/stopping the pumps and opening/closing the valves).

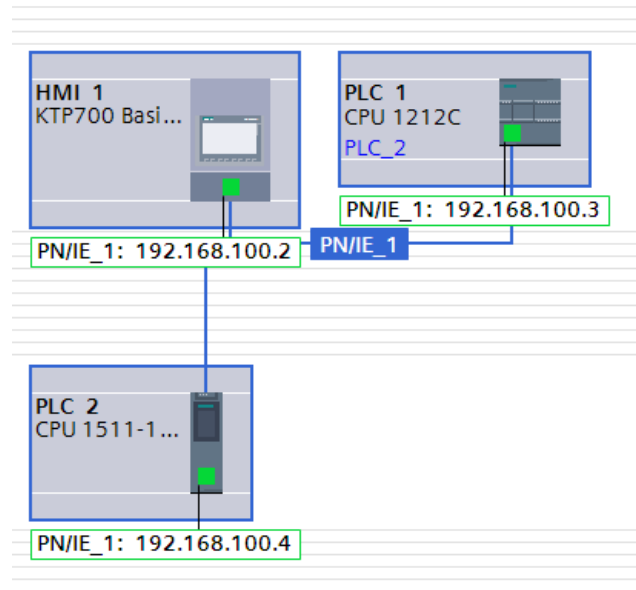


Figure 27. Network Topology Using TIA Portal

To observe and reverse engineer the communication protocols used by the industrial network, we use a passive Ethernet tap device. As shown in Figure 28, the tap is placed as a “bump in the wire” at every Ethernet cable that connects the devices in the industrial network. The green arrows represent the normal flow of the network traffic, while the red arrows indicate the flow of captured traffic. The Ethernet tap is a device that can forward the input traffic passively in three different directions. In other words, it takes a network flow as input and generates three separate output flows. Because full-duplex communication is established with Ethernet and PROFINET, the Ethernet tap splits them into two separate captured outputs, which are indicated by the top and bottom red arrows in Figure 28. The laptop connecting to the Ethernet tap has to place its NIC card into promiscuous mode, which enables the kernel of the operating system to capture all the packets arriving at the NIC. After promiscuous mode is enabled, the Wireshark utility is used to capture and display the packets.

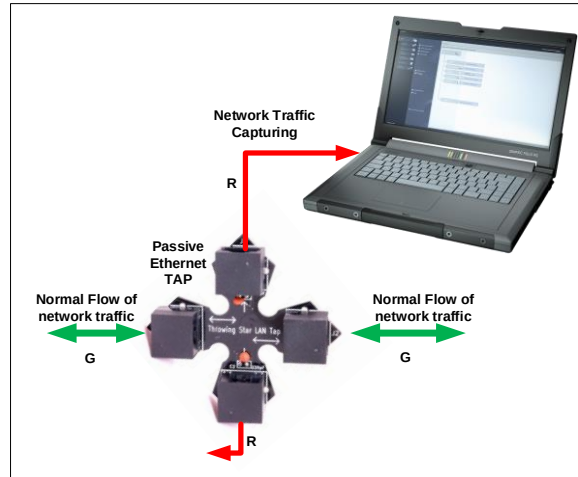


Figure 28. Topology for Capturing and Observation of Network Traffic

The network topology of the industrial network is shown in Figure 29. The programming unit represented by a laptop in the figure only connects to the network when the operator needs to program the industrial equipment or when the operator wants to access the online diagnostics provided by the PLC controllers and the HMI. The colors of the arrows indicate different types of traffic and protocols within the industrial network.

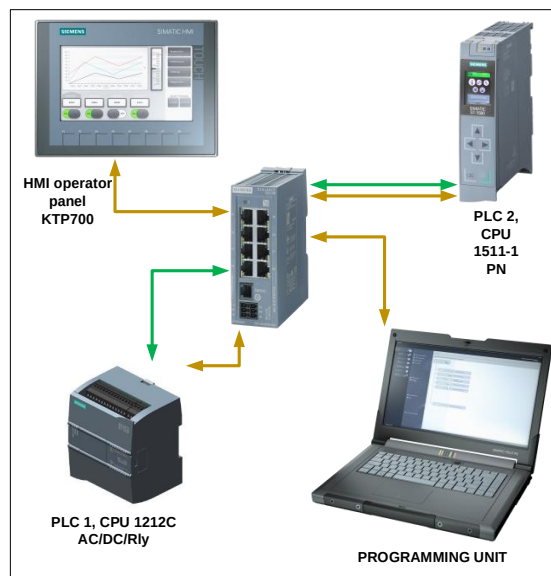


Figure 29. Topology of Communication Paths

During the programming phase, we captured four different types of network traffic. Three of them were obtained when the LAD program is downloaded to the PLC controllers and HMI using the TIA Portal API. The fourth type is from a firmware update; specifically, it was captured during a firmware update of the CPU 1212C AC/DC/Rly. We captured packets for two reasons:

- a) observe the types of protocols used, and
- b) harvest information related to the hardware architecture and instruction set transferred or exchanged in the payload of the packets during the programming or firmware updates.

The following is a list of protocols observed:

- a) Ethernet,
- b) Internet protocol version 4 (IPv4),
- c) Transmission Control Protocol (TCP),
- d) TPKT and ISO 8073/X.224 COTP at the application layer,
- e) PROFINET Dynamic Configuration Protocol (PN-DCP) encapsulated in a typical Ethernet frame of IEEE 802.3 or 802.1q, and
- f) PROFINET Media Redundancy Protocol (PN-MRP) encapsulated in Ethernet frame of IEEE 802.3.

After the devices were programmed and configured, we observed the testbed in action and were able to monitor the small-scale industrial system using the HMI. In addition, the PLCs were observed using online diagnostics and emulations provided by the TIA Portal. The network traffic between the HMI and PLCs was captured and afterwards was analyzed. As mentioned previously, the HMI requires a continuous update of parameters such as temperature, pressure, and the status of the industrial devices. As shown in Figure 30, the COTP data, which is encapsulated by TCP, is sent from the HMI to the PLCs via the network switch. The COTP data is used to provide commands to the PLCs.

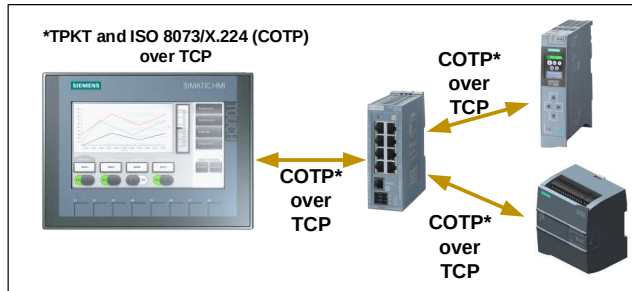


Figure 30. Established Communication between PLC Controllers and HMI

Communications between two PLCs are synchronous, bidirectional and full duplex. We observed that the default data exchange between the PLCs uses the COTP protocol. To communicate, the devices use the GET and PUT functional blocks offered by the TIA Portal API. In order to observe PROFINET traffic, we set up PROFINET IO using the TIA Portal API. The PROFINET IO exclusively relies on PROFINET RT and IRT communication protocols. Communication between two PLCs using PROFINET RT is illustrated in Figure 31. The only option provided through the TIA Portal API for RT communication is the I-device configuration. The I-device configuration allows partnership between high level and low level controllers. Through this partnership, memory areas called “transfer areas” are located in controllers and can be shared. The advantage of this is that a complex automation task can be separated into smaller tasks and can be distributed to partnered controllers.

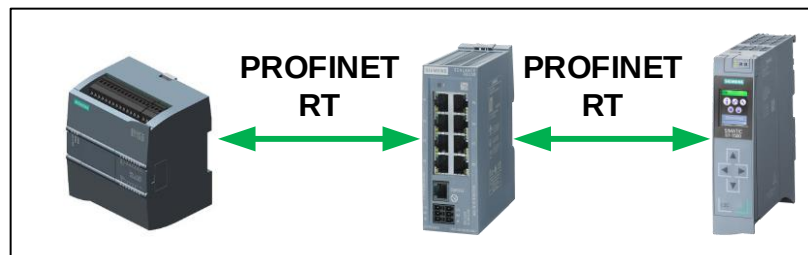


Figure 31. Established Communication between PLCs over PROFINET RT/IRT

Our goal is not to deploy a full PROFINET IO system but rather to observe the PROFINET protocol. In addition, we created transfer areas in both of the controllers so

we could observe how changes in the shared memory area affect the payload of the PROFINET PDUs; specifically, we looked for patterns. The shared memory area between the two PLCs was one byte. We also intentionally programmed PLC 2 to cyclically change this byte from 0x00 to 0x0f with a frequency of 2.0 Hz.

The content of two successive PROFINET PDUs that were captured is shown in Figure 32. The captured traffic is from the link connecting the network switch and PLC 1. The MAC addresses for the two PLCs (bytes 0x00 – 0x09) are highlighted by the green and purple shaded rectangles shown in Figure 32; the data is in hexadecimal format. The MAC address in the green rectangle is for PLC 2, while the address in the purple rectangle is for PLC 1. The yellow rectangle in the figure contains some PROFINET data fields that are appended to the end of the PDU. These fields are the cycle counter (bytes 0x38-0x39), data status (byte 0x3A), and transfer status (byte 0x3B). More information regarding the data fields is illustrated in Figure 42 in the Appendix. We see that the cycle counter is represented by the first two bytes, which cyclically changes from 0 to 65536 with step-size of 64; the cycle counter counts the number of cycles that have been completed successfully. One increment of the cycle is equivalent to 31.25 μ s. Since the cycle counter has a step-size of 64, the bus clock cycle is 2.0 ms (i.e., 31.25 μ s multiplied by 64); this corresponds to the value programmed initially through the programming API. As a result, we deduce that PROFINET RT packets are transmitted at a constant rate at intervals of 2.0 ms. Any disruption or delay in transmitting PROFINET PDUs can cause an interruption; Additionally, this interruption can cause an alarm which requires the devices to reestablish their PROFINET communication channel.

Apart from the cycle counter, there are also two other fields that indicate the status of the communication between the devices, the data status and the transfer status (bytes 0x3A and 0x3B). After the cycle counter bytes comes the data status byte, which remains constant in all the captured frames. The last byte (0x3B) is the transfer status, and this also remains constant. As seen in the figure, the remainder of the PDU is padded with zeros.

The only distinguishable differences in the two successive PDUs are the bytes in line 0010 (bytes 0x10–0x11). These two bytes correspond to the value that is stored by

and exchanged between the two PLCs through the configuration of the transfer areas. In our programming example, we intentionally programmed the PLCs to exchange a single byte so we could easily observe the effect of the byte change on the payload of the PROFINET PDU. The value of the byte is changed by PLC 2, switching from 0x00 to 0x0f cyclically. This cyclical change of the transfer area corresponds to a change in the payload portion of the PROFINET PDU and is clearly shown in Figure 32. The current captured traffic is from PLC 2 to PLC 1. Since it is full-duplex communication, traffic from PLC 1 to PLC 2 is similar to the traffic from PLC2 to PLC 1 except that the packets from PLC 1 have zeros padded as PLC 1 is not programmed to send/share byte to PLC 2.

Since timing is critical, the cycle counter values in the PROFINET PDUs are important for device synchronization. The last observation that can be made is that the minimum size of a PROFINET-IO PDU is 60 bytes, and the portion of the PDU not used for data exchange between PLCs is padded with zeros.

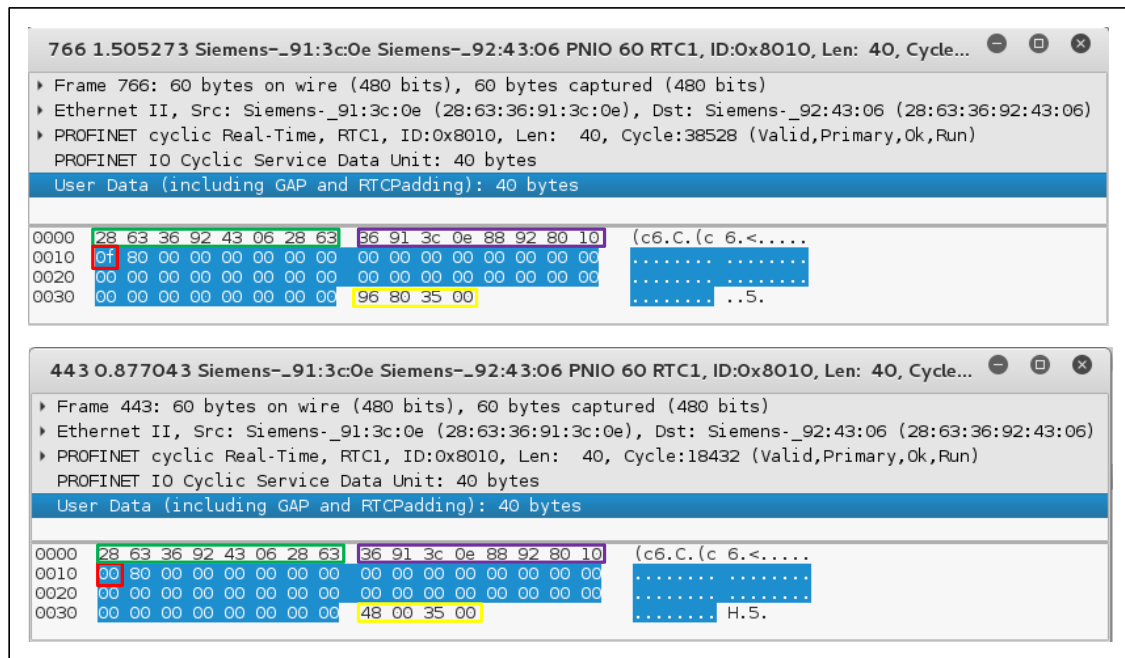


Figure 32. Two Subsequent PROFINET PDUs as appeared in Wireshark

Our observations of the capture traffic yield a lot of information, such as MAC addresses of the devices and mapped/shared data values between PLCs. In addition, we

observe that PROFINET does not use any field or mechanism to confirm the validity and integrity of the received frames. This opens the possibility for a replay attack, which we exploit later to see if there is any impact on the devices in the network. For example, if we keep sending similar frames, would the PLC be able to distinguish them from frames that were received from the original sender? Moreover, can we change a byte and replay the modified frame? Also, if we issue fake commands and modify monitored parameter values like temperatures and pressures, is the controller able to validate the modified commands?

Another experiment that we did was to examine the connection disruption and establishment of devices. While the PROFINET-IO RT was still established, PLC 2 was disconnected from the network switch. This was to examine the behavior of the devices, especially between the PLCs. We observed that both PLCs generate an error that was logged in their diagnostic buffer. Afterwards, we reconnected the PLC 2 to the network switch. After the reconnection, we observed that three messages were transmitted to reconnect the two PLCs. These three messages, sent over UDP, were captured and displayed in Figure 33. In this example, the IP address for PLC 1 is 192.168.100.3, and the address for PLC 2 is 192.168.100.4. Also, the messages are carried over UDP using the Distributed Control Environment/Remote Procedure Call (DCE/RPC) Protocol. They contain header fields that uniquely identify the connection establishment between the two PLCs; more information with regards to the message encapsulation can be found in the Appendix, Figure 47. As shown in Figure 33, PLC 1 accepts PLC 2's request for connection establishment and control as well as confirms PLC's control. (More information about the sequence of message exchange is provided in the Appendix, Figures 60 and 61). With this observation, we try to exploit the sequence of transmitted packets and perform a replay attack against PLC 1.

Source	Destination	Protocol	Length	Info
192.168.100.4	192.168.100.3	PNIO-CM	392	Connect request, ARBlockReq, IOCRBlockReq, IOCRBlockReq, ExpectedSubmoduleBlockReq, AlarmCRBlockReq
192.168.100.4	192.168.100.3	PNIO-CM	174	Control request, IODControlReq Prm End.req, Command: ParameterEnd
192.168.100.4	192.168.100.3	PNIO-CM	174	Control response, OK, IOXBlockRes Application Ready.rsp, Command: Done
Source	Destination	Protocol	Length	Info
192.168.100.3	192.168.100.4	PNIO-CM	232	Connect response, OK, ARBlockRes, IOCRBlockRes, IOCRBlockRes, AlarmCRBlockRes, ARServerBlock
192.168.100.3	192.168.100.4	PNIO-CM	174	Control response, OK, IODControlRes Prm End.rsp, Command: Done
192.168.100.3	192.168.100.4	PNIO-CM	174	Control request, IOXBlockReq Application Ready.req, Command: ApplicationReady

Figure 33. PDU Sequence for Reestablishment of PROFINET Communication between PLC 1 and PLC 2

Before describing the software tool developed for packet capture and network attack, we used some open sourced and publicly available exploitation and information gathering tools and examined their effects on our small-scale industrial testbed. The first tool that we used is Zenmap, a network scanning and discovery utility. It provided us some useful information, such as the network footprint of the industrial devices, network services, and flavor/version of the operating system running in the devices. All the scanning activities were performed via the industrial switch.

The scanning results using Zenmap are listed in Table 1 (nmap network scan command line arguments: nmap -sS -sU -p 1-65535 -T4 -A -v -Pn 192.168.100.0/24). For both of the PLCs, ports 80, 102 and 443 are opened for TCP and port 161 is opened for UDP. Since both of the PLCs are running a web server, ports 80 and 443 are also opened. The web servers can be activated or disabled through the TIA Portal API. Depending on the privileges of the login user of the web server, the user can access diagnostic buffers of a remote device, make minor adjustments to the device configuration, or disable it. TCP port 102 is used for TPKT and COTP traffic. Communication using this port, according to the TIA Portal software platform, is known as S7comm and is enabled when devices are exchanging data using the GET and PUT function blocks. Additionally, it is used to configure and access the online diagnostics utility of the devices. Also listed in Table 1, UDP port 161 is used for the SNMPv1 service. There is no indication that this port was activated through the API or which service or utility of the PLC makes use of this open port. Although Zenmap was able to

pick up some of the network services, it was not able to detect any of the UDP ports that are assigned for DCE/RPC. These are UDP ports 34962 to 34964 and should be opened since they are used for PROFINET messages.

Table 1. Transportation Layer Ports and Associated Services in PLC 1 and PLC 2

Device	Port Number	Service/Application
PLC 1/ PLC 2	TCP 80	HTTP for Web Server Application
PLC 1/ PLC 2	TCP 443	HTTPS for Web Server Application
PLC 1/ PLC 2	TCP 102	TPKT and ISO 8073/X.224 COTP for S7comm communication
PLC 1/ PLC 2	UDP 161	SNMPv1 server

As for the HMI, Zenmap was able to discover 23 ports. These ports are listed in Table 2; there are five TCP ports and eighteen UDP ports. TCP port 102 and UDP port 161 run the S7comm and SNMP services, which are the same as the ones run at the PLCs. Apart from these, the rest of the TCP and UDP ports, according to Zenmap, appeared to be opened or filtered or not associated with any particular service or protocol.

Table 2. Transportation Layer Ports and Associated Services in the HMI

Port	Service/Application
TCP 102	TPKT and ISO 8073/X.224 COTP for S7comm communication
TCP 34910, 49967, 51474, 60664	Unknown
UDP 161	SNMP
UDP 16918, 16939, 17946, 18449, 19181, 19402, 20419, 22778, 26040, 26077, 32125, 33682, 39337, 48982, 54038, 56809, 60109	Unknown

The scanning results of the network switch without any security settings are listed in Table 3. Ports 22 and 23 are used for remote configuration of the network switch, ports 80 and 443 are used for HTTP and HTTPS, respectively, and port 44818 is used by

the EtherNet/IP protocol. According to the manufacturer, this network switch also supports industrial devices running EtherNet/IP protocols. Lastly, ports 139 and 84 are used for the NetBIOS and Common Trace Facility protocols, respectively.

Table 3. Transportation Layer Ports and Associated Services in the Network Switch

Port Number	Service/Application
22	OpenSSH 6.6 (protocol 2.0)
23	LANDesk Remote Management
80	HTTP
84	Common Trace Facility
139	(Filtered) Netbios Session Service
443	HTTPs
44818	EtherNet/IP-2

Besides obtaining the footprint of the devices in the network, Zenmap was also used to discover the operating systems of the devices running in the network. It was not able to determine the operating system of the HMI but was able to determine with high probability the operating systems for PLC 1 and PLC 2. For PLC 1, the operating system was determined to be “Polycom VSX 7000e/8000” with an accuracy rate of 94%; for PLC 2, the operating system was determined to be “Extreme Networks Summit48si switch (ExtremeWare 7.1–7.8)” with an accuracy rate of 93%. We were able to use Zenmap to guess the operating systems but were not able to validate the results because we do not know the internally implemented software and hardware architecture.

Since Zenmap was not able to provide the operating system for all the devices in the network, we used Nessus. Nessus can be used in either “Basic Scan” or “Advanced Scan”; after scanning the network, Nessus had two important findings. First, it detected that the HMI is running Linux Kernel version 2.4. As for the PLCs, it just guessed that they are “Siemens Devices” without giving specifics of the operating systems. Second, it detected that both the PLCs have a vulnerability characterized as “Critical.” The vulnerability is related to “SNMP Agent Default Community Names.” As stated by Nessus, this vulnerability can be exploited by an attacker either to gain knowledge of the

system or to change its configuration. From what we know, this service might be related to the configurable service offered by the API software or embedded Web server; however, we were not able to relate this service to any utility of the PLCs. At the device, this service was password protected, and Nessus associated this vulnerability with the Common Vulnerabilities and Exposures (CVE) entry under Common Enumeration Platform name “cpe:/a:snmp:snmp”.

Subsequently, we tried to see whether the Metasploit framework and Armitage tool would provide us any clues about the devices present in the network. Initially, we performed a scan against the devices but no new findings were discovered since Metasploit actually utilizes nmap (Zenmap) scripts to perform the scanning. Additionally, no available attack vectors or exploits were provided. The only useful tool found in the Metasploit framework was an auxiliary module related to SNMP enumeration. This is a tool that is associated with the open UDP port 161 that was discovered by Zenmap and Nessus. This auxiliary module launched successfully against the PLCs and takes advantage of the vulnerability discovered by Nessus. The result was that in the Metasploit console we were able to see a complete description of the PLCs, including their exact type, firmware version, the available interfaces, assigned IP addresses, transport layer ports, established connections, and additional characteristics.

The next tool we used is “p0f” version 3.07b created by Michal Zalewski. It is a passive scanning utility that listens for packets over the network and makes inferences about the operating system of the devices in the network. The p0f tool was able to detect that the HMI is running Linux Kernel 2.2.x-3.x. This result correlates with the result from Nessus, but the detection of the Kernel version is different. As for PLC 2, the p0f tool was not able to make any inference but was able to infer that PLC 1 was running Windows 7/8 operating system. This result might be partially correct since Microsoft is offering solutions for Real Time Operating Systems (RTOS) for embedded devices. In addition, it is known that Windows CE is for other RTOS devices manufactured by Siemens AG.

The last tool we used was Ettercap version 0.8.2. It is a multipurpose network traffic sniffer and can be used for man-in-the-middle attacks. We used this tool to

perform a man-in-the-middle attack via ARP poisoning so we could observe all the PDUs that are exchanged between two devices.

We launched Ettercap three times. The first time it was used against the HMI and PLC 1. The second time, it was launched against the HMI and PLC 2. The third time was it launched against both PLCs. For the first two cases, we were able to perform ARP poisoning and MITM attack for all the protocols running over TCP and UDP including COTP and S7comm. As for the third case, although we were able to see all the traffic between the two PLCs, we were not able to observe any packets related to PROFINET; this is most likely due to the fact that PROFINET IO RT runs on top of data link layer.

C. HARDWARE DISASSEMBLY OF THE PLCs

Every approach to detecting vulnerabilities in systems should always go through the process of gaining expertise related to the internal hardware architectural implementation. Vulnerabilities that have been discovered and exploited in a computer system are often heavily reliant on complete intelligence about a system.

During a typical reconnaissance phase, an attacker steps through the process of gaining knowledge about the operating system and services that are running on a particular device. Furthermore, any source code related to the OS or the services is useful for detecting and triggering vulnerabilities. Finally, it is equally important that an exploitation of a vulnerability involves complete knowledge of the processor unit architecture and the instruction set that is utilized by a computer system.

If an attacker attempts to exploit a Linux-based computer system, then we assume the attacker has a great amount of information about the platform and probably knows that in most cases the target computer system is running on an Intel-compatible processor; if this is the case, the attacker can benefit from the information about the instruction sets Intel has released for developers. Additionally, an attacker will be able to obtain the source code of the Kernel of the operating system or all other binaries files that are related to services running on a Linux-based computer system.

In our case, there was no available information for the operating system, source code or their microprocessors'/microcontrollers' architecture related to the PLCs, the HMI or the network switch. In the previous section, we obtained some clues about our testbed devices and operating system but not enough to gain a complete understanding of the devices. In order to obtain more information, we moved one step further and disassembled one of the devices to see whether we could find any clues about the microprocessor/microcontroller. A disassembled PLC is shown in Figure 34. The IC indicated by the red arrow inside the rectangular area is the processor of the controller. Apart from this IC, some other peripheral components were recognized that can potentially help in our effort to understand the internal hardware architecture.

Below the highlighted component that we consider to be the processor in Figure 34 is another rectangular IC which is labeled with the number 2. Research found similar components in the market that matched the description of a 32-bit bus DDR SDRAM memory. This important finding helps to infer with high probability that the processor is a 32-bit architecture. The component right below the SDRAM labeled with the number 3 appears to be a NAND Flash memory. The component illustrated with the number 4 seems to be an Ethernet controller IC. Finally, the component labeled with the number 5 is a real-time clock.

The interesting findings related to the hardware disassembly are the three sockets highlighted with the orange rectangle. Numbers 7 and 8 are sockets that are utilized for communication between the CPU and other peripheral modules that can be connected adjacent to the PLC at the same DIN rail. These can be expansion modules for a general purpose input/output or communication module (i.e., for support to serial communication protocols). Socket 6 is not visible unless the front cover of the PLC is removed. The important observation is that in all these sockets, we are able to track the copper traces that connect these sockets with the pins of the processor. Not all of the pins are used or connected. Some of them are connected to the ground or power supply, and we can distinguish them due to the thickness of their traces. The thin traces seem to correspond to signal pins. Utilizing a logic analyzer with a protocol decoder and observing the traffic on these sockets might allow to us to gather some information.

In conclusion, open source intelligence gathered through the Internet only partially succeeded. We were unable to find any IC datasheet after performing an extensive search using the identification number printed on the IC that we considered to be the processor component. As a result, we cannot make any safe assumptions about the processor architecture; however, we were able to recognize several other peripheral components. With the use of appropriate electronic equipment, like logic analyzers, further intelligence gathering about the system is still possible.

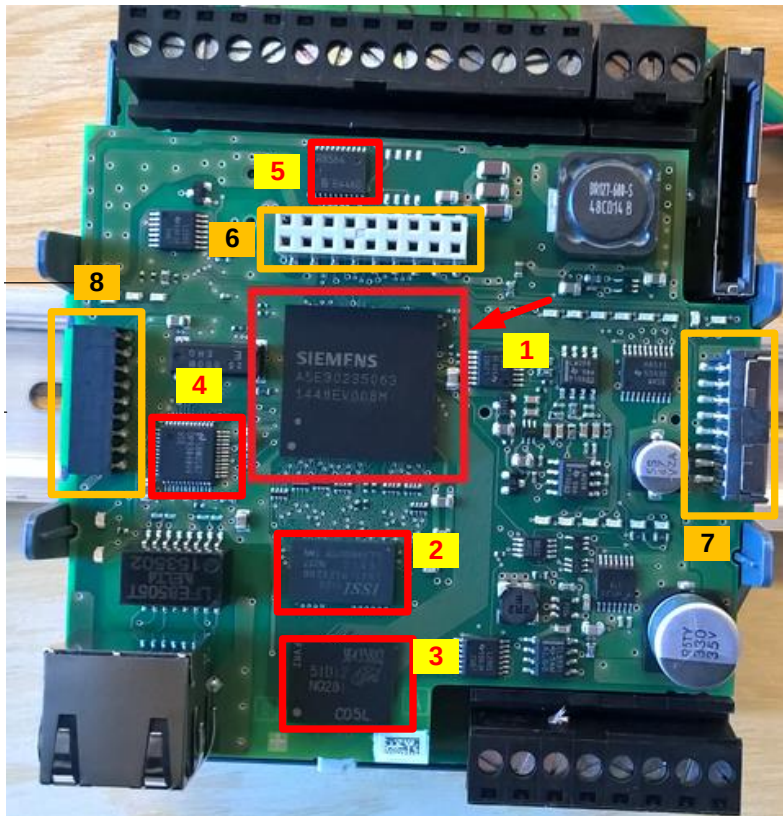


Figure 34. PLC s7-1200 View with Front Cover Removed

THIS PAGE INTENTIONALLY LEFT BLANK

VI. SOFTWARE TOOL FOR FUZZING AND TESTING

Many approaches can be considered for vulnerability detection. One can discover software vulnerabilities by interacting directly with a device; interaction with the device is possible through the communication protocol. This is the basic idea for creating a tool that can build custom-made PDUs that might trigger vulnerabilities inside an industrial device. Tools that enable packet crafting and protocol fuzzing are not new; additionally, there are many open source tools that can be used for fuzzing or crafting custom PDUs. The novel approach presented in this thesis research is to build a platform that incorporates multiple Industrial Ethernet-based communication protocols.

Apart from finding vulnerabilities, this fuzzing and packet crafting tool can be used for protocol discovery and reverse engineering. It can inject packets into the system, observe a response and make inferences about the behavior of the device or the protocol. This is extremely useful, especially for cases where a protocol or device specification is not publicly available. In addition, many embedded systems, apart from their normal operation, are able to support other operations which are unknown and intentionally hidden. Some examples of hidden functionality are covert channels or non-legitimate behavior upon reception of a specific pattern of a signal. A fuzzer might help discover such hidden functionalities in a device.

The development of the proposed multipurpose software tool was exclusively based in the C programming language. C is supported by most operating systems and also can be easily deployed on a large variety of hardware platforms. In addition, it can be used for low level packet handling. Also, C is a compiled language; thus, we expect to perform faster than other programming languages like Python, which is interpreter based.

A. SOCKET PROGRAMMING, LIBNET, AND LIBPCAP LIBRARIES

The Unix and Linux operating systems offer great capabilities for programming in C. These operating systems incorporate a variety of system calls and C functions in user space libraries that allow us to do low-level handling of PDUs transmitted by the network interface card. The term “low-level” refers to the ability to control the values of the fields

in the headers at different levels of the TCP/IP protocol stack during the encapsulation process. At an abstract level, we can literally convert the network interface card of a computer running on a Linux or Unix based operating systems to a user-controlled transmitter that sends data over the wire. This is not always the case as by default the kernel of the operating system has the absolute control of the packets that are received and transmitted by the operating system.

The most fundamental system call that provides us this functionality is the `socket(int domain, int type, int protocol)` function. This system call “creates an endpoint for communication” [41]. Once again at an abstract level, the socket and socket file descriptor that is returned as a result of this system call may be likened to the creation of a door in a computer that can be used for process communication within the same computer or between processes that are running on different computers in a computer network. A process is an entity that is responsible for executing instructions on behalf of a user. These instructions are part of a program that the user creates. Utilizing this door, we observe that the processes are running on two endpoints can exchange messages, which we have called PDUs. The size, the contents, the type, and the number of PDUs exchanged between the processes are determined by the communication protocol and the specifications of the program or the application.

The key point is that, depending on the arguments provided in the socket function, we can achieve a different level of control over the data that are written to the PDU. As discussed in Chapter III, the way in which PDUs are built is based on encapsulation, where headers are appended to the payload of the data we want to transmit. By default, all applications and processes need to encapsulate their data. Each of these processes, under the coordination of the kernel of the operating system, builds its own packets and transmits them through the NIC of a computer. Essentially, each of these processes needs to make a system call of the type `socket( )`, as well as many other related system calls in order to establish a communication channel.

As shown in Figure 35, the way in which the arguments are passed to the `socket( )` system call allows us to achieve a more granular control of the data that are written in the PDUs. At the top level, `socket(AF_INET, SOCK_STREAM/DGRAM,0)` provides the

most limited control. This type of socket creates an Internet Protocol (IP) socket as described by RFC 1112. The `SOCK_STREAM` or `SOCK_DGRAM` options are used to create sockets that utilize the TCP and UDP protocols, respectively. The user in this case can control some of the fields in TCP or UDP protocol headers and the payload included in layer 4 and above.

With the system call `socket(AF_INET, SOCK_RAW, protocol)`, the user application is given an IP socket. In this case, the user can specify all headers and payload contained in the PDU from layer 3 and above. This means that the user can totally ignore TCP and UDP headers and introduce his or her own protocol headers or any kind of payload. In the third version, the control of the PDU is more privileged; thus, the user from layer 3 and above can introduce his or her own headers, protocols, and payload, not necessarily including the IP protocol. The most granular control of the packet is provided by the fourth system call, which provides the user full control of the contents of the packet. The user, depending on the maximum transmission unit size, can introduce either random data or build his or her own stack of protocols.

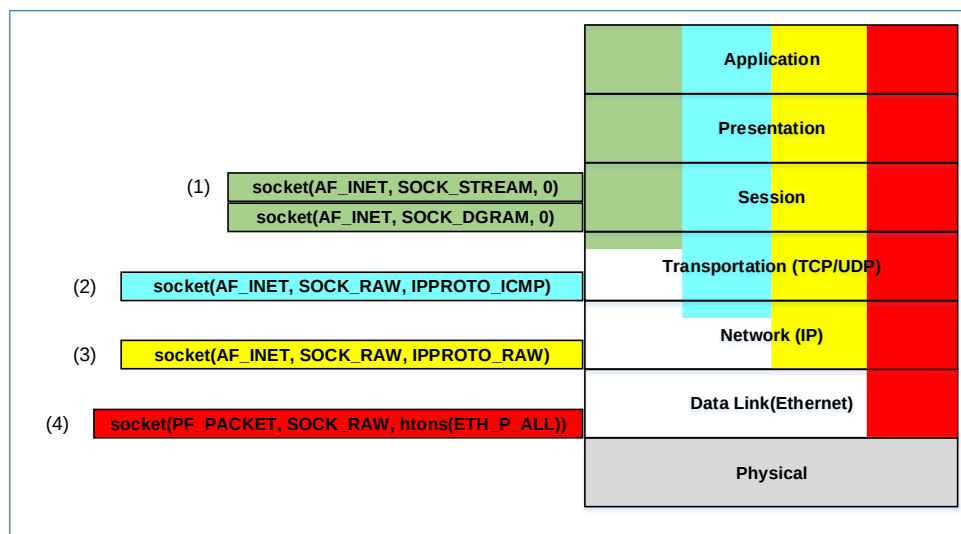


Figure 35. System Call `socket()` Control over PDU Headers.
Adapted from [42].

So far, all the industrial protocols presented are based on the Ethernet; thus, using the system call `socket( )`, we obtain full control of the packet contents, including the data link layer. The data link layer is based on Ethernet 802.3 or 802.1q standards. If we can control the contents of the packets, then we can incorporate into these packets all the data structures that refer to classes of messages described in Chapter IV and in the Appendix. As a result, we can craft and inject packets without any restrictions as the kernel has provided the user enough privileges to manipulate the contents of the packets.

Despite the `socket( )` system call and related function calls that are provided in the Unix and Linux operating systems, some tools and libraries have been developed to simplify and speed up the process of building and receiving packets through the NIC. Examples of such libraries include `libnet` and `libpcap`, which are written in the C programming language. The `libnet` library was developed by Mike Schiffman and is used for packet injection. The packet capturing library, `libpcap`, was first developed by the Lawrence Berkeley National Laboratory.

Both libraries utilize the system and function calls provided by the Unix and Linux operating systems. Based on these libraries, they create an abstraction layer. This abstraction provides utility functions that allow users to create programs that perform low-level packet handling.

A block and flow diagram shown in Figure 36 depicts the typical process for building a complete PDU and transmitting it through the NIC. The `libnet_init( )` function is used for initialization of the device (i.e., network interface) and the level of control that the user desires to have over the contents of the packet. The sequence of the function calls is important. The user must call the `libnet_build_*` functions in the order in which the encapsulation process is actually taking place, starting from the payload and going from the top to the bottom layers. At the end, the `libnet_write( )` function writes the PDU on the wire through the NIC. The same process can be repeated as many times as necessary or can use a terminating condition. In the end, the `libnet_destroy( )` function should be called for closing the socket associated with the network interface device and then free all the associated data structures that were created during the `libnet_init( )` function call.

A similar block and flow diagram related to the capturing process using libpcap functions is shown in Figure 37. In a similar manner, libpcap should at the beginning initialize the network interface. Then, using a series of functions, we can apply several filters to the traffic arriving at the interface in order to limit the number of packets we want to examine further. After that we can either examine each packet individually or introduce a loop. Inside the loop, assuming that we have previously created a file using the `pcap_dump_open ( )` function, we can dump each received packet that matches the specified filter into the opened file. This file will be in a format of .pcap file, which can be opened with the Wireshark program or other libpcap functions for further processing of contents.

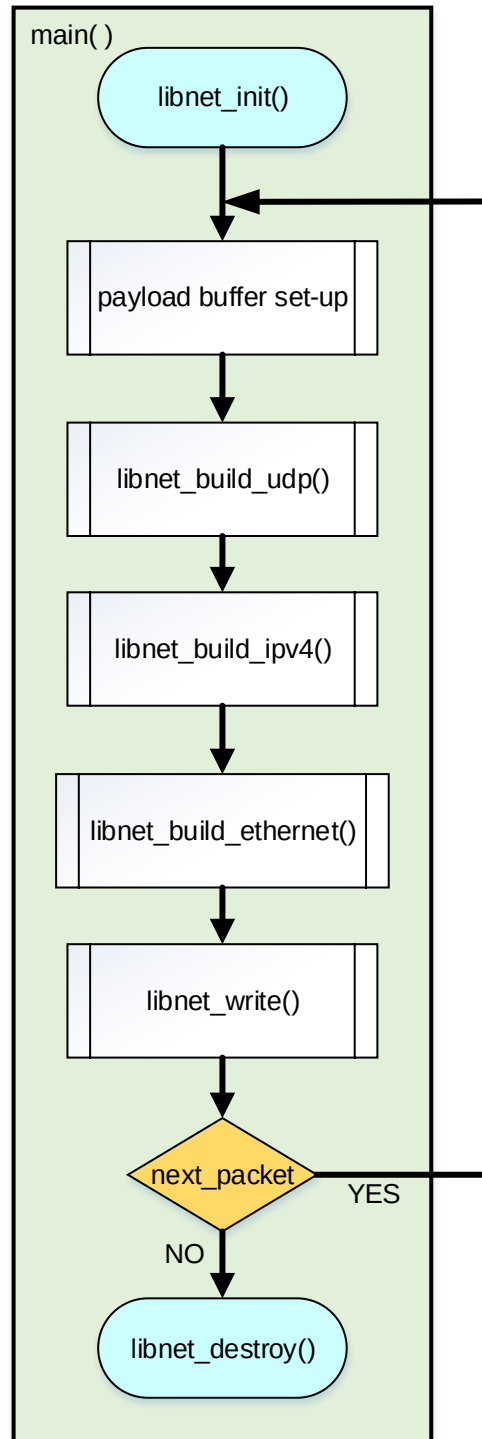


Figure 36. Process Flow for Custom-made PDU using C Libnet Utility Functions

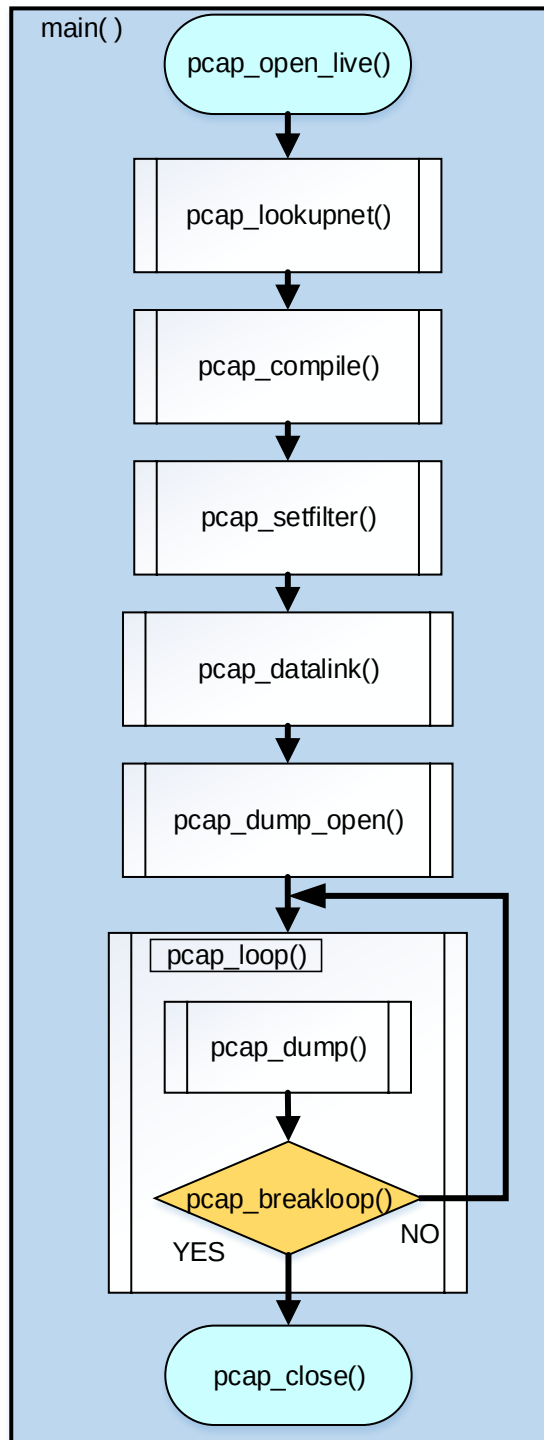


Figure 37. Process Flow for Initiating PDU Capturing with C Libpcap

B. HERMES: THE INDUSTRIAL ETHERNET PACKET INJECTOR

The raw socket programming presented so far is used for the development of the proposed software tool, which we call HERMES IECPPi (Industrial Ethernet Communication Protocol Packet Injector); this tool is simply referred to as Hermes for the remainder of this thesis. Although there is another similar library, Scapy, built into the Python programming language, for this particular task we have chosen to build the industrial protocol stacks using the C programming language. We have done so for two reasons. The first is the capability of C language to run on numerous development platforms and boards. The second is related to speed and performance issues. As mentioned earlier, the industrial communication protocols should meet the requirements for certain time constraints. As a result, any tool developed for packet injection in industrial equipment should also conform to the performance requirements of the industrial communication protocols. These requirements are related to the deterministic behavior of the control systems interaction, bus cycle time, jitter, and clock synchronization.

At the time of this thesis research, most of the industrial protocols investigated here were proprietary and no access was provided to the description, source code, behavior, and specification of the protocols. All classes of messages associated with each protocol which are incorporated in the HERMES for the PROFINET case were derived by observation of captured traffic from the testbed. For all other protocols, like Ethernet/IP, EtherCAT, and Modbus/TCP, we used the descriptions provided by references discussed in Chapter IV.

The operation of HERMES does not imitate the behavior of the protocol. We intentionally attempted to create a tool to test industrial devices under malformed or specially crafted PDUs. In other words, HERMES cannot interact with any industrial devices; however, it can inject specially crafted packets. Its operation is closer to a fuzzer, and the user is responsible for formulating these packets during HERMES start up. In the future it could be possible, if access to the specifications is obtained, for somebody to enhance the C programs with features that imitate the behavior of the protocol.

HERMES was developed to incorporate fuzzing capability for all Ethernet-based industrial communication protocols in one software platform; however, this was not the only case we had to consider. During experimental programming and interaction with the testbed, as discussed in Chapter IV, we discovered many other protocols that we usually observe in IT networks apart from the PROFINET industrial communication protocol. Some examples of these are the TCP, UDP, ARP, LLDP, STP, and DHCP. The logical inference is that industrial equipment communication capabilities are not limited to the relevant industrial communication protocols. Instead, their software design incorporates several other protocols. The question then became whether HERMES should also incorporate these protocols since it is a device-centric tester. The answer is that a complete test of an industrial device must go through the process of evaluating the implementation of each of these protocols. Only then can we be sure about the reliability of a test procedure and the detection of any vulnerabilities in the device. The current implementation of HERMES does not consider all of these protocols. The focus area might be the Ethernet-based industrial communication protocols, but since the industrial devices appear to implement other protocols too, in future work we should consider them.

C. HERMES SOFTWARE DESIGN DESCRIPTION

HERMES is entirely based on C socket programming and the C libnet and libpcap libraries. Its development was done in the Kali Linux operating system; however, it is possible to operate in any Linux-based operating system equipped with the gcc compiler and the dependencies related to the C libnet and libpcap libraries. Because of the low-level packet formatting, privileged access to the OS is required to operate this tool.

The HERMES packet injector operates at a command line interface for interaction with the user. During development, we considered four major objectives. The first was ease of use from the user's perspective. This concept allows users to experiment with and educate themselves in the Ethernet-based industrial communication protocols. The second was related to flexibility of the tool to perform several tasks and support various strategies for packet injection. In this way, users are able to perform several kinds of attack scenarios. The third consideration was to provide future developers with the ability

to expand HERMES' capabilities by incorporating new protocols or protocol behavior emulation. The final consideration focused on optimizing performance related to packet injection speed. This last requirement was set for making HERMES capable of meeting the requirements of the RT and IRT classes of industrial communication protocols. The last requirement is the main reason that error checking was kept to a minimum in many parts of the program.

HERMES can be initiated from any command line terminal. The program execution is initiated by simply executing the command `./hermes`. For this to be done, all source code files that are included in the directory need to be compiled using the `gcc` compiler with the flags `-lnet` and `-lpcap`. These flags inform the linker to link the functions included in the source code files with the library functions contained in the `libnet` and `libpcap` libraries. All files, including binaries, should be under the same working directory.

The principle of operation of the HERMES packet injection tool is illustrated in the block diagram in Figure 38. A startup process provides the user with an elementary command line interface. This process is responsible for starting HERMES and printing the menu options. The options include the type of the protocol with which the user wishes to experiment. After selection of the protocol, the process provides the user with an interactive command line interface. The purpose of this interface is to give the user the ability to formulate at a granular level the values of the fields in each of the headers contained in the selected protocol. The user interface process is also responsible for sanitizing and checking the validity of the input values. In case of failure caused by erroneous argument, input check validation provides informative messages. In the case of successful reading of valid arguments, it sets up an array of pointers to character strings. These strings are the previously read input arguments and reflect the options that were configured by the user. These arguments also determine all parameters and fields of the packets that will be injected.

In HERMES, each of the protocols is independently implemented in a separate file. Each file contains the source code for packet construction and injection related to the protocol that was selected. In addition to this, each file contains source code for capturing

capabilities. The thought behind this feature is that we always want to be able to capture the response to our generated input. In other words, we stimulate the system and then capture the response to see the effect. As shown in Figure 38, after the user interface process has read all the input arguments, it invokes the `execve( )` system call. The effect of the `execve( )` can be likened to the current process replacement or current process transformation. The result is that the current process becomes a new process. The new process corresponds to the process that is responsible for performing packet injection for a specific protocol. All possible process transformations are illustrated in colored boxes of Level 2 in Figure 38.

The system call `execve( )` invocation requires three arguments (i.e., `int execve(const char *filename, char *const argv[], char *const envp[])`). The first argument is the filename of the binary that is to be executed. As shown in Figure 38, every protocol, after the compilation of all source files, has a separate binary file. The user interface process creates the second argument, which contains a string representation of the user input selections described in the previous paragraph. The third argument contains the environment variables. Each time only one process replacement takes place based on the user's protocol selection. This implementation allows future developers to add extra binaries if there is a requirement for adding new Industrial Ethernet-based protocol variations or types of messages.

At the last level, the newly created process of Level 2 generates two new processes. The Level 2 process is the parent process, and the newly created processes are the child processes and belong to Level 3, as illustrated in Figure 38. To achieve the generation of the new processes, we use the `fork( )` system call twice. This system call creates two processes that are exact copies of the parent process; however, each of them executes a separate task. As shown in Figure 38, one of them is responsible for creating the PDUs based on the specification previously entered by the user. The second process is exclusively responsible for setting the network interface in promiscuous mode and capturing all the traffic generated by the “injecting” process or the device that is under test. The program is then terminated when the packet injection process has finished

injecting the specified packets, when an interrupt signal (SIGINT) is received, or when an error occurs in any of the three processes (i.e., parent or the children).

Upon completion, the capturing process generates one or several files that contain all the captured network traffic in an appropriate .pcap format. These files contain the packets that were injected by the host machine or the packets that are sent by the industrial device under test. These files can be opened later for further processing by utilizing Wireshark or other packet analysis tools.

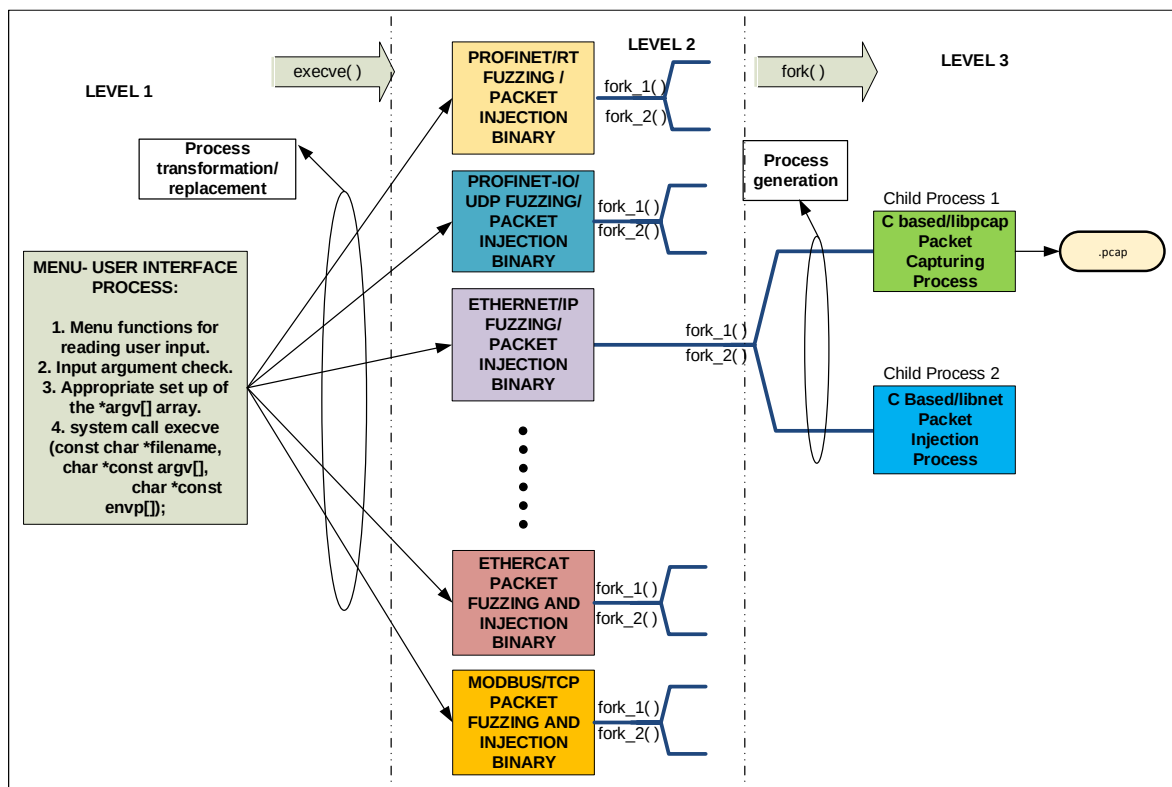


Figure 38. HERMES Structural and Functionality Description Block Diagram

D. HERMES CAPABILITIES

HERMES incorporates many different functions. This enhanced functionality was developed to allow a user to perform several attack scenarios against industrial communication protocols and control systems.

The development and software design process had to deal with two major challenges. The first one was the selection and deployment of the most appropriate randomization scheme. The second one was related to the development of source code that would run fast enough and allow us to interfere in the communication channel established between the industrial devices.

In a protocol fuzzing operation, we are not usually interested in the validity of the input provided to a device. We can feed a device any input we think might cause a crash, restart of the device, or produce a non-recoverable error. There is no way for someone to determine a priori what can cause strange behavior in a device, especially when there are limited clues about the architectural design of the device under test; however, in many cases it is desirable and more efficient to probe the device under test with a ‘mostly’ valid input. This input leads the device to believe that something legitimate was received. The device then attempts to process the contents of the malformed input, and we may observe the actual effect on the device. An example is the creation of an IP protocol packet. In this case, it is less efficient, and maybe pointless, to fuzz the IP destination field (32-bit quantity) of the IP protocol header. The device under test in this case trivially discards any packet that does not contain a destination IP address that references the device. In this particular case, we wish to keep some fields of the packet constant and cyclically randomize other fields of the packet.

The randomization process is not always an easy task. For field values included in the protocol headers that are up to a 32-bit size, brute-forcing is a relatively easy task and can be done by simply counting from 0 up to 2^{32} ; this represents the number of all possible combinations of the 32-bit value; however, when we need to perform all possible combinations for many fields and generate all possible permutations of these fields, the number of unique inputs becomes impossibly large. The amount of time required to test

all possible inputs is prohibitive in such cases. There is also the case where certain fields in a protocol header are greater than 32 bits. For example, some unique identifier fields required in some protocols may be as large as 16 bytes (128 bits). The number of unique values to generate in this case would take a lifetime to generate.

In HERMES, two different approaches deal with these difficulties. The user has the option to set certain fields as constant or variable. The user interface process, as discussed in Section C in this chapter, asks the user to determine which fields have a default constant value and which of the remaining fields are changing cyclically. For fields that are up to 64 bits, the brute-force approach is utilized. In other words, for field values with an 8-byte length, a simple counter implements all possible combinations. For fields that have a greater than 8-byte length, a random value is assigned. In the ‘computer’s world’, having a true random generator capable of providing high entropy values for injection is difficult. In this case, we used the `/dev/urandom` device that is available in Linux or Unix based operating systems. This virtual device can be opened and read as any other file and is considered a good pool of randomness that can be read by a program. HERMES provides the user with an additional option to specify his/her own input file (e.g., in case the user has available an input file with high entropy randomized content or a specially crafted payload).

Apart from the option allowing the user to specify the contents of the packet depending on the protocol, HERMES incorporates some extra auxiliary functions. One of them is the capability to replay traffic. With this feature, the user can replay any available traffic previously captured during the interaction of two devices. The user interface process that initiates HERMES asks the user to provide the name of a single file or several files that should be replayed. The user can also specify the order in which to replay the .pcap files. Additionally, the user can specify the time delay between transmitted packets (e.g., transmit a packet every 2 ms).

Furthermore, when previously captured traffic is available, another feature of HERMES allows the user to replay the traffic after it has been modified. The user has the option not only to replay the captured traffic, but he or she can apply a randomization ‘mask’ on top of the packet’s contents. If we assume that every Ethernet frame is an array

of 1514 bytes, the user can specify which individual bytes or range of bytes to assign a random value. HERMES dynamically assigns random values to the bytes specified and replays all captured Ethernet frames in the sequence specified by the user.

In the HERMES implementation, packet injection occurs simultaneously with packet capture, as discussed in Section D in this chapter. This capability requires a significant amount of space to be reserved for storing the captured traffic. To deal with this requirement, HERMES breaks down the captured traffic into small files of 250,000 packets. This feature makes the handling of captured network traffic easier in case further analysis of captured traffic is required.

E. TESTING HERMES AGAINST TESTBED DEVICES

During experimentation with HERMES we tested several concepts. All experiments were performed against the testbed. The tests were performed by injecting packets either individually into each device or from within the network setup through the network switch.

To make the process of fuzzing efficient, two different approaches were applied. In the first approach, we tried to generate traffic by specifying step-by-step the contents of the packets utilizing HERMES auxiliary functions. In the second approach, we took advantage of the captured traffic as discussed in Chapter V and used HERMES to replay the captured packets either unmodified or by applying the randomization mask discussed previously. It was shown that the second approach was faster to apply. To apply the second approach, we first had to utilize Wireshark to generate files in an appropriate format that could be opened and read by HERMES functions. With Wireshark, we took all available captured traffic and applied several kind of filters offered in Wireshark to isolate traffic based on protocol description, IP addresses, MAC addresses, or the contents of the packets. In the next step, we exported the filtered traffic into pcap save files. Then by invoking HERMES, we attempted several approaches for injecting the captured traffic.

In the first injection scenario, we tried to attack the communication channel established between PLC 1 and PLC 2. As discussed in Chapter V, Section B, there were

two kinds of protocols observed in the communication between PLC 1 and PLC 2. The first one was COTP (based on the GET and PUT function blocks) and the second one was PROFINET IO RT (based on I-device configuration). Having attached the injecting interface to the network switch, we launched HERMES and selected the PROFINET IO protocol. The purpose of this scenario was to examine whether it was possible for a device that is not associated with the established channel to inject non-legitimate traffic. Furthermore, we needed this approach so that we could observe what the devices' behavior would be in this case. Our observation was that any attempt of traffic injection with packet transmission rate slower than the normal traffic (i.e., a packet every 2.0 ms) does not have any significant impact in the established communication between the PLCs. Further lowering of the transmission rate (e.g., a packet every 20.0 ms) does not even allow us to observe any incoming packets in the interface of either of the PLCs; thus, in order to interfere in the prioritized traffic channel, we had to adjust HERMES to inject a packet every 10.0 ms or less.

Once we were able to inject our PROFINET IO RT frames, we started manipulating the contents by applying a mask in the PROFINET payload portion that was related to the transfer area exchanged between the PLCs. The result was that through the alteration of the single byte quantity exchanged between the PLCs, we could partially control the relay output contact associated with the single byte. This made the relay output contact of the PLC open or close in a random and abnormal fashion. Apart from this, any further increment of the transmission rate caused a desynchronization of the devices and, finally, loss of communication between the devices.

In the next scenario, we tried to see whether a replay attack would be successful against the PLC 1. To achieve this, we created a large packet capture of PROFINET IO RT packets. We utilized the PROFINET IO over DCE/RPC packets that were sent by PLC 2 to PLC 1 for establishing the communication between the PLCs, as shown in Figure 39. This sequence of packets was described in Chapter V, Section C, and this sequence of packets can be obtained either during the startup of the devices or after disconnecting and reconnecting PLC 2 or PLC 1 to the network switch. The exact sequence of packets sent by PLC 2 is illustrated in Figure 60 of the Appendix. In this

experiment, we completely disconnected PLC 2 from the network. In the diagnostics buffer of PLC 1 and PLC 2, we saw an alarm related to the loss of communication between the PLCs. At the same moment, having disconnected PLC 2, we attached our injecting network interface on the switch. We initiated injection, with HERMES injecting a packet every 1.9–2.0 ms. The result was that for a limited amount of time (approximately 10.0 s) we could completely eliminate the alarm on PLC 1 by emulating that PLC 2 was still connected, while our injecting machine was the one that was actually sending packets to PLC 1. In addition to this, the relay output contact associated with the shared transfer area remained closed due to our replay attack.

Source	Destination	Protocol	Length	Info
192.168.100.4	192.168.100.3	PNIO-CM	392	Connect request, ARBlockReq, IOCRBlockReq, IOCRBlockReq, ExpectedSubmoduleBlockReq, AlarmCRBlockReq
192.168.100.4	192.168.100.3	PNIO-CM	174	Control request, IODControlReq Prm End.req, Command: ParameterEnd
192.168.100.4	192.168.100.3	PNIO-CM	174	Control response, OK, IOXBlockRes Application Ready.rsp, Command: Done

Figure 39. PROFINET over DCE/RPC Connectionless Protocol Packets

After the first 10.0 s, we were unable to control PLC 1 further. At this point, we were unable to fully explain why the replay attack failed to go through the entire capture we had replayed; that said, we were able to make a useful observation, which was that the PLC 1 sent an ALARM over PROFINET IO, for which the breakdown is shown in Figure 40. In the same figure, we also observe that a PROFINET IO frame was used to carry the alarm, that it was sent over IEEE 802.1Q, and that a hex value of 0xfe01 was specified in the frame ID field to signify the alarm. The complete description of the alarm is highlighted with the red arrow. This error is associated with an “AR-RPC-Control error,” but no further information or reason is provided for the error [43]. We can assume, though, that the main cause for this error is the bad coordination of the cycle counters between the devices, which are included in the exchanged PROFINET IO packets.

```

▷ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
▷ Ethernet II, Src: Siemens-_92:43:06 (28:63:36:92:43:06), Dst: Siemens-_91:3c:0e (28:63:36:91:3c:0e)
✦ 802.1Q Virtual LAN, PRI: 5, CFI: 0, ID: 0
  101. .... = Priority: Video, < 100ms latency and jitter (5)
  ...0 .... = CFI: Canonical (0)
  .... 0000 0000 0000 = ID: 0
  Type: PROFINET (0x8892)
✦ PROFINET acyclic Real-Time, Alarm Low, ID:0xfe01, Len: 40
  FrameID: 0xfe01 (Real-Time: Acyclic PN-IO Alarm low priority)
✦ PROFINET IO Alarm
  AlarmDstEndpoint: 0x0000
  AlarmSrcEndpoint: 0x0010
  ✦ PDUType, Type: ERR-RTA-PDU, Version: 1
    .... 0100 = Type: ERR-RTA-PDU (0x04)
    0001 .... = Version: 0x01
  ✦ AddFlags, Window Size: 1, Tack: 0
    .... 0001 = WindowSize: 1
    0000 .... = TACK: 0x00
  SendSeqNum: 0xfffe
  AckSeqNum: 0xfffe
  VarPartLen: 0x0004
  ✦ Status: Error: "RTA error", "PNIO", "RTA_ERR_CLS_PROTOCOL", "AR RPC-Control error (RTA_ERR_ABORT)"
    ErrorCode: RTA error (0xcf)
    ErrorDecode: PNIO (0x81)
    ErrorCode1: RTA_ERR_CLS_PROTOCOL (253)
    ErrorCode2: AR RPC-Control error (RTA_ERR_ABORT) (24)

```

Figure 40. PROFINET-IO Alarm Message Packet

In addition to the previously described scenarios, several fuzzing techniques were tried, as well as stress testing of the communication links and interfaces of the devices. In the latter case, we observed that generation of an excessive amount of traffic inside the PROFINET network caused a corruption and denial of service to all the devices. In the case of the HMI screen, an excessive amount of traffic generated against its interface (i.e., packets associated with the IP and MAC address of the HMI) caused a complete loss of all the indications/monitoring parameters and the control capabilities that the operator has through the HMI. For the PLCs, the excessive amount of traffic desynchronized the devices in a way that PROFINET IO communication could not be reestablished between the two PLCs. This might not actually be a realistic scenario of an attack, as the testbed setup would normally expect to be isolated and protected behind several layers of defense (e.g., firewalls, IDS); however, we were able to observe what could happen if a malicious software managed to intrude in this isolated platform. For the former case, no significant observations have been made, so we can confirm that specially crafted packets can cause a software ‘crash’ or stop of the devices. We were able, though, to replay certain kinds of

messages, such as the alarm presented previously. With the PLCs not being able to identify whether this message is legitimate, they would accept the message, causing an interruption of their service.

Overall, apart from the scenarios of traffic injection tested, no exploitable vulnerabilities were discovered. Nevertheless, even if any were present, we would still not be able to exploit them further as we had no information regarding the instruction set or the operating system of the devices.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND FUTURE WORKS

The purpose of this research was to perform an exploration into the principles of operation of some of the most common Ethernet-based industrial communication protocols, to obtain integrated reconnaissance on the operation of industrial automation devices, and finally, to assess the vulnerability of these devices, their operating systems, and their network communication protocols. This investigation relied to a large extent on study of the models, methods, and the classes of messages utilized by the industrial devices and their related communication protocols.

Apart from the theoretical approach, part of the contribution is the “hands on” investigation of actual industrial equipment. For this investigation, we used Siemens AG industrial automation equipment to build a small-scale testbed that was able to simulate the monitoring and control of a hypothetical industrial process.

During the testbed setup and development, we properly programmed the industrial devices in order to make full use of their functionality and their communication capabilities. This allowed us to test several exploitation tools, observe the behavior of the devices and gather information with regard to their network footprint or software design. Furthermore, we were able to investigate and reverse engineer the PROFINET communication protocol as well as other protocols that are used in communication between Siemens AG devices.

Another contribution is the development of a software tool dedicated to performing the injection of malformed packets relevant to the industrial communication protocols that were investigated. This tool was built using the C programming language and provides a command line interface and an enhanced functionality for users to test several scenarios of attacks. The tool was tested and functioned satisfactorily.

For future work, we highly recommend further expansion of the testbed and development of the HERMES software tool. So far the testbed simulates an environment at the fieldbus level; thus, a possible horizontal and vertical expansion of this testbed is required. This kind of expansion is graphically illustrated in Figure 41. The horizontal

expansion includes the incorporation of other fieldbus networks that employ other Ethernet-based industrial protocols like EtherNet/IP, EtherCAT, and Modbus/TCP. Additionally, each of the fieldbus networks should be enhanced with more advanced functionalities like motion control, distributed controllers, and distributed general purpose IO. For the interconnection of these components, proper gateways are also required.

In the vertical view, the fieldbus level should be expanded to include a control level and operational level. These levels can be enhanced with PC-based PLCs or software controllers capable of performing the role of supervisory management and configuration workstations. The platform could also incorporate data historians and business information consoles to make the testbed more realistic. Finally, at the very bottom level, an extra layer of communication known as the Actuator Sensor Interface (AS-i) can be added.

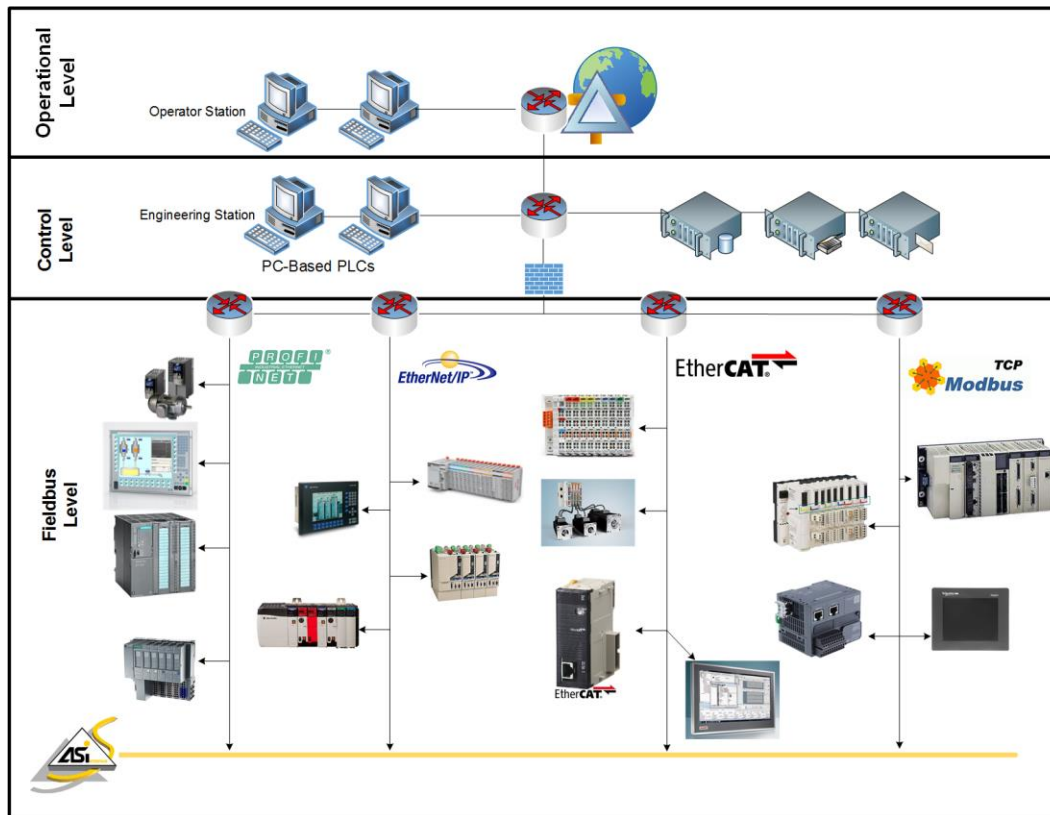


Figure 41. Testbed Vertical and Horizontal Expansion

As the era of the Industrial Internet of Things (IIoT) approaches, research should focus on several concepts that require further investigation. One concept is the capability of Ethernet-based technologies to handle the huge expansion of the devices' networks. Moreover, wireless technologies and protocols based on IEEE 802.15.4 and IEEE 802.11 are expected to be introduced in the industrial environment, and this introduces additional challenges in maintaining the security and safety of industrial and factory automation systems.

With regard to HERMES' capabilities expansion, a large effort is required to enhance its functionality. This expansion should include the incorporation of more protocols and classes of messages that are supported per protocol. Additionally, it would be useful for implementation of HERMES in an RTOS since HERMES needs to interact with devices that utilize RTOS. Furthermore, better randomization schemes should be introduced to make the fuzzing operation more efficient. For the purposes of this study, efficiency has been defined as the likelihood of detection and triggering vulnerabilities in industrial devices.

Finally, greater effort is required to gain knowledge of the industrial devices' internal hardware and software implementation and an in-depth knowledge of the communication protocols' source code implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. SUPPLEMENTAL MATERIAL

Table 4. Equipment's Description Utilized for Testbed

Device Name/Description	Device Technical Features
Programmable Logic Controller : Type CPU 1212C AC/DC/Rly (Part No. 6ES7 212-1BE40-0XB0)	Firmware version V4.1, Work memory 75 kB, power supply of 120/240 VAC, eight digital inputs with 24 VDC operation, six relay based digital outputs, two on board analog inputs, four high-speed counters, 4 pulse outputs on board, performance 0.04 ms/1000 instructions, on board PROFINET interface for programming, HMI and PLC to PLC communication, Includes integrated Web server
Programmable Logic Controller : Type CPU 1511-1 PN (Part No. 6ES7 511-1AK01-0AB0)	Firmware version V1.8, CPU with display, work memory 150 kB code and 1 MB data, 60 ns bit instruction time, 4-stage protection concept, integrated technology functions: motion, closed-loop control, counting and measuring, PROFINET IO controller, supports RT/IRT, MRP and TCP/IP, protocols, S7 communication, Includes integrated Web server, constant bus cycle time, routing.
Expansion Module for digital inputs of CPU 1511-1 PN : Type DI 16x24VDC HF, (Part No. 6ES7 521-1BH00-0AB0)	Sixteen digital inputs with 24 VDC operation, input delay 0.05-20ms, configurable diagnostics, supports hardware interrupts
Expansion Module for digital outputs of CPU 1511-1 PN : Type DQ 16x24VDC/0.5A ST , Part No. 6ES7 522-1BH00-0AB0	Sixteen digital outputs with 24VDC operation, configurable diagnostic, configurable substitute value for output
HMI operator panel: Type KTP700 Basic PN, Part No. 6AV2 123-2GB03-0AX0	Firmware Version 13.0.1.0, 7'' TFT display, 800 x 480 pixel, 64 k colors, Key and Touch operation, 8 function keys, one PROFINET interface, one USB interface
Local Area Network Switch SCALANCE XB-208, Part No. 6GK5208 - 0BA00 - 2TB2	Manageable layer 2 Industrial Ethernet switch with eight 10/100 Mbits/s RJ45 ports, one console port for diagnostics , supports also ETHERNET/IP.
Application/User Programming Interface for Siemens industrial equipment.	Totally Intergraded Automation Portal (TIA Portal)Version 13 with SP1 and Update 8, STEP 7 Professional Version 13 with SP1 and Update 8, WinCC Basic Version 13 with SP1 and update 8

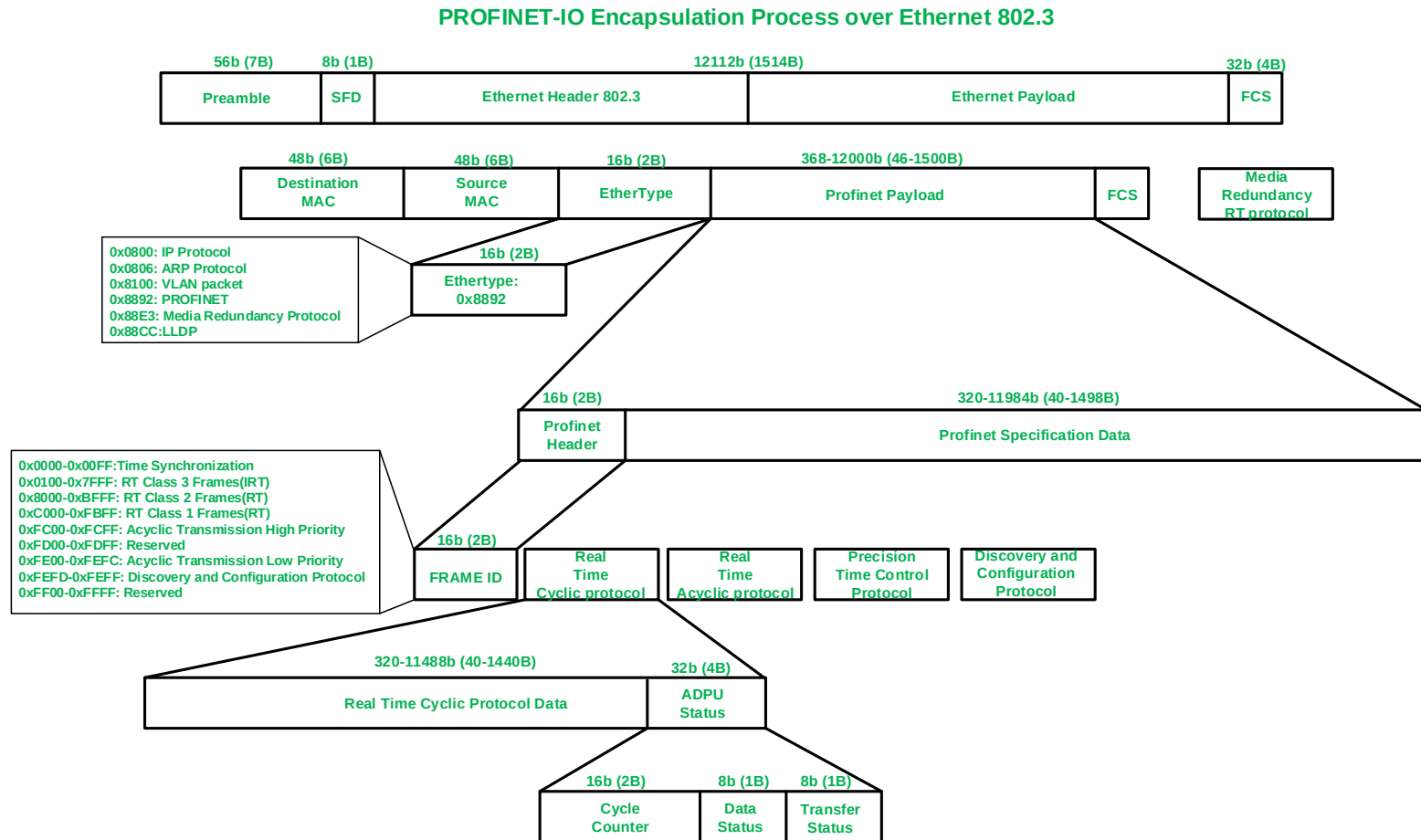


Figure 42. PROFINET IO Encapsulation Process for RT and IRT Traffic on Top of IEEE 802.3 Ethernet Frames

PROFINET-IO Encapsulation Process over Ethernet 802.1q

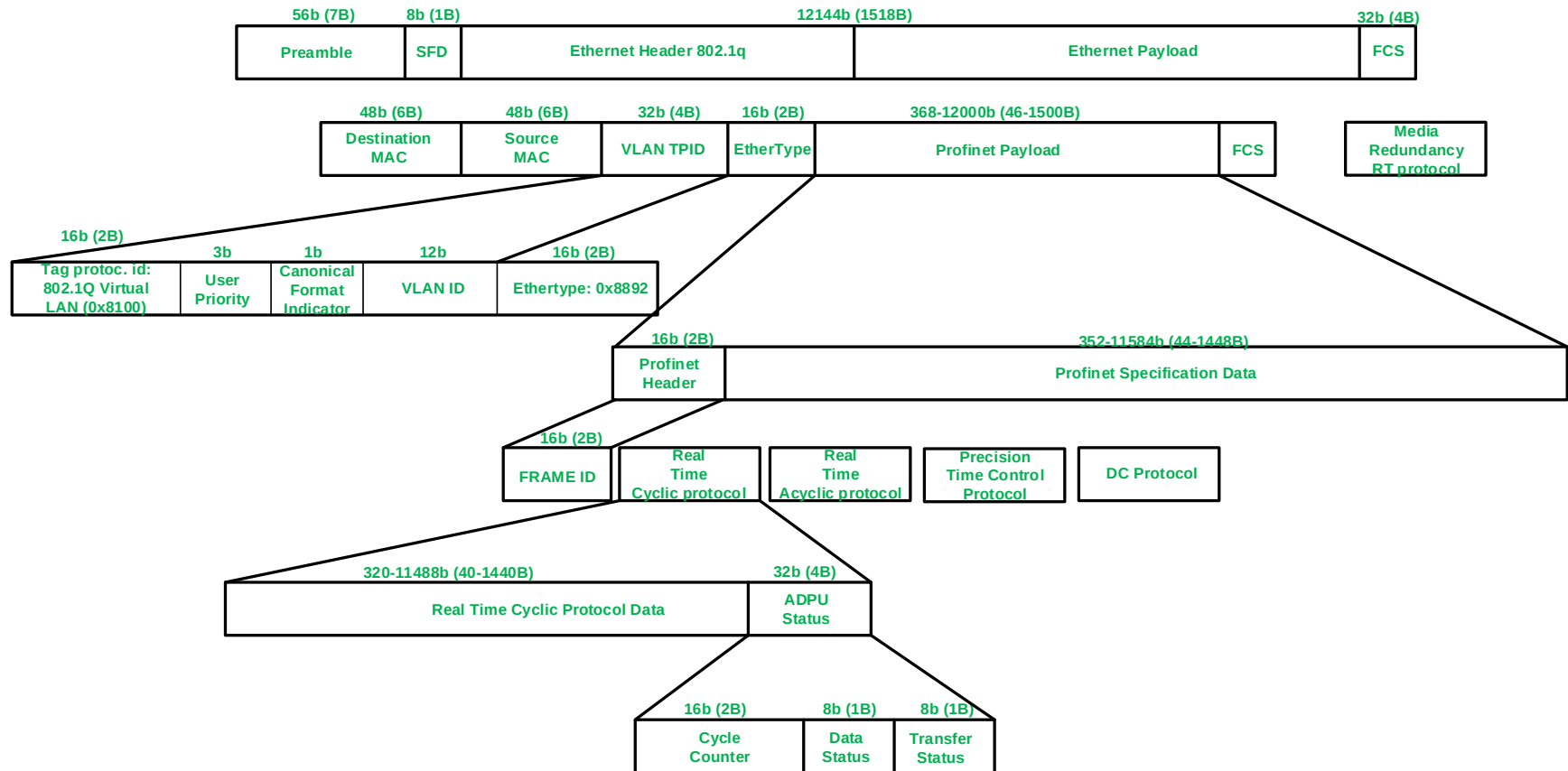


Figure 43. PROFINET IO Encapsulation Process for RT and IRT Traffic on Top of IEEE 802.1q VLAN Frame

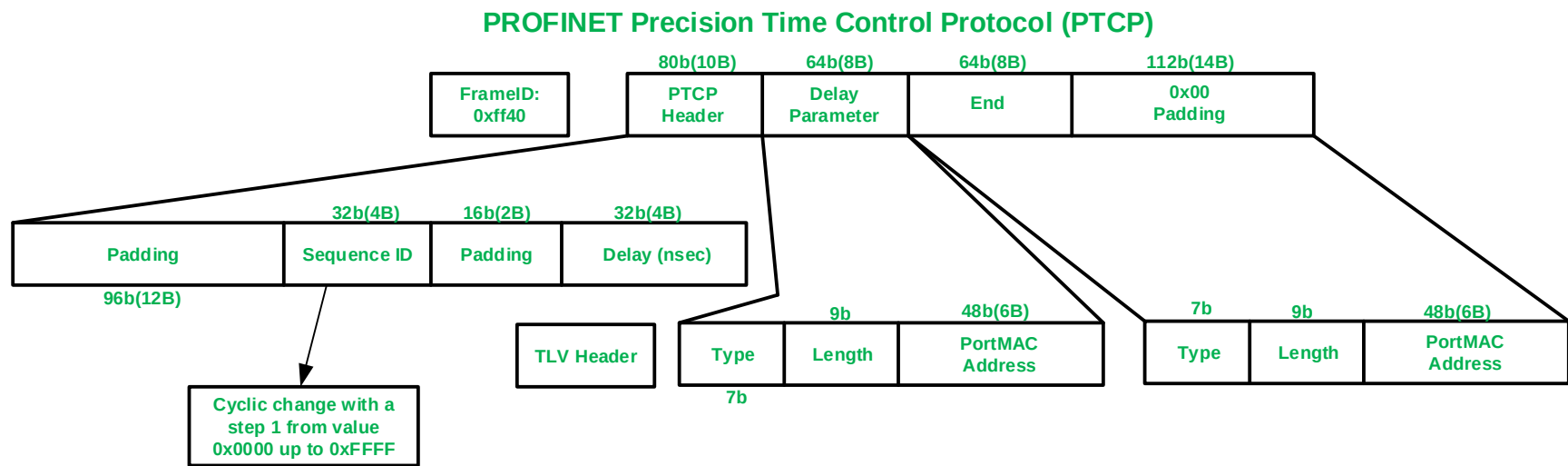


Figure 44. PROFINET IO PTPC Protocol as Part of the Payload Encapsulated in the PROFINET RT Payload

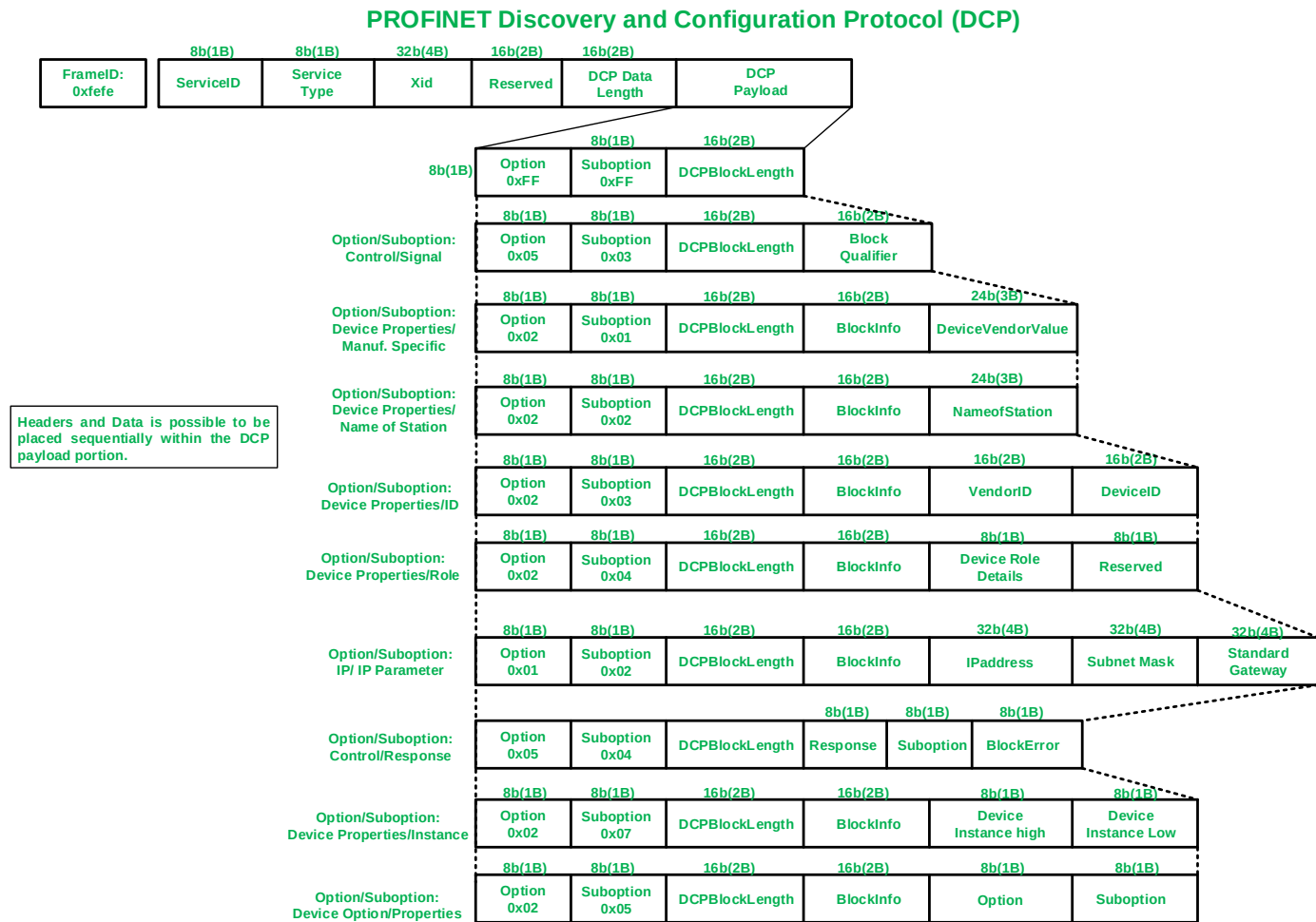


Figure 45. PROFINET IO DCP Protocol as Part of the Payload Encapsulated in the PROFINET RT Payload

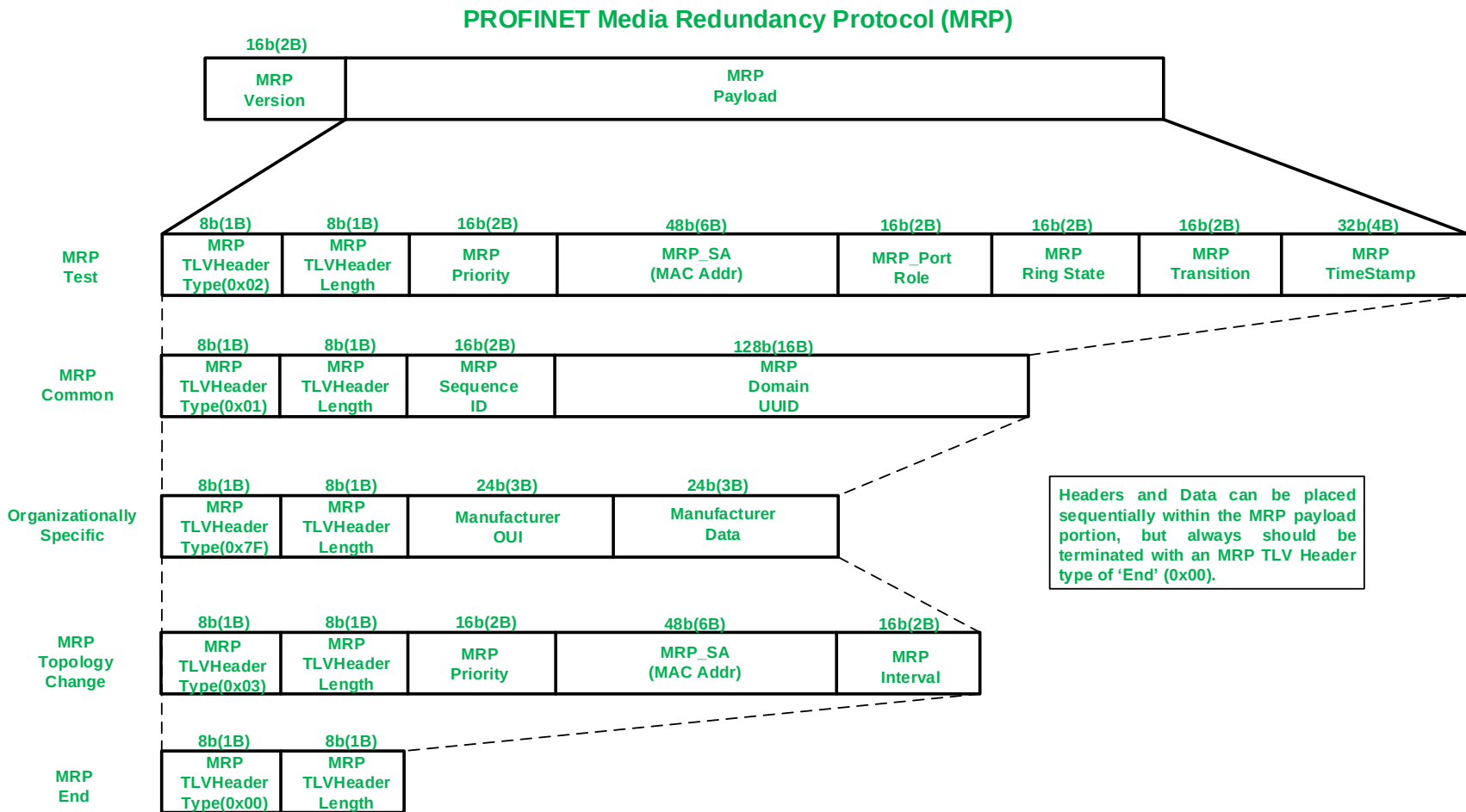


Figure 46. MRP Protocol as Part of the Payload Incorporated in the IEEE 802.3 Ethernet Frame

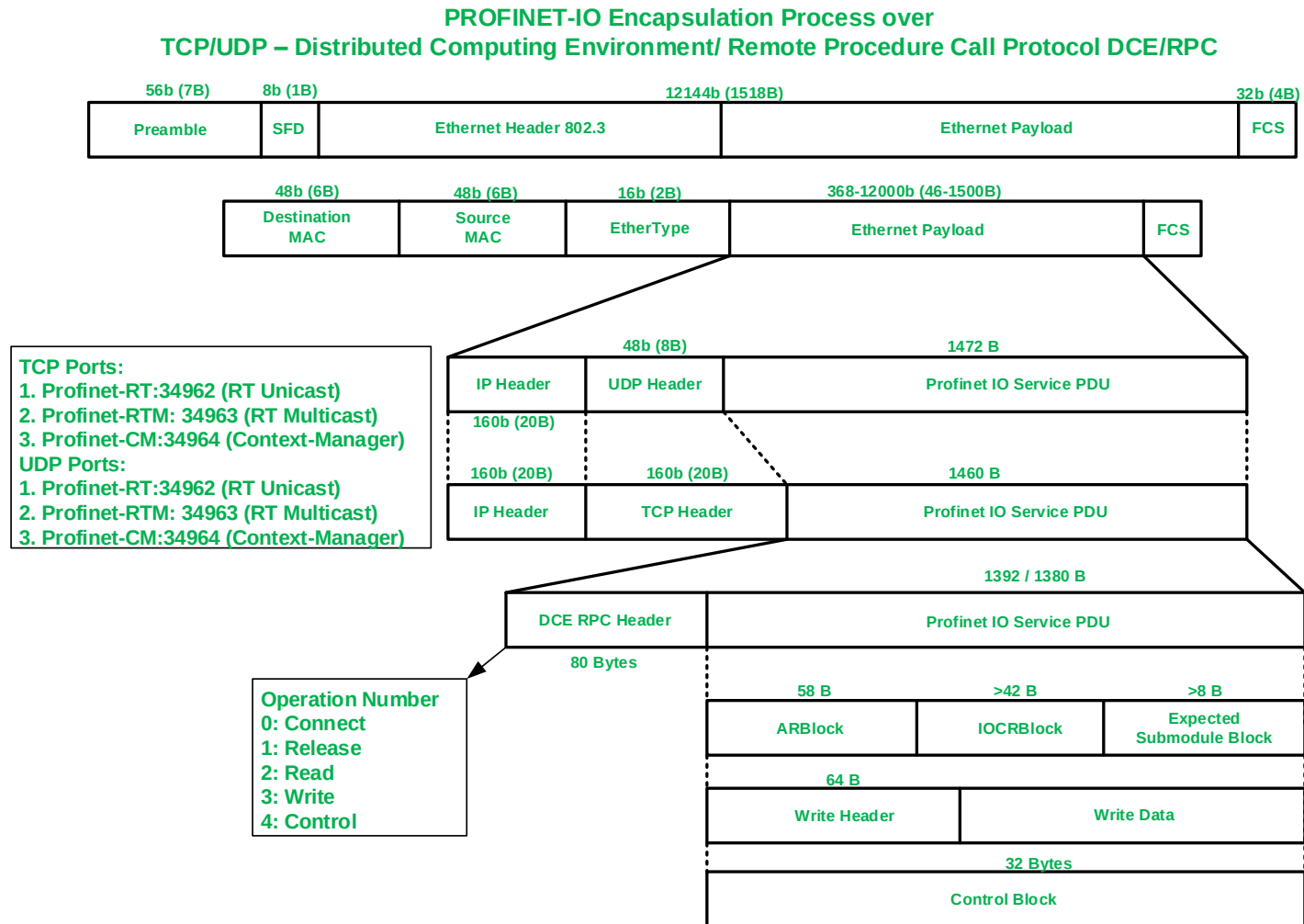


Figure 47. PROFINET IO Protocol on Top of IP/UDP/DCE-RPC Protocols of RT traffic

s7 comm over TPKT and ISO 8073/X.224 COTP

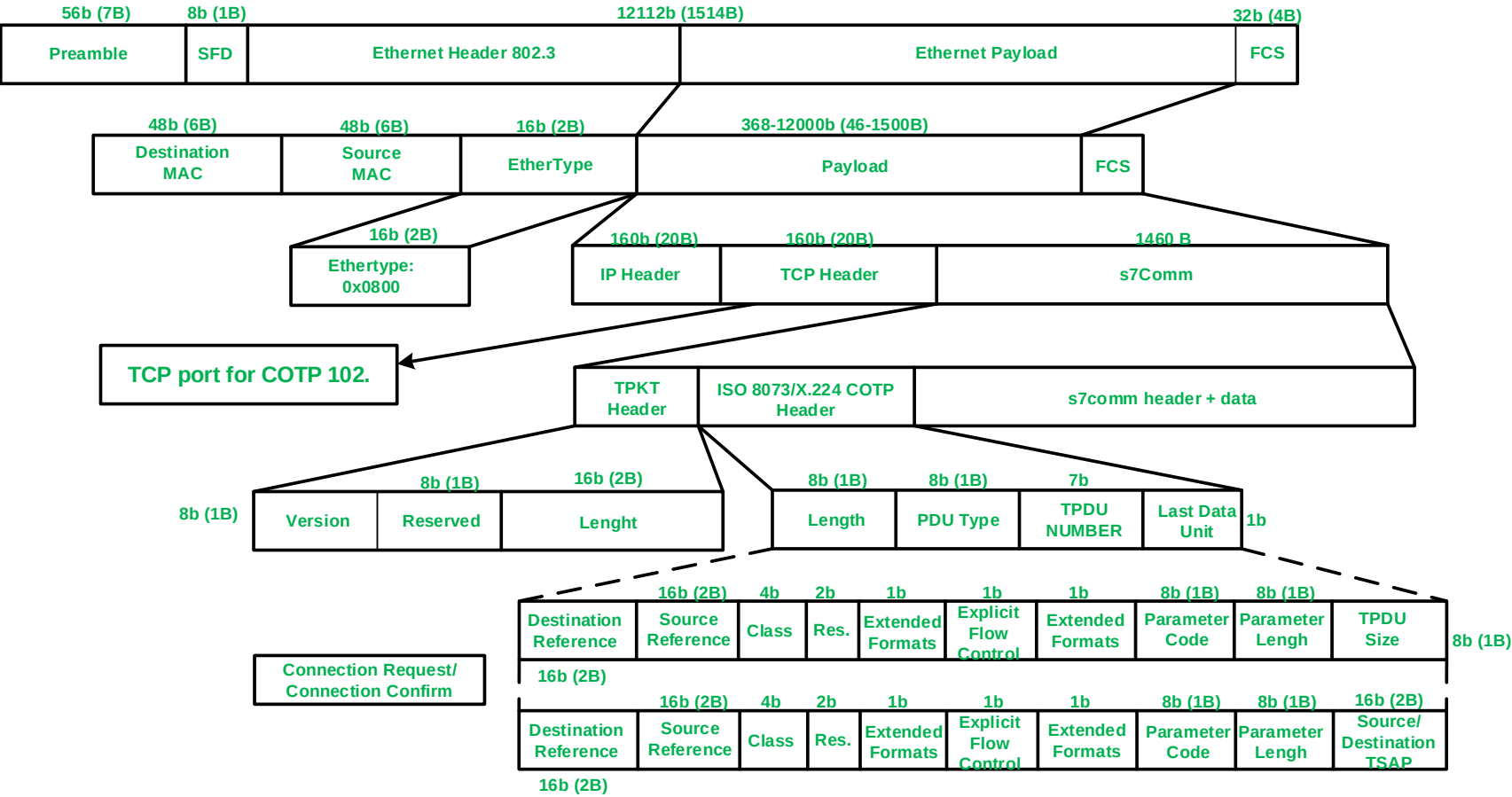


Figure 48. PROFINET Communication Protocol Encapsulation Process for Non-RT Traffic

s7comm over TPKT and ISO 8073/X.224 COTP headers and payload

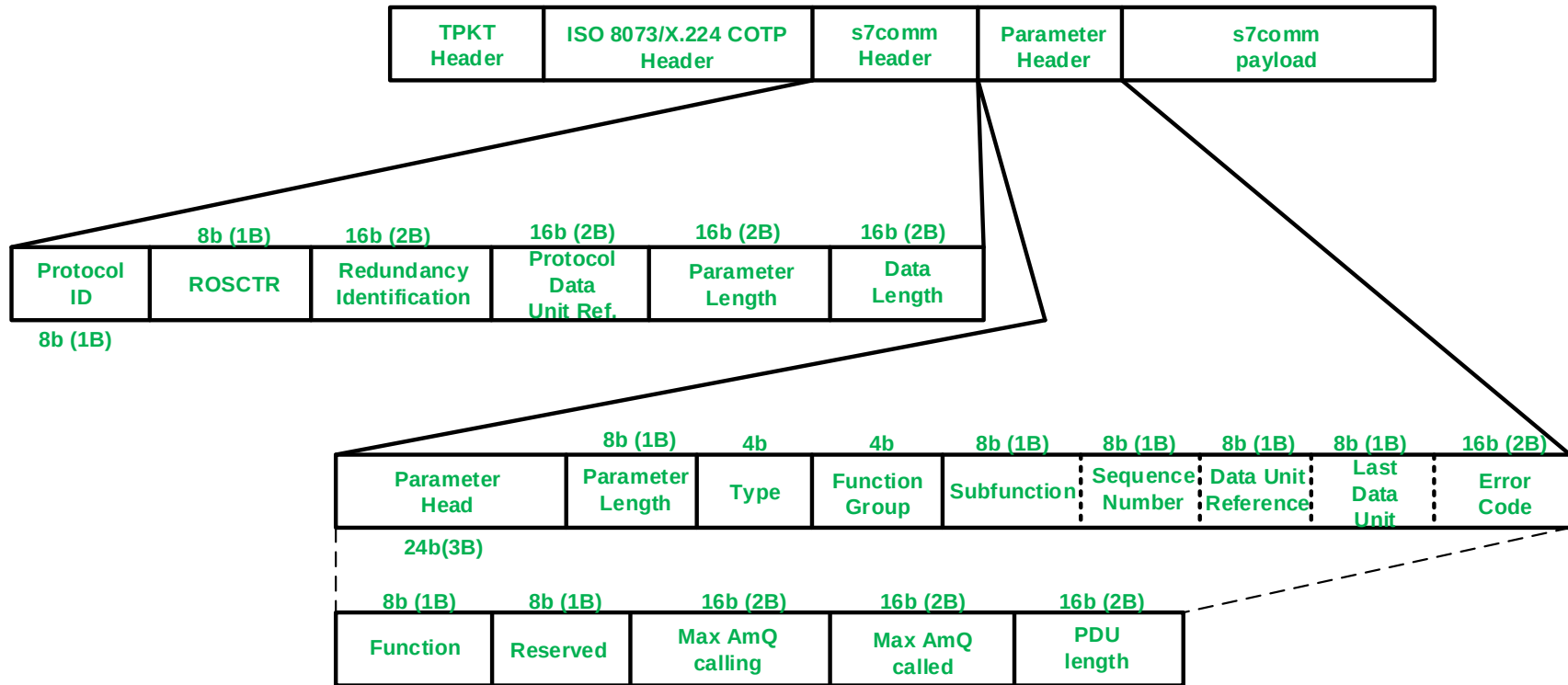


Figure 49. PROFINET Communication Protocol Encapsulation Process for Non-RT Traffic

EtherCAT Encapsulation Process over Ethernet 802.3

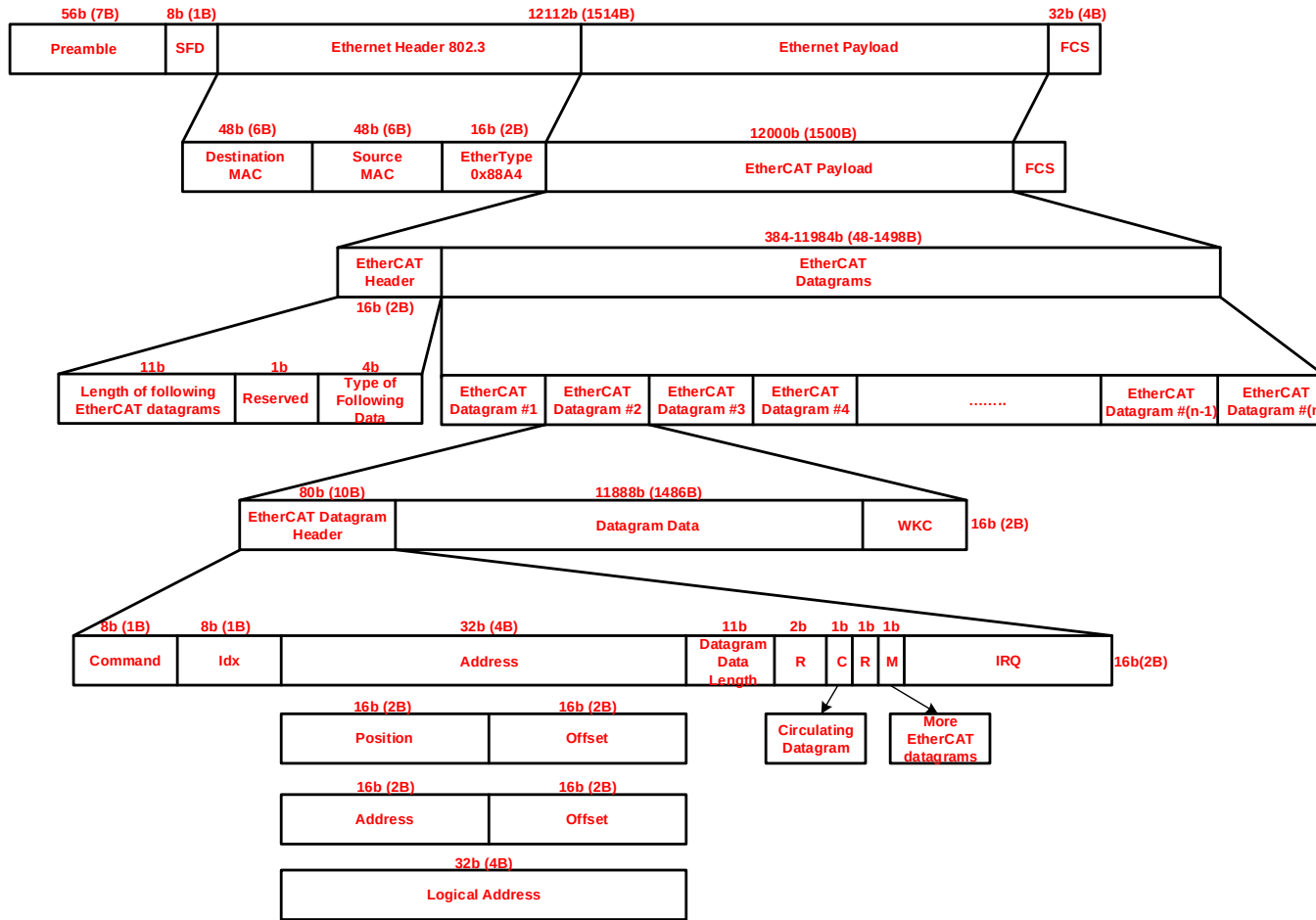


Figure 50. EtherCAT Communication Protocol Encapsulation Process for Non-RT Traffic.
Adapted from [28]–[31], [44]–[47].

EtherCAT over TCP-UDP/IP

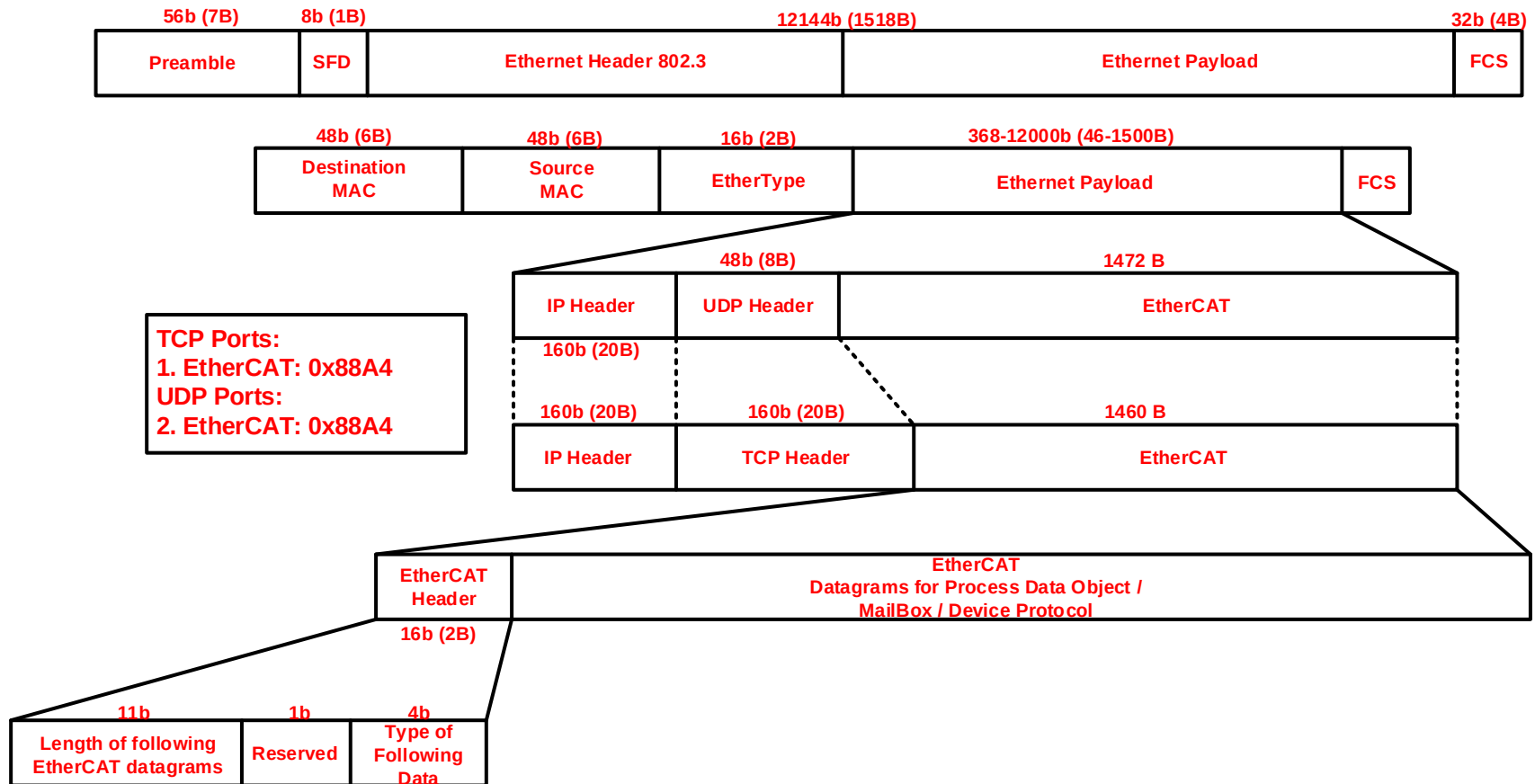


Figure 51. EtherCAT Communication Protocol Encapsulation Process Non-RT Traffic.
Adapted from [28]–[31], [44]–[47].

EtherCAT Encapsulation Process - Type of EtherCAT header No 1 – EtherCAT Device Protocol

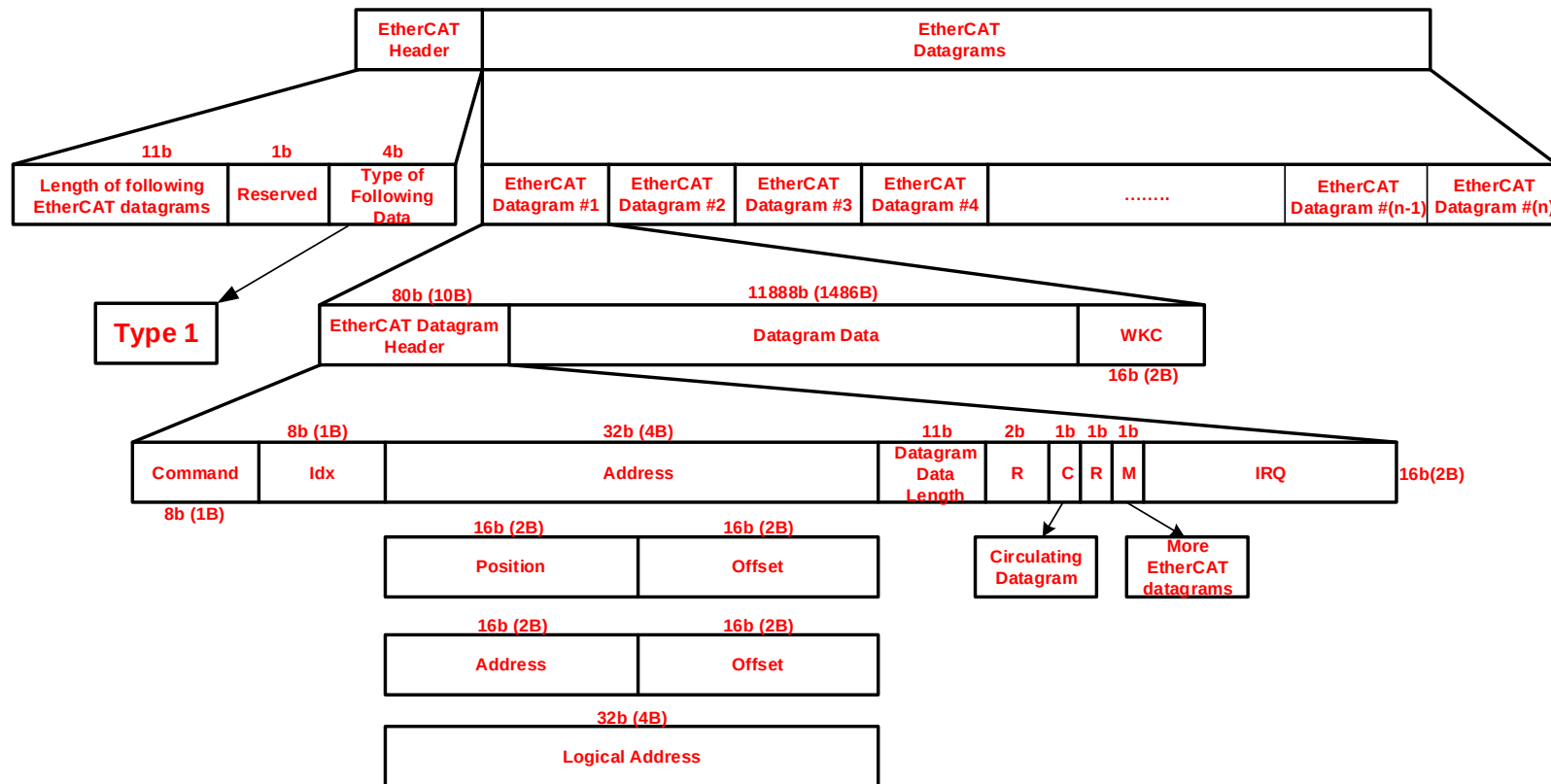


Figure 52. EtherCAT Communication Protocol Encapsulation Process for EtherCAT Device Protocol.
Adapted from [28]–[31], [44]–[47].

EtherCAT Encapsulation Process - Type of EtherCAT header No 4 - EtherCAT Mailbox

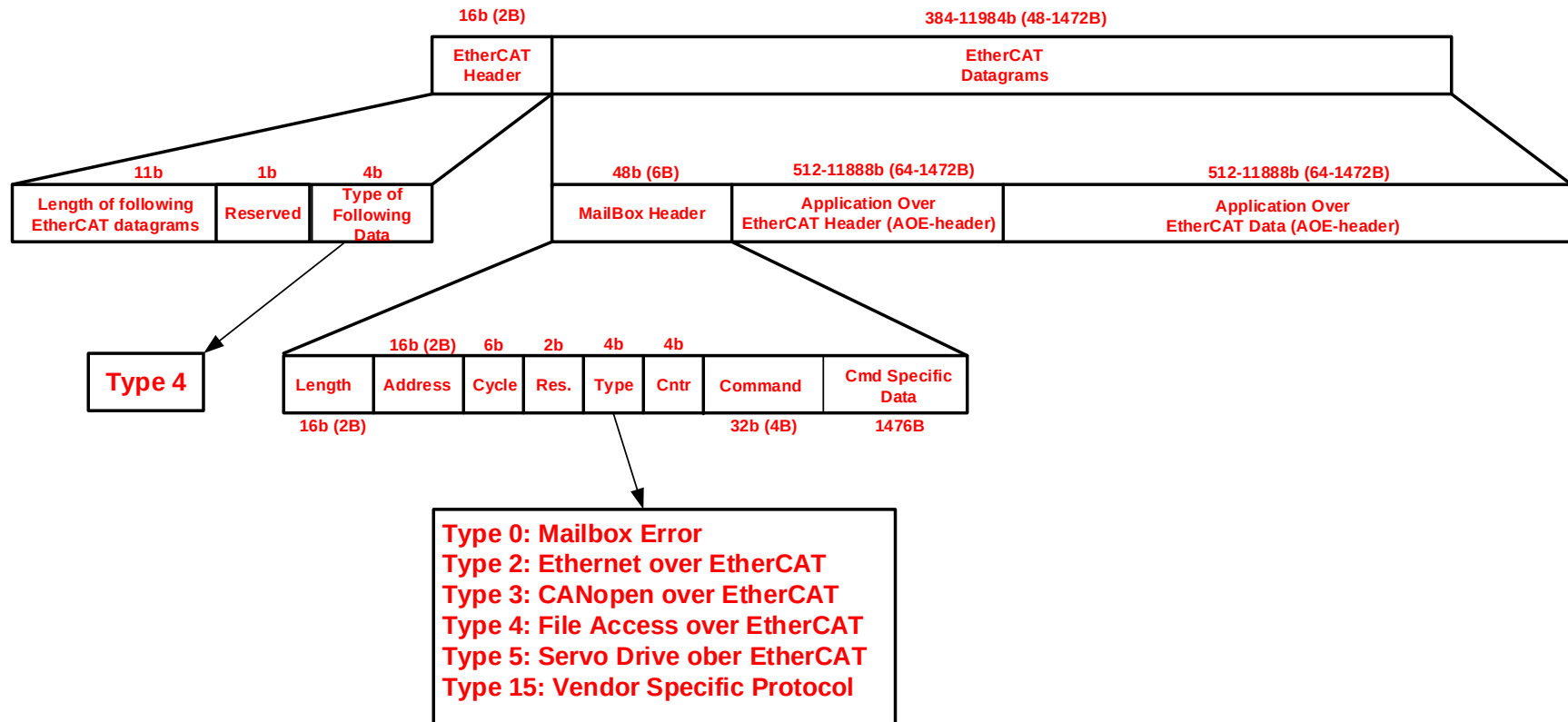


Figure 53. EtherCAT Communication Protocol Encapsulation Process for EtherCAT Mailbox.
Adapted from [28]–[31], [44]–[47].

EtherCAT Encapsulation Process – EtherCAT Mailbox Type No2 – Ethernet over EtherCAT

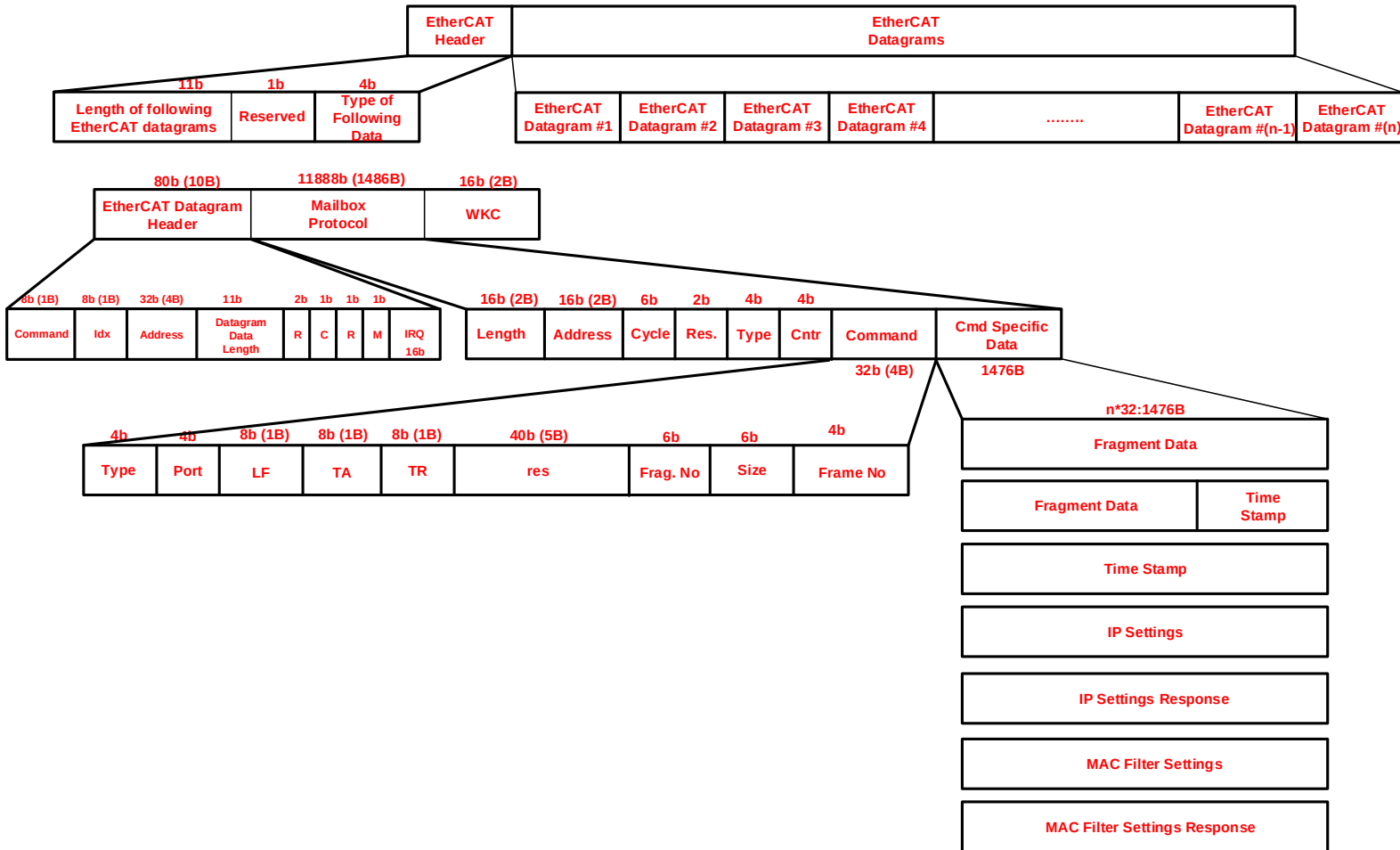


Figure 54. EtherCAT Communication Protocol Encapsulation Process for EoE. Adapted from [28]–[31], [44]–[47].

EtherCAT Encapsulation Process - EtherCAT Mailbox Type 3 – CANopen over EtherCAT

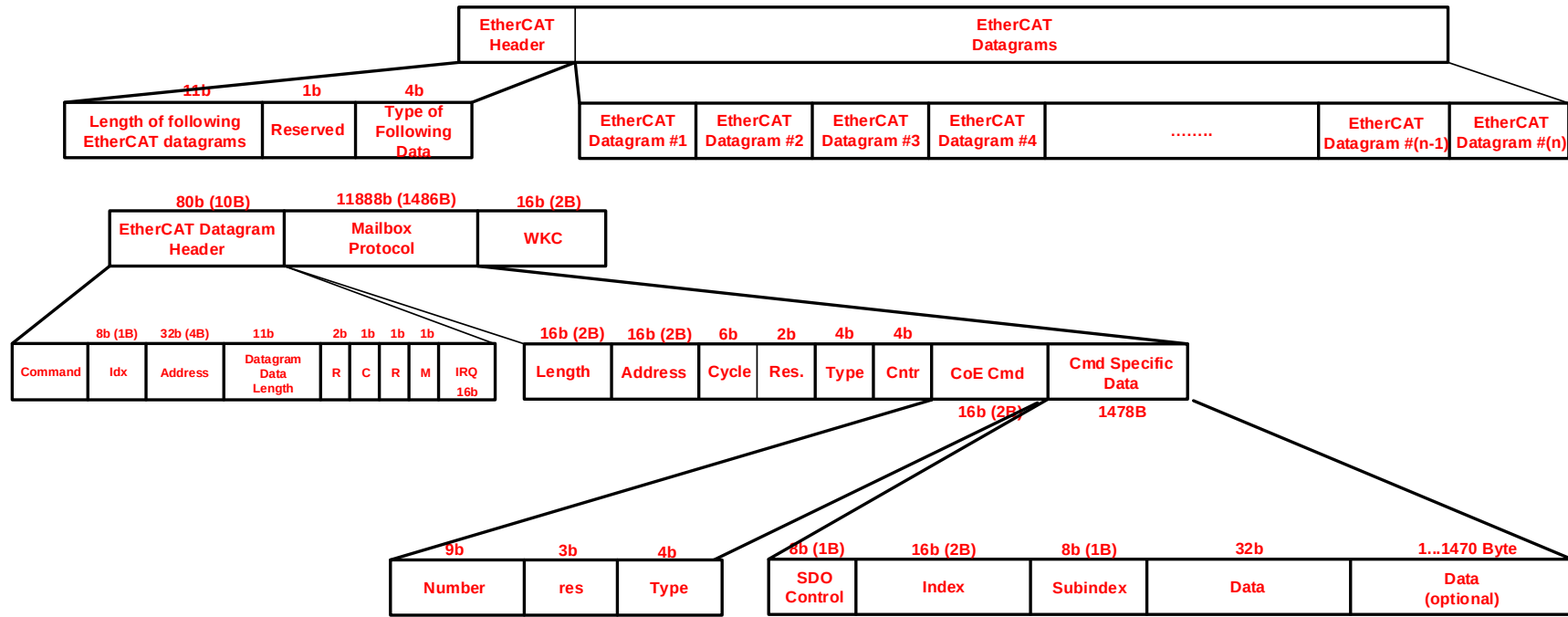


Figure 55. EtherCAT Communication Protocol Encapsulation Process for CoE. Adapted from [28]–[31], [44]–[47].

EtherCAT Encapsulation Process Type 5 – EtherCAT Mailbox – Servo Drive Over EtherCAT (SOE)

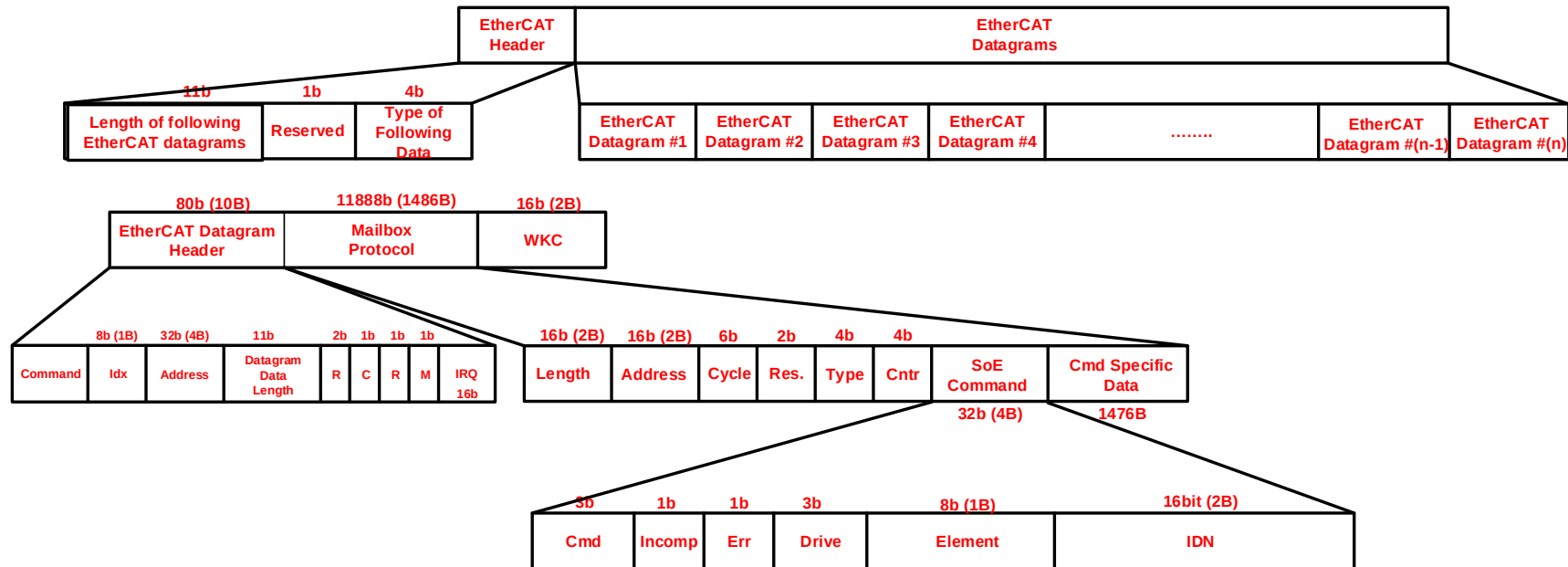


Figure 56. EtherCAT Communication Protocol Encapsulation Process for SoE. Adapted from [28]–[31], [44]–[47].

EtherCAT Encapsulation Process - Type of EtherCAT header No 5 – EtherCAT Process Data Object (PDO)

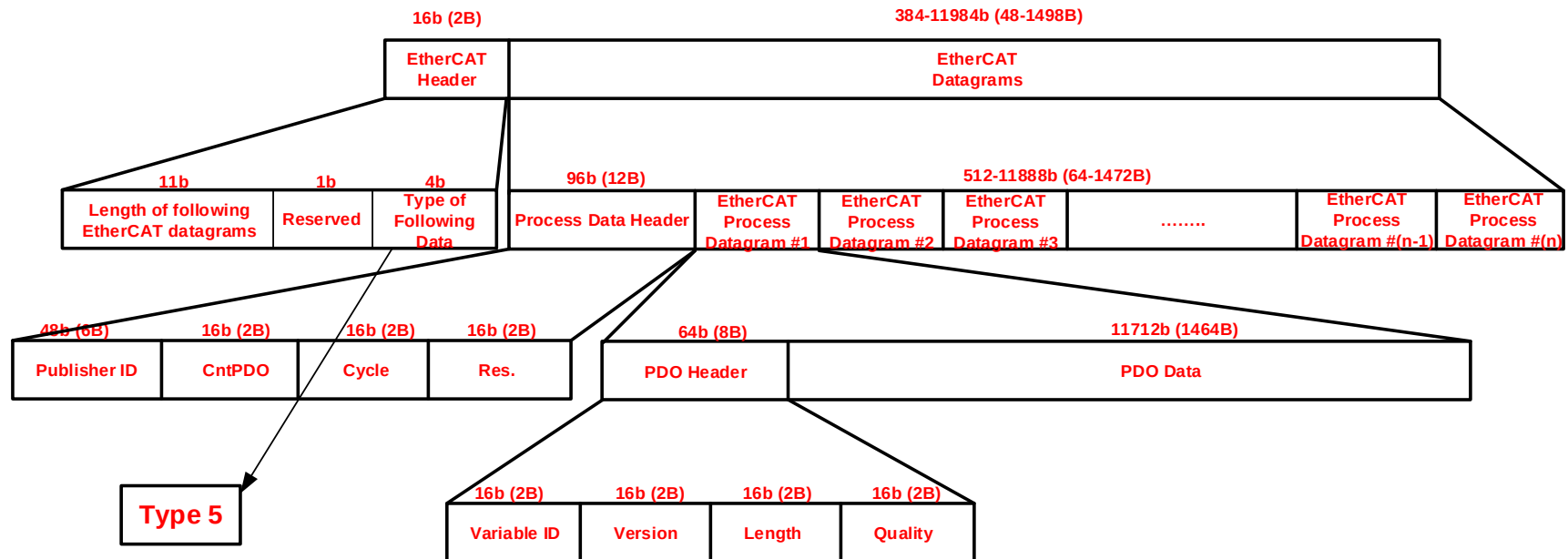


Figure 57. EtherCAT Communication Protocol Encapsulation Process for PDO. Adapted from [28]–[31], [44]–[47].

Ethernet/IP Encapsulation Process

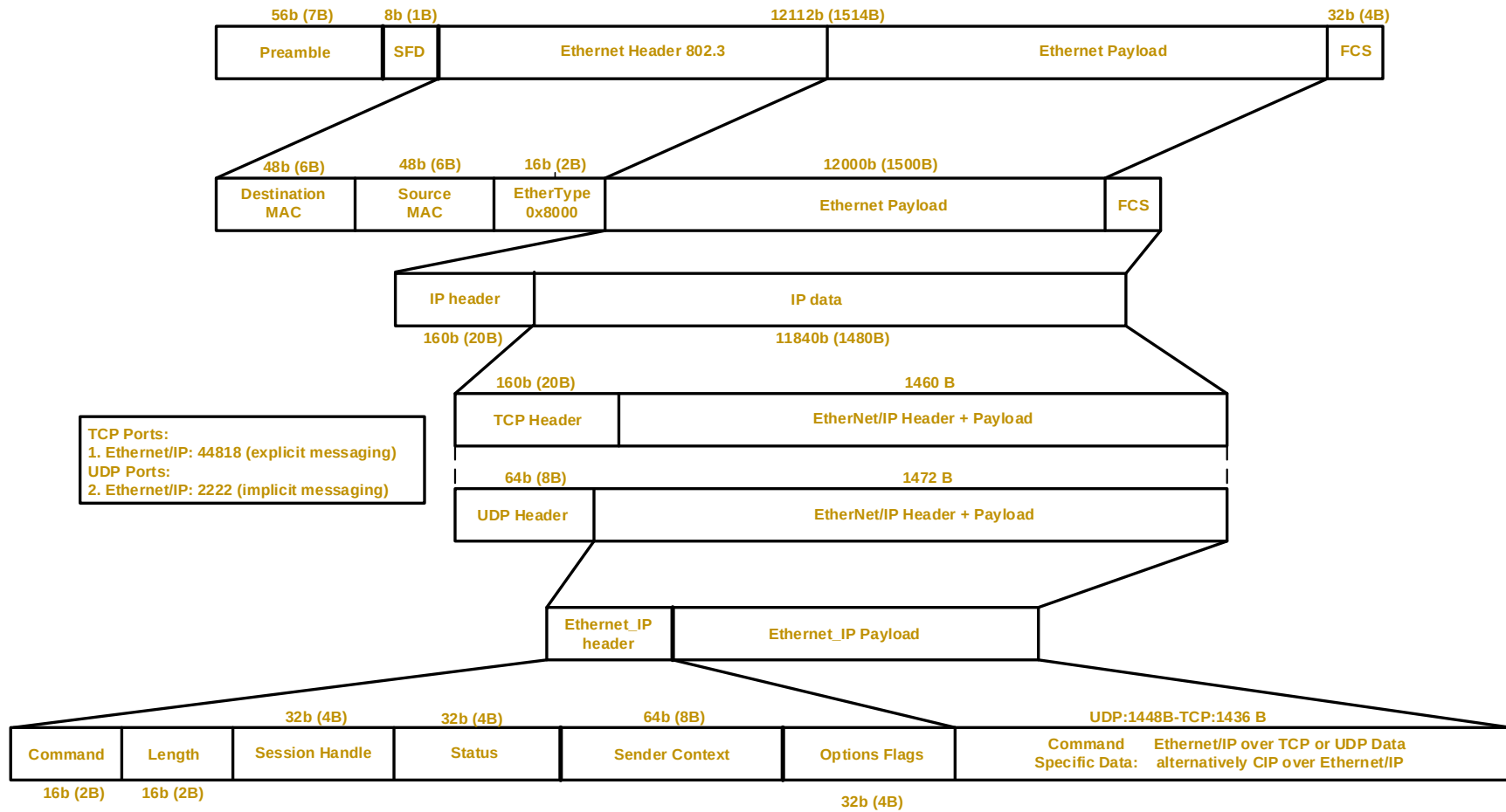


Figure 58. EtherNet/IP Encapsulation Process

Modbus Encapsulation Process

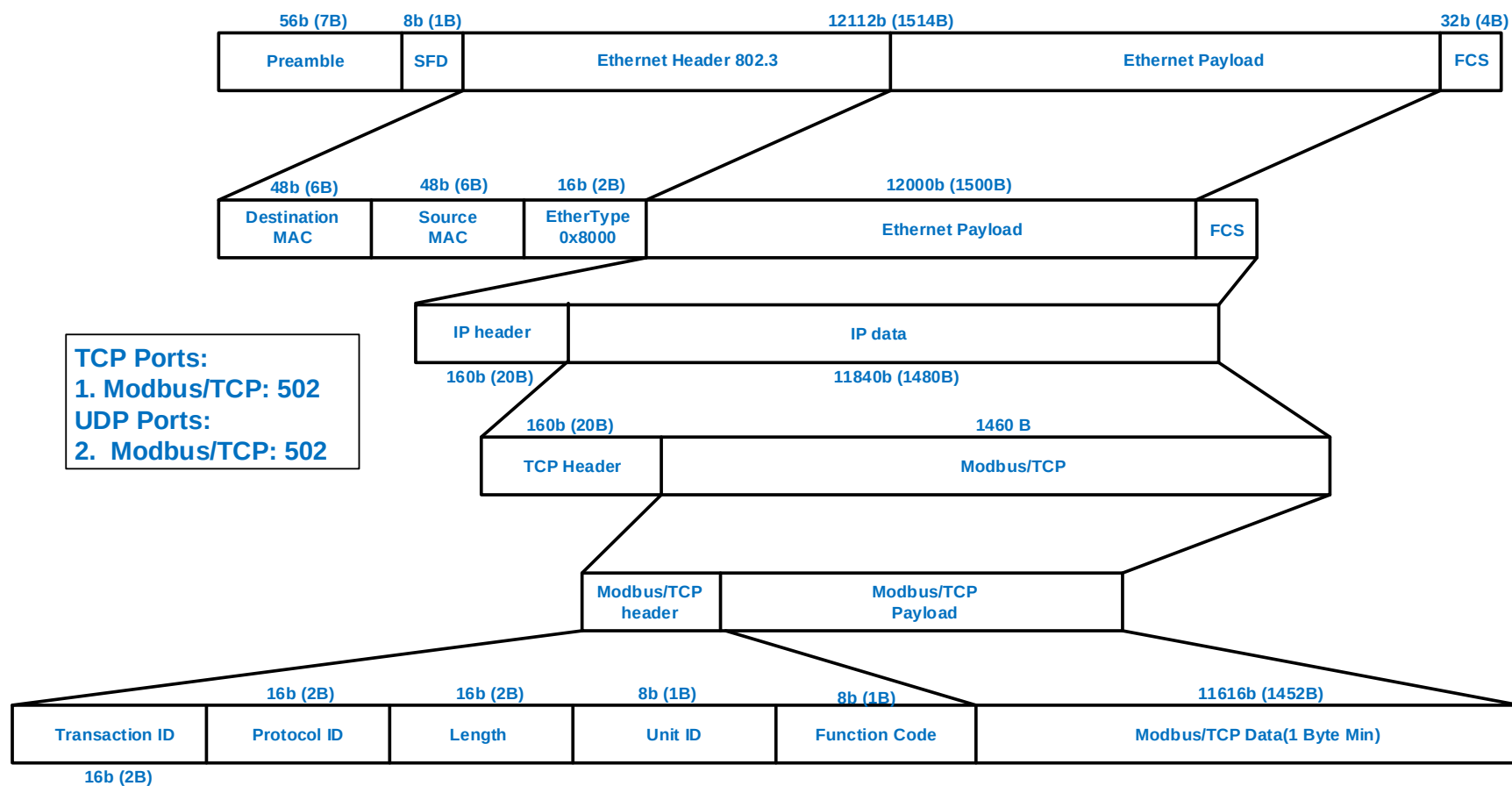


Figure 59. Modbus/TCP Communication Protocol Encapsulation Process for Non-RT Traffic

Time	Source	Destination	Protocol	Length	Info
1470246650.875289	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:25984 (Valid,Primary,Ok,Run)
1470246650.877305	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26048 (Valid,Primary,Ok,Run)
1470246650.879298	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26112 (Valid,Primary,Ok,Run)
1470246650.879307	192.168.100.3	192.168.100.4	PNIO-CM	232	Connect response, OK, ARBlockRes, IOCRBlockRes, IOCRBlockRes, AlarmCRBlockRes, ARServerBlock
1470246650.881313	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26176 (Valid,Primary,Ok,Run)
1470246650.883297	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26240 (Valid,Primary,Ok,Run)
1470246650.885306	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26304 (Valid,Primary,Ok,Run)
1470246650.887302	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26368 (Valid,Primary,Ok,Run)
1470246650.889268	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26432 (Valid,Primary,Ok,Run)
1470246650.891297	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26496 (Valid,Primary,Ok,Run)
1470246650.893331	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26560 (Valid,Primary,Ok,Run)
1470246650.895328	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26624 (Valid,Primary,Ok,Run)
1470246650.897301	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26688 (Valid,Primary,Ok,Run)
1470246650.899308	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26752 (Valid,Primary,Ok,Run)
1470246650.901305	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26816 (Valid,Primary,Ok,Run)
1470246650.903314	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26880 (Valid,Primary,Ok,Run)
1470246650.905316	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:26944 (Valid,Primary,Ok,Run)
1470246650.907313	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27008 (Valid,Primary,Ok,Run)
1470246650.909308	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27072 (Valid,Primary,Ok,Run)
1470246650.911305	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27136 (Valid,Primary,Ok,Run)
1470246650.913302	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27200 (Valid,Primary,Ok,Run)
1470246650.915293	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27264 (Valid,Primary,Ok,Run)
1470246650.917321	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27328 (Valid,Primary,Ok,Run)
1470246650.919283	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27392 (Valid,Primary,Ok,Run)
1470246650.921275	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27456 (Valid,Primary,Ok,Run)
1470246650.923327	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27520 (Valid,Primary,Ok,Run)
1470246650.925300	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27584 (Valid,Primary,Ok,Run)
1470246650.927276	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27648 (Valid,Primary,Ok,Run)
1470246650.929079	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0xf9ea042
1470246650.929305	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27712 (Valid,Primary,Ok,Run)
1470246650.931293	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27776 (Valid,Primary,Ok,Run)
1470246650.933287	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27840 (Valid,Primary,Ok,Run)
1470246650.935318	Siemens-_92:43:06	Siemens-_91:3c:0e	PNIO	64	RTC1, ID:0x8000, Len: 40, Cycle:27904 (Valid,Primary,Ok,Run)
1470246650.936885	192.168.100.3	192.168.100.4	PNIO-CM	174	Control response, OK, IODControlRes Prm End.rsp, Command: Done
1470246650.936893	192.168.100.3	192.168.100.4	PNIO-CM	174	Control request, IOXBlockReq Application Ready.req, Command: ApplicationReady

Figure 60. PROFINET IO Sequence of PDUs for Establishment of Communication between PLC 1 and PLC 2

Time	Source	Destination	Protocol	Length	Info
1470246650.090328	192.168.100.2	192.168.100.3	TCP	60	51351 → 102 [ACK] Seq=511 Ack=332 Win=2920 Len=0
1470246650.277781	fe80::3617:eb...	ff02::fb	MDNS	218	Standard query response 0x0000 PTR _udisks-ssh._tcp.local PTR kali [34:17:eb:b4:78:4e]._workstation
1470246650.860811	192.168.100.4	192.168.100.3	PNIO-CM	392	Connect request, ARBlockReq, IOCRBlockReq, IOCRBlockReq, ExpectedSubmoduleBlockReq, AlarmCRBlockReq
1470246650.880234	192.168.100.4	192.168.100.3	PNIO-CM	174	Control request, IODControlReq Prm End.req, Command: ParameterEnd
1470246650.881144	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:59904 (Valid,Primary,Ok,Run)
1470246650.883126	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:59968 (Valid,Primary,Ok,Run)
1470246650.885142	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60032 (Valid,Primary,Ok,Run)
1470246650.887144	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60096 (Valid,Primary,Ok,Run)
1470246650.889112	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60160 (Valid,Primary,Ok,Run)
1470246650.891113	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60224 (Valid,Primary,Ok,Run)
1470246650.893116	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60288 (Valid,Primary,Ok,Run)
1470246650.895112	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60352 (Valid,Primary,Ok,Run)
1470246650.897137	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60416 (Valid,Primary,Ok,Run)
1470246650.899154	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60480 (Valid,Primary,Ok,Run)
1470246650.901143	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60544 (Valid,Primary,Ok,Run)
1470246650.903147	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60608 (Valid,Primary,Ok,Run)
1470246650.905119	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60672 (Valid,Primary,Ok,Run)
1470246650.907146	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60736 (Valid,Primary,Ok,Run)
1470246650.909150	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60800 (Valid,Primary,Ok,Run)
1470246650.911143	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60864 (Valid,Primary,Ok,Run)
1470246650.913140	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60928 (Valid,Primary,Ok,Run)
1470246650.915156	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:60992 (Valid,Primary,Ok,Run)
1470246650.917148	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61056 (Valid,Primary,Ok,Run)
1470246650.919112	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61120 (Valid,Primary,Ok,Run)
1470246650.921140	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61184 (Valid,Primary,Ok,Run)
1470246650.923154	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61248 (Valid,Primary,Ok,Run)
1470246650.925124	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61312 (Valid,Primary,Ok,Run)
1470246650.927092	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61376 (Valid,Primary,Ok,Run)
1470246650.929094	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61440 (Valid,Primary,Ok,Run)
1470246650.931139	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61504 (Valid,Primary,Ok,Run)
1470246650.933151	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61568 (Valid,Primary,Ok,Run)
1470246650.935146	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61632 (Valid,Primary,Ok,Run)
1470246650.937139	Siemens-_91:3...	Siemens-_92:43...	PNIO	64	RTC1, ID:0x8010, Len: 40, Cycle:61696 (Valid,Primary,Ok,Run)
1470246650.937616	192.168.100.4	192.168.100.3	PNIO-CM	174	Control response, OK, IOXBlockRes Application Ready.rsp, Command: Done

Figure 61. PROFINET-IO Sequence of PDUs for Establishment of Communication between PLC 1 and PLC 2

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] TED. (2011, March). Ralph Langner: Cracking Stuxnet, a 21st-century cyber weapon. [www.ted.com videos]. Available: http://www.ted.com/talks/ralph_langner_cracking_stuxnet_a_21st_century_cyber_weapon. Accessed Feb. 3, 2016.
- [2] Stuxnet. (2015, n.d.). *Wikipedia*. [Online] Available: <https://en.wikipedia.org/wiki/Stuxnet>. Accessed Feb. 3, 2016.
- [3] N. Falliere, L. O Murchu, and E. Chien, “W32.Stuxnet dossier,” Symantec Security Response, Mountain View, CA, Tech. Report. Version 1.4, Feb. 2011.
- [4] Multiple vulnerabilities in Cisco IOS software common industrial protocol. (2015, Mar. 31). Cisco Security Advisory. [Online]. Available: <https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20150325-cip>
- [5] Advisory (ICSA-14-079-02), Siemens SIMATIC S7-1200 vulnerabilities. (2014, Mar. 20). ICS-CERT. [Online]. Available: <https://ics-cert.us-cert.gov/advisories/ICSA-14-079-02>
- [6] Advisory (ICSA-16-040-02), Siemens SIMATIC S7-1500 CPU vulnerabilities. (2016, Feb. 09). ICS-CERT. [Online]. Available: <https://ics-cert.us-cert.gov/advisories/ICSA-16-040-02>
- [7] Advisory (ICSA-12-342-01B) Rockwell Allen-Bradley MicroLogix, SLC 500, and PLC-5 Fault Generation vulnerability (Update B). ICS-CERT. [Online]. Available: <https://ics-cert.us-cert.gov/advisories/ICSA-12-342-01B>
- [8] P. Amini (2007). Fuzzing frameworks [Online]. Available: https://www.blackhat.com/presentations/bh-usa-07/Amini_and_Portnoy/Whitepaper/bh-usa-07-amini_and_portnoy-WP.pdf
- [9] K. Stouffer, J. Falco, and K. Scarfone, Special Publication 800–82, “Guide to Industrial Control Systems (ICS) security,” U.S. Department of Commerce, National Institute of Standards and Technology, Washington, DC, 2003, pp. 2–3.
- [10] E. D. Knapp, *Industrial Network Security, Securing Critical Infrastructure Networks for Smart Grid, SCADA and Other Industrial Control Systems*, 1st ed. Waltham, MA: Syngress, 2011, pp. 89–97.
- [11] R. Zurawski, *Industrial Communication Technology Handbook*, 2nd ed. Boca Raton, FL: CRC Press, 2015, pp. 1.1–1.50, 18.1–18.26.

- [12] *Digital Data Communications for Measurement and Control—Fieldbus for Use in Industrial Control Systems, Part 1: Introduction*, IEC 61158-1, 2003.
- [13] P. Danielis, J. Skodzik, V. Altmann, E. Schweissguth, F. Golasowski, D. Timmermann, and J. Schacht, “Survey on real-time communication via Ethernet in industrial automation environments,” in *IEEE 19th International Conference on Emerging Technology and Factory Automation*, Barcelona, Spain, 2014, pp. 1–8.
- [14] Industrial Ethernet. (2016, Feb. 13). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Industrial_Ethernet. Accessed Oct. 28, 2016.
- [15] OSI model. (2016, Jun. 27). *Wikipedia*. [Online], Available: https://en.wikipedia.org/wiki/OSI_model. Accessed Jun. 28, 2016.
- [16] Internet protocol suite. (2016, Jun. 20). *Wikipedia*. [Online], Available: https://en.wikipedia.org/wiki/Internet_protocol_suite. Accessed Jun. 28, 2016.
- [17] The data encapsulation process (n.d.). CCI Learning Solutions Inc. [Online]. Available: <http://strata.ccilearning.com/Lesson3NetworkingandInternetworking/TheDataEncapsulationProcess/tabid/733/language/en-CA/Default.aspx>
- [18] L-force communication Ethernet communication manual, Ethernet in industrial applications (n.d.). Lenze. [Online]. Available: http://download.lenze.com/TD/Ethernet__Ethernet%20in%20industrial%20applications__v3-1__EN.pdf. Accessed Dec. 10, 2015.
- [19] M. Rostan. Industrial Ethernet technologies: Overview and comparison (2014, Apr. 04). EtherCAT Technology Group. [Online]. Available: https://www.ethercat.org/en/downloads/downloads_5AED3409F60546F9B5F0049D361969C4.htm
- [20] P. Prakash, T. Mauer (2015, Sep. 3). Optimizing the industrial design cycle with programmable multiprotocol industrial communication. A new era in industrial networking [Online]. Available: <http://www.ti.com/tool/tmdsice3359>
- [21] PROFIBUS Nutzerorganisation e.V. PNO. (2011, June). PROFINET system description – Technology and application [Online]. Available: https://scadahacker.com/library/Documents/ICS_Protocols/PI%20-%20PROFINET%20System%20Description%20-%20Technology%20and%20Application.pdf
- [22] Profinet. (2016, June 14). *Wikipedia*. [Online]. Available: <https://de.wikipedia.org/wiki/Profinet>. Accessed Jul. 13, 2016.

- [23] Automate with the leading Industrial Ethernet standard and profit now. PROFINET, Answers for industry. (2010, Aug.). Siemens AG Industry Sector. [Online]. Available: <https://w3.siemens.com/mcms/water-industry/en/Documents/PROFINET.pdf>
- [24] A. Verwer, (2010, Oct. 3). Introduction to PROFINET [Online]. Available: [http://www.profibus.com/uploads/media/pxddamkey\[9234\]_FA_2010_Oct_3_Introduction_to_PROFINET_PeteBrown.pdf](http://www.profibus.com/uploads/media/pxddamkey[9234]_FA_2010_Oct_3_Introduction_to_PROFINET_PeteBrown.pdf)
- [25] R. Pigan, M. Metter, *Automating with PROFINET-Industrial Communication based on Industrial Ethernet*, 2nd ed. Berlin and Munich, Germany: Publicis Publishing, Erlangen, 2008, p. 46.
- [26] PROFINET protocol family. (n.d.). *Wireshark Wiki*. [Online]. Available: <https://wiki.wireshark.org/PROFINET>. Accessed Jul. 24, 2016.
- [27] PROFINET IO, Measurements of telegram-sequences with Ethereal. (2005, Jun. 27). PNCC Burgdorf. [Online]. Available: http://www.felser.ch/download/6_PN_Measurements_E02.pdf
- [28] EtherCAT communication, communication principles (2011, Mar.). EtherCAT Technology Group. [Online]. Available: ftp://ftp.mefi.cz/Skoleni%20EtherCAT%202013/EtherCAT_Communication.pdf
- [29] B. M. Wilanowski, J. D. Irwin, *The Industrial Electronics Handbook – Industrial Communication Systems*, 2nd ed. Boca Raton, FL: CRC Press, 2011, pp. 38.1–38.10.
- [30] EtherLab. (2012, Feb. 09). Ingenieurgesellschaft IGH. [Online]: Available: <http://etherlab.org/en/index.php>
- [31] *EtherCAT for factory networking* (2009, Nov. 1). EtherCAT Technology Group (ETG). [Online]. Available: https://www.ethercat.org/en/downloads/downloads_8A145EAB9B0A4B4CB3292E23EF04E9D7.htm
- [32] Technology overview series: Ethernet/IP, CIP on Ethernet technology. (2015, Oct. 01). ODVA. [Online]. Available: https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00138R6_Tech-Series-EtherNetIP.pdf
- [33] Technical reference – EtherNet/IP. (n.d.). Acromag Inc. [Online]. Available: https://www.acromag.com/sites/default/files/Acromag_Intro_EthernetIP_747A.pdf

- [34] Common Industrial Protocol (CIP™) and the family of CIP networks. (2015, Oct. 01). ODVA & ControlNet International, Ltd. [Online]. Available: https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00138R6_Tech-Series-EtherNetIP.pdf
- [35] The CIP networks library, EtherNet/IP adaption of CIP. (n.d.). ODVA & ControlNet International, Ltd. [Online]. Available: http://www.tud.ttu.ee/im/Kristjan.Sillmann/ISP0051%20Rakenduslik%20Andmeside/CIP%20docs/CIP%20Vol2_1.4.pdf. Accessed Aug. 1, 2016.
- [36] Technical Reference – Modbus TCP/IP. (n.d.). Acromag Incorporated. [Online]. Available: https://www.acromag.com/sites/default/files/Acromag_Intro_ModbusTCP_765A.pdf. Accessed: Jul. 24, 2016.
- [37] Modbus application protocol specification V1.1b3. (2012, Apr. 26). The Modbus Organization. [Online]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf
- [38] Modbus messaging on TCP/IP implementation guide V1.0b. (2006, Oct. 24). The Modbus Organization. [Online]. Available: http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf
- [39] M. Herrero Collantes and A. Lopez Padilla. (2015, May) Instituto Nacional de Ciberseguridad de Espagna (INCIBE). [Online]. Available: https://www.incibe.es/extfrontinteco/img/File/intecocert/ManualesGuias/incibe_protocol_net_security_ics.pdf
- [40] ICS whitepapers.(n.d.) SANS [Online]. Available: <https://ics.sans.org/media/ics-2015.pdf>. Accessed Jul. 1, 2015.
- [41] Linux programmer’s manual, online manual pages: Socket(2). (n.d.). man7.org. [Online]. <http://man7.org/linux/man-pages/man2/socket.2.html>. Accessed Aug. 05, 2016.
- [42] “Raw Sockets Overview,” class notes for Advanced Vulnerability Assessment, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, spring 2015.
- [43] Protocol API PROFINET IO RT/IRT device V3.5.x.x. (2014, Jul.). Hilscher Gesellschaft für Systemautomation mbH. [Online] Available: http://www.hilscher.com/fileadmin/cms_upload/de/Resources/pdf/PROFINET_IO-Device_V3.5_Protocol_API_08_EN.pdf

- [44] EtherCAT – the Ethernet fieldbus (2014, Nov. 24). EtherCAT Technology Group. [Online]. Available:
https://www.ethercat.org/en/downloads/downloads_07CC7DD0539B4C0EB526ADD5D4BC25CD.htm
- [45] PC-Control: EtherCAT - the Ethernet fieldbus (2011, Nov. 01). EtherCAT Technology Group. [Online]. Available:
https://www.ethercat.org/en/downloads/downloads_DE7F7AE8FCA2402CBCF9E32A701AD7AA.htm
- [46] EtherCAT Automation Protocol (EAP): EtherCAT for factory networking (2010, Jul. 19). EtherCAT Technology Group. [Online]. Available:
https://www.ethercat.org/en/downloads/downloads_42EBC3A04BB0428BAC1A7654609E098B.htm
- [47] M. Rostan. (2004, Jul. 12). High speed industrial Ethernet for semiconductor equipment [Online]. Available:
https://www.ethercat.org/download/documents/Ethernet_for_Semi_0607.pdf

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California