Behavioral Modeling for Anomaly Detection in Industrial Control Systems

Iñaki Garitano Garitano

Supervisors:

Roberto Uribeetxeberria Ezpeleta

and

Urko Zurutuza Ortega

A thesis submitted to Mondragon Unibertsitatea for the degree of Doctor of Philosophy

Department of Electronics and Computer Science

Mondragon Goi Eskola Politeknikoa

Mondragon Unibertsitatea

December 2013

gurasuei ta bidelagun izan zaten danoi

Abstract

In 1990s, industry demanded the interconnection of corporate and production networks. Thus, Industrial Control Systems (ICSs) evolved from 1970s proprietary and close hardware and software to nowadays Commercial Off-The-Shelf (COTS) devices. Although this transformation carries several advantages, such as simplicity and cost-efficiency, the use of COTS hardware and software implies multiple Information Technology vulnerabilities. Specially tailored worms like Stuxnet, Duqu, Night Dragon or Flame showed their potential to damage and get information about ICSs. Anomaly Detection Systems (ADSs), are considered suitable security mechanisms for ICSs due to the repetitiveness and static architecture of industrial processes. ADSs base their operation in behavioral models that require attack-free training data or an extensive description of the process for their creation.

This thesis work proposes a new approach to analyze binary industrial protocols payloads and automatically generate behavioral models synthesized in rules. In the same way, through this work we develop a method to generate realistic network traffic in laboratory conditions without the need for a real ICS installation. This contribution establishes the basis of future ADS as well as it could support experimentation through the recreation of realistic traffic in simulated environments. Furthermore, a new approach to correct delay and jitter issues is proposed. This proposal improves the quality of time-based ADSs by reducing the false positive rate.

We experimentally validate the proposed approaches with several statistical methods, ADSs quality measures and comparing the results with traffic taken from a real installation. We show that a payload-based ADS is possible without needing to understand the payload data, that the generation of realistic network traffic in laboratory conditions is feasible and that delay and jitter correction improves the quality of behavioral models.

As a conclusion, the presented approaches provide both, an ADS able to work with private industrial protocols, together with a method to create behavioral models for open ICS protocols which does not requite training data.

Resumen

En los años 90, la industria proclamó la interconexión de las redes corporativas y los de producción. Así, los Sistemas de Control Industrial (SCI) evolucionaron desde el hardware y software propietario de los 70 hasta los dispositivos comunes de hoy en día. Incluso si esta adopción implicó diversas ventajas, como el uso de hardware y software comunes, conlleva múltiples vulnerabilidades. Gusanos especialmente desarrollados como Stuxnet, Duqu, Night Dragon y Flame mostraron su potencial para causar daños y obtener información. Los Sistemas de Detección de Anomalías (SDA) están considerados como mecanismos de seguridad apropiados para los SCI debido a la repetitividad y la arquitectura estática de los procesos industriales. Los SDA basan su operación en modelos de comportamiento que requieren datos libres de ataque o extensas descripciones de proceso para su creación.

Esta tesis propone un nuevo enfoque para el análisis de los datos de la carga útil del tráfico de protocolos industriales binarios y la generación automática de modelos de comportamiento sintetizados en reglas. Así mismo, mediante este trabajo se ha desarrollado un método para generar tráfico de red realista en condiciones de laboratorio sin la necesidad de instalaciones SCI reales. Esta contribución establece las bases de un futuro SDA así como el respaldo a la experimentación mediante la recreación de tráfico realista en entornos simulados. Además, se ha propuesto un nuevo enfoque para la corrección de retraso y latencia. Esta propuesta mejora la calidad del SDA basados en tiempo reduciendo el ratio de falsos positivos.

Mediante la experimentación se han validado los enfoques propuestos utilizando algunos métodos estadísticos, medidas de calidad de SDA y comparando los resultados con tráfico obtenido a partir de instalaciones reales. Se ha demostrado que son posibles los SDA basados en carga útil sin la necesidad de entender el contenido de la carga, que la generación de tráfico realista en condiciones de laboratorio es posible y que la corrección del retraso y la latencia mejoran la calidad de los modelos de comportamiento.

Como conclusión, las propuestas presentadas proporcionan un SDA capaz de trabajar con protocolos privados de control industrial a la vez que un

método para la creación de modelos de comportamiento para SCI sin la necesidad de datos de entrenamiento.

Laburpena

90. hamarkadan industriak sare korporatibo eta industrialen arteko konexioa eskatu zuen. Horrela, Kontrol Sistema Industrialak (KSI) 70. hamarkadako hardware eta software jabedun eta itxitik gaur eguneko gailu estandarretara egin zuten salto. Eraldaketa honek hainbat onura ekarri baditu ere, era berean gailu estandarren erabilerak hainbat Informazio Teknologietako (IT) zaurkortasun ekarri ditu. Espezialki diseinatutako zizareek, Stuxnet, Duque, Night Dragon eta Flame esaterako, ondorio latzak gauzatu eta informazioa lapurtzean beraien potentzia erakutsi dute. Anomalia Detekzio Sistemak (ADS) KSI-etako segurtasun mekanismo egoki bezala kontsideraturik daude, azken hauen errepikakortasun eta arkitektura estatikoa dela eta. ADS-ak erasorik gabeko datu garbietan ikasitako edo prozesuen deskripzio sakona behar duten jarrera modeloetan oinarritzen dira.

Tesi honek protokolo industrial binarioak aztertu eta automatikoki jarrera modeloak sortu eta erregeletan sintetizatzen dituen ikuspegia proposatzen du. Era berean lan honen bidez laborategi kondizioetan sare trafiko errealista sortzeko metodo bat aurkezten da, KSI-rik behar ez duena. Ekarpen honek etorkizuneko ADS baten oinarriak finkatzen ditu, baita esperimentazioa bultzatu ere simulazio inguruneetan sare trafiko errealista sortuz. Gainera, atzerapen eta sortasun arazoak hobetzen dituen ekarpen berri bat egiten da. Ekarpen honek denboran oinarritutako ADS-en kalitatea hobetzen du, positibo faltsuen ratioa jaitsiz.

Esperimentazio bidez ekarpen ezberdinak balioztatu dira, hainbat metodo estatistiko, ADS-en kalitate neurri eta trafiko erreal eta simulatuak alderatuz. Datu erabilgarriak ulertzeko beharrik gabeko ADS-ak posible direla demostratu dugu, trafiko errealista laborategi kondizioetan sortzea posible dela eta atzerapen eta sortasunaren zuzenketak jarrera modeloen kalitatea hobetzen dutela.

Ondorio bezala, protokolo industrial pribatuekin lan egiteko ADS bat eta jarrera modeloa sortzeko entrenamendu daturik behar ez duen eta KSI-en protokolo irekiekin lan egiteko gai den metodoa aurkeztu dira.

Eskertza

Asko dia azken urtietan bidiakin jarraitxu eta egunerokuan animau dozten pertsonak, zuzenian edo zeharka lagundu eta etsipen momentutan barre bat atera doztenak. Hori dala ta eskerrak emotia gustauko jaten ondorengo pertsona eta taldiei:

Lelengo ta behin Roberto Uribeetxeberria eta Urko Zurutuzai eskertzia nahiko nuke, tesian zuzendari lanetan jardutiaz gain, lan hau burutzeko aukeria emon, gertutasun eta laguntasun tratu bat eskeini, falta ein izan dabenian bultzada bat emon eta hainbat irakaspen eskaintziarren. Era berian Mondragon Unibertsitateko hainbat langilei eta departamentui nere esker ona adieraztia nahiko notzieke.

I would also like to thank the people that I met during my stay at the JRC. First of all Marcelo Masera, Marc Honandel and Christos Siaterlis; thank you for making possible the stay as well as for your help during my visit. Thank you also to Evans Boateng, Andrés Pérez, Francesca De Rigo Conte, Gürcan Gerçek, Bilge Yurderi, Béla Genge, Enis Karaarslan, Marco del Pra and Marko Hribar. Your help and wisdom pushed me to improve myself.

Tesixa burutu bitxartian hainbat gela ta edifiziotan egon naiz, bertan lagun ta lankide ugari izan ditxudalarik, zuei be eskerrik asko: Enaitz, Alex, Joseba, Kepa, Leire, Keldor, Aritz, Iñaki[2], Lorea, Judit, Dani, Alain, Joxe, Unai, Iker, Lorena, Maitane, Idoia, Aitor[2], Irati, Igone, Jon[2], Oscar, Zuriñe, Raul eta Elio. Mila esker deskantso momentutan egunerokotasuna modu erraz baten erutia lagundu doztazuenei, Maite Beamurgia, Kerman Calleja eta azkenik Mikel Iturbe, hasieratik laguntzeko prest eta dokumentuai hainbat begirada emotiarren. Azkenik, Lorea Belategi eta Aitor Urbieta, hainbat afaritxan koziñero moduan fitxau ta beti onduan animuak emoteko egotiarren.

Bestalde betiko lagunei, beti hor egon zatelako ta bizitzako alde honen parte zatelako.

Familixiai, lelengo ta behin gurasuei, naizena naizelako, nigan beti sinistu dozuelako eta bide hau jarraitzeko aukera eskaini doztazuelako. Baita gainontzeko danoi be, aitxita ta amamai, tio, tia ta lengusuei, ta bereziki Xabiri be, astebukaera askotan animuak emon ta entretenitzen laguntziarren. Eta nola ez Maialen Ayastuyri, azken urte hauetan nere albuan egon eta egunero animoz betetziarren.

Azkenik, tesi hau Eusko Jaurlaritzako Hezkuntza, Hizkuntza Politika eta Kultura Sailaren Ikertzaileak Prestatzeko eta Hobetzeko Programaren BFI 09.321 bekaren bitartez burutu da, eskerrik asko.

Contents

List of Figures

List of Tables

List of Algorithms

List of abbreviations

AAKR Autoassociative Kernel Regression

ABB Asea Brown Boveri

ADS Anomaly Detection System

ANN Artificial Neural Network

BASE Basic Analysis and Security Engine

BPF Berkeley Packet Filter

CDF Cumulative Distribution Function

CI Critical Infrastructure

CIP Critical Infrastructure Protection

CIT Communications and Information Technology

COTP Connection-Oriented Transport Protocol

COTS Commercial Off-The-Shelf

CPNI Centre for the Protection of National Infrastructure

CPS Cyber-Physical System

CSV Comma Separated Values

DCS Distributed Control System

DHS Department of Homeland Security

DNP3 Distributed Network Protocol

DPI Deep Packet Inspection

EPIC Experimental Platform for Internet Contingencies

HMI Human Machine Interface

ICC Industrial Cybersecurity Centre

ICS Industrial Control System

ICS-CERT Industrial Control Systems Cyber Emergency Response Team

IDS Intrusion Detection System

IED Intelligent Electronic Device

IO Input/Output

IP Internet Protocol

IPSC Institute for the Protection and Security of the Citizen

ISA International Society of Automation

IT Information Technology

JRC Joint Research Centre

MAPE Mean Absolute Percentage Error

MATLAB MATrix LABoratory

MITM Man In The Middle

MMS Manufacturing Message Specification

MTU Master Terminal Unit

NERC North American Electric Reliability Corporation

NCS Networked Control System

NIC Network Interface Controller

NIST National Institute of Standards and Technology

NTP Network Time Protocol

OPC OLE for Process Control

PCS Process Control System

PDU Protocol Data Unit

PDV Packet Delay Variation

PING Packet InterNet Groper

PLC Programmable Logic Controller

PTP Precision time protocol

PTSL Protocol Trace Specification Language

ROC Receiver Operating Characteristic

RTU Remote Terminal Unit

SAN Sensor Actuator Network

SAN Stochastic Activity Network

SCADA Supervisory Control And Data Acquisition

SHINE SHodan INtelligence Extraction

TCP/IP Transmission Control Protocol/Internet Protocol

TPKT ISO Transport Services on top of the TCP

UDP User Datagram Protocol

UR Update Rate

US United States

VPN Virtual Private Network

WISN Wireless Industrial Sensor Network

Chapter 1

Introduction

This chapter describes the main motivation that inspired the author to research in the area of Industrial Control Systems cyber security in order to provide solutions to the observed problems. Research objectives as well as hypotheses or thesis statements are also established, and the dissertation scope is defined. Finally, the methodology followed to reach the presented objectives is described, and the main contributions of the work are summarized.

1.1 Motivation

Supervisory Control And Data Acquisition (SCADA) systems provide management with real-time data on production operations, implement more efficient control paradigms, improve plant and personnel safety, and reduce costs of operation. These benefits, are made possible by the use of standard hardware and software in SCADA systems, combined with improved communication protocols and increased connectivity to external networks. In 1990s, industry demanded corporate and production networks interconnection [Igu06]. Thus, ICSs evolved from 1970s standalone, proprietary hardware and software to standard operating systems, open source software, TCP/IP based communication protocols, and also connected to the Internet [Hen08; Chr07]. Although this transformation allows the reduction of development, operational, and maintenance costs, as well as the time needed to deploy new installations, the use of COTS hardware and software also implies multiple Information Technology (IT) vulnerabilities [Cun10].

ICSs are widely used in energy, water supply, transport, financial services, health care and telecommunication sectors [Hen08]. Critical Infrastructures (CIs), like nuclear power plants, also rely on ICSs for their operation. As essential as those industries are for

modern society to survive, a malicious and successful attack against a CI could cause catastrophic physical and economic losses, including adverse environmental effects together with the loss of human lifes. Thus, protecting and preserving ICSs against any type of attack is a must.

Among different ICSs, it is a usual trend to find new and legacy devices working together. In fact, ICSs' lifetime is really long compared with traditional IT devices, as they can last for several decades. In addition to that, the coexistence of different vendor devices is also common. The installation of software updates and patches in any Process Control System (PCS) is a complex task due to the fact that many processes must be available 24 hours 7 days per week, specially in devices that control CIs. PCSs' patching cycles are considerably longer and require extensive testing in order to guarantee the reliability of the control network. Thus, software of most devices is deprecated, leaving them unprotected against old and new vulnerabilities. Considering that devices from different periods, brands and software versions work together with few possibility of being upgraded, it is necessary to develop security mechanisms that are appropriate for this kind of requirements.

Specially tailored worms like Stuxnet, Duqu, Night Dragon or Flame [Fal11; McM10; Sym11; Sav12; Fid12] showed their potential to damage and get information about ICSs. Even if only Stuxnet had the capacity to directly interrupt the processes, Duqu, Night Dragon and Flame were also dangerous due to the fact that they were created to spy and obtain substantial information, making possible the creation of specialized worms. Stuxnet was the first one opening the way for a new era; allegedly, governments of United States and Israel were behind its creation [San12], while Iranians' uranium enrichment project was their only target. In fact, they succeeded in their attack showing how precise and advanced a cyber weapon can be. Thus, ICSs are not only individual hackers or crackers' targets, organizations without financial issues found a new way to reach their objectives. This increases the necessity to protect the ICSs.

Last Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) reports [ICS12; ICS13] emphasize the increase of cyber incidents across all CI sectors. The number of attacks increased from nine on 2009 to 198 on 2012; already during the first six months of 2013, there were more than 200 incidents. These numbers highlight the growth of attackers' interest in ICSs. The energy sector has been the most attacked one, as more than 50% of attacks have had it as a target, followed by manufacturing

and communications sectors. Figure 1.1 shows which are the most attacked sectors.

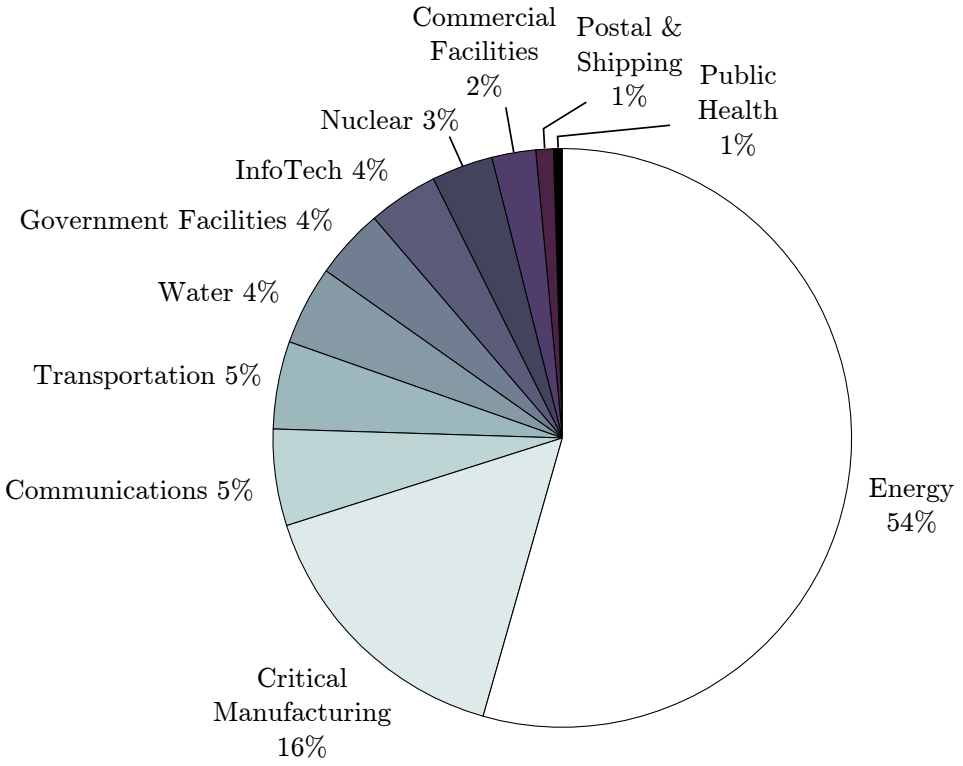


Figure 1.1: Relation between different types of ICSs.

Project Basecamp [Dig13] highlights and demonstrates the fragility and insecurity level of most ICS devices. Its goal is to hit the industry in order to push it to secure ICSs. As a result, many unknown vulnerabilities were published at the same time that Metasploit modules [Rap13] and Nessus plugins [Ten13] were developed. Historically, an attack was a difficult task that required many previous steps, resources and effort. Nowadays, it is relatively easy to perform due to the fact that automatized tools such as Metasploit do most of the hard work. This brings the capacity to perform attacks to users without any special skill; opening the application area to many users, compromising the integrity of CIs.

Searching for and obtaining information about desired targets is the first step of any kind of attack. This task requires some expertise as well as an amount of time that most of unexpert or occasional attackers do not have. SHODAN [Mat09] search engine helps in this duty offering an extended database of devices attached to Internet. Project SHINE (SHodan INtelligence Extraction) [Byr13], related to SHODAN, queries the SHODAN database with specific search terms related to SCADA and ICS products.

So far, over 1,000,000 unique IP addresses that appear to belong to either SCADA and control systems devices or related software products have been collected. All this show the actual situation as well as how easy information gathering is for attackers. This project's final goal is to encourage asset owners to be more proactive regarding security.

IT networks and ICSs differ from each other in different aspects. Availability, reliability and the protocols they use are some of them. While IT systems are typically focused first on confidentiality, then on integrity and finally on availability, ICSs emphasize availability more than integrity and confidentiality. ICSs components usually have memory capacity limitations, preventing the accommodation of relatively large programs associated with security monitoring activities. These differences, their focus and available resources, make traditional IT security mechanisms like firewalls, Intrusion Detection Systems (IDSs) and antivirus programs unsuitable for ICSs.

We can summarize the different aspects already discussed that reinforce the necessity to research in this area and create the necessary security mechanisms in order to minimize threats for society:

- Increasing amount of ICSs' vulnerabilities.
- Possible consequences of successful attacks.
- Long lifetime of ICSs devices.
- Device patching difficulty.
- Worms specially designed to hit ICS devices.
- Significant change on attackers profiles.
- Release of ICS devices' unknown vulnerabilities.
- Databases of unprotected devices connected to Internet.
- Lack of specifically created security solutions.

ICSs are well-known as repetitive and predictable systems and therefore, they are relatively easy to model and in consequence, IDSs make suitable candidates for security mechanisms. This topic has received wide attention from the scientific community; most conducted research has focused on ADSs that require attack-free training data and/or

a long time to create behavioral patterns. The main issues with these approaches are the difficulty to obtain this kind of training data as well as the cost of creating new behavioral patterns each time the process changes.

This thesis presents a new approach to create behavioral patterns of ICS processes in an automatic way. Thus, we demonstrate that it is possible to create behavioral patterns without any attack-free training data. The presented proposal reduces human intervention time and does not need any attack-free training data for pattern creation. This opens a new way for securing ICSs without the need of security minded professionals on site.

1.2 Research Objectives, Thesis Statements, Contributions and Limitations

Network protocols of Industrial Control Systems can be classified into two main groups: private protocols and public ones. Hence, this project has two main parts or directions sharing a main goal: ***to explore, design and develop an Anomaly Detection System for ICSs***. While the first direction is to develop a process to generate signatures based on data of unknown binary protocols, the second one creates behavioral patterns avoiding the necessity of training data for public protocols. These are the hypotheses that this thesis demonstrates:

1. The repetitiveness and stationary characteristic of an ICS can be used to automatically generate robust network behavioral patterns, both in public or private protocols, such that can feed a Network Intrusion Detection System looking for anomalies.

2. Learning behavioral patterns from a real ICS installation requires the network be free of attacks or anomalies. In order to avoid this issue, the network traffic can be generated from a Programmable Logic Controller (PLC) program description. This also solves the time and costs consuming task of setting up or recreating the behavioral models whenever the process is changed.

3. In theory the ICSs behavioral models have to be periodical which makes the de-

tection of anomalies easier. In this case time is one of the parameters that characterizes the behavioral pattern. However, the different system deployments can suffer from clock issues due to small differences in their hardware and software characteristics as well as propagation delays in the communication network. It is possible to correct synchronization issues before the data analysis step performed by the time-based ADS. Solving system clock synchronization problems reduces false positive rates in time-based Anomaly Detection Systems.

Industrial Control Systems are used to control industrial processes that in most cases are repetitive over time, e.g. an automotive component manufacturing process. Hence, network traffic generated by industrial processes follow similar patterns. In this context, we claim that IDSs can be used to detect anomalies instead of intrusions, behaving in an antagonistic way. This is achieved by generating an alert each time a network trace is indescribable by rules instead of generating an alert each time a network trace satisfies a rule description.

In general terms, PLCs can be considered as autonomous devices that follow program guidelines to perform different actions based on environment conditions. Alternatively, Human Machine Interfaces depend on PLCs due to the fact that they just show process conditions controlled by PLCs. Therefore, communication between Human Machine Interface or SCADA server and PLC is performed following a pull strategy, that is, the first one asks the PLC about variable values each time they need to update their state. Based on this idea, we claim that network traffic's behavioral pattern can be predicted having a PLC program description. This assumption avoids training data requirement, therefore, we claim that public protocol's behavioral pattern generation is not training data dependent.

PLCs and Human Machine Interfaces (HMIs) are designed to meet specific requirements with limited hardware resources, they must work in real-time environments where delays are intolerable. In contrast, ADSs require computing resources not available in ICS equipment. Therefore, ADSs cannot be installed on PLCs, demanding another node to host them. It is very common that system clocks lose synchronization over time, this is undesirable for systems that detect anomalies based on time. Hence, we claim that a system able to solve system clock synchronization issues increases ADSs' quality by reducing false positive rates.

This research project will contribute as follows:

- This thesis offers an automatic analysis and signature generation process for ICS binary protocols. Generated signatures can be used by an IDS to detect anomalies including zero-day attacks.
- This project differs from others on the process of creating behavioral patterns of public protocols. Most behavioral pattern creation processes need attack-free training data in order to analyse it and extract a traffic model. The presented pattern creation process uses a PLC program description to define directly the traffic pattern. This improves previous techniques in terms of quality and time.
  - In terms of quality, generated behavioral patterns are not influenced by network issues or unnecessary traffic. They are generated directly as an algorithm output and therefore, they are ideal to perform the task. Network characteristics are taken and applied to the model to become more realistic.
  - In terms of time, the behavioral pattern creation process does not require any previous traffic analysis in order to extract the pattern. The pattern creation process is direct, extracting a behavioral model during the PLC's program development. There are two other benefits directly related with time saving: cost reduction and flexibility. Training data dependent behavioral pattern creation process require a security expert for data analysis and pattern generation, which can be directly translated as an economic cost. In the same way, data analysis and pattern generation consumes a considerable amount of time each time the ICS process changes. Generating patterns directly from a PLC program's description avoids the need for security skills, moreover, it considerably reduces the time required to generate the pattern.
- The ADS used by ICS protocol behavioral pattern is susceptible to systems' clock desynchronization. This work offers an adaptive system to reduce timing errors improving anomaly detection quality by reducing the false positive rate.

The presented behavioral patterns creation processes are focused on some ICS protocols, specially on those that are used on PLC and SCADA servers or HMI communications. Therefore, the following cases are left out of the scope of this work:

1. Communications between PLCs or Remote Terminal Units.

2. Network traffic generated by process alarms is dismissed. Process alarms do not follow any pattern, they are generated in an unexpected way. For instance, an operator working in the plant can push an emergency-stop button or a temperature sensor can exceed a predefined threshold generating an alarm.

3. Network traffic of ICSs is composed of multiple networking protocols. This project is focused on the main protocol used by PLCs, Manufacturing Message Specification (MMS) protocol, leaving out of scope other common protocols such as, ARP, DNS, DHCP, etc.

The first part of the thesis statement focuses on behavioral pattern creation for ICSs's binary private protocols. In this context private protocols are considered unknown for the behavioral pattern creation process.

The second part of the thesis statement is centered on traffic model creation for ICSs's open protocols. Finally, the third part of the thesis statement focuses on the adjustment of the delay and jitter of generated behavioral patterns.

1.3 Methodology

Research context, including available infrastructures, accessibility and proximity of experts, synergies with ongoing research projects and so forth, can completely change the undertaken research methodology. In this sense, our methodology is based on design and creation, experimentation and quantitative data analysis strategies.

We began updating our knowledge by reviewing recent and state-of-the-art publications and also performing the analysis of ICS network traces. Then, at the time that we contributed to knowledge, we designed and developed a signature extraction process as well as a traffic model creation algorithm by widening process scope gradually in an iterative process. Later, the generated process and algorithm were tested by controlled experiments to investigate cause and effect relationships, seeking to prove thesis statements, also known as hypothesis. Experiments were repeated many times, under varying conditions, and the generated process and algorithm were redesigned in an iterative process in order to meet defined requirements. Experimental results were analyzed through quantitative data analysis, using statistics as well as visual aids like tables and

different types of charts, in the way to draw some conclusions. Finally, we disseminated the obtained knowledge and experiences to the scientific community.

1.4 Document Overview

The document is divided into 6 chapters. The first one introduces the reader to the field of the actual ICSs security situation and gives a description of the problems, together with the main motivations that stimulate the author to provide a solution to the problems observed. In this chapter, the main objectives that had to be covered during the development of the thesis work and the hypotheses that respond to the proposed objectives are given. In the same way the main limitations of the thesis as well as the methodology that has been followed are described.

The second chapter gives an overview of Industrial Control Systems, explaining its evolution together with their main components that made up and their differences compared to Information Technologies. In the same way, this chapter provides an extensive overview of the previous related research work. The thesis is focused on ADSs of ICSs, therefore it is important to know previous contributions and identify their advantages together with their disadvantages.

Chapter 3 describes a new approach of an Anomaly Detection System for binary private protocols of Industrial Control Systems. In the same way, it explains which are its main components and how they work, together with an example that makes easier to understand the design of each component. Some results are shown, validating the design in terms of quality.

In Chapter 4, a method to generate realistic network traffic in laboratory conditions without the need of a real ICS installation is introduced. An algorithm is also presented together with the followed methodology for its construction. Later, the method followed for its validation and the obtained results are presented.

Chapter 5 presents a method to correct packet delay and jitter issues. First it discusses which kind of delay and jitter could be find in ICSs is discussed, then it presents an algorithm to correct them and finally it shows the results obtained after the algorithm is applied to the network traffic.

Finally, Chapter 6 provides the conclusions, contributions and future work.

Chapter 2

Related Work

The thesis work grounds on Industrial Control Systems (ICSs) as well as on Intrusion Detection Systems (IDSs). It is therefore necessary to give a background of ICSs and explain their evolution together with their characteristics and architecture and the main differences between ICSs and ITs. Additionally, it is necessary to describe previous investigations on such disciplines to place the topic of this thesis.

2.1 Definitions

In order to introduce the reader into the field of Industrial Control System (ICS) cybersecurity subject, several concept definitions are provided in this section. There are multiple possible explanations for the following concepts, hence, this section tries to empathize those which clearly explain their meaning.

Threat, ***vulnerability*** and ***attack*** concepts are extensively used across the document, thus, it is important to clearly define them.

ISO 27005 [ISO11] defines ***threat*** as:

Definition 2.1. *A potential cause of an unwanted incident, which may result in harm to a system or organization.*

In the same way, the National Institute of Standards and Technology (NIST) [Nat06] defines it as:

Definition 2.2. *Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access,*

destruction, disclosure, modification of information, and/or denial of service. Also, the potential for a threat-source to successfully exploit a particular information system vulnerability.

Vulnerability concept is defined by the ISO 27005 [ISO11] as:

Definition 2.3. *A weakness of an asset or group of assets that can be exploited by one or more threats.*

Additionally, the NIST [Nat06] defines it as:

Definition 2.4. *Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.*

An ***attack***, as discussed by Fovino, Carcano, Masera and Trombetta is defined [Car09] as:

Definition 2.5. *The entire process allowing a Threat Agent to exploit a system by the use of one or more Vulnerabilities.*

And it [Fov09] occurs when:

Definition 2.6. *A threat agent exploits a system by targeting one or more vulnerabilities.*

2.2 Overview of Industrial Control Systems

An Industrial Control System (ICS), as defined by Hentea [Hen08], is a device or set of devices to manage, command, direct, or regulate the behavior of other devices or systems. Also referred as Cyber-Physical Systems (CPSs), ICSs are large-scale, geographically dispersed, federated, heterogeneous, life-critical systems that comprise sensors, actuators and control and networking components [Mit13]. In spite of that, ICS is a general term that involves several types of control systems, e.g. Supervisory Control And Data Acquisition (SCADA) systems, Distributed Control Systems (DCSs), Networked Control Systems (NCSs), Sensor Actuator Networks (SANs), Wireless Industrial Sensor Networks (WISNs), Process Control Systems (PCSs) and Programmable Logic Controllers (PLCs) [Sto13]. These devices are often found in the industrial sectors and Critical Infrastructures (CIs), such as electrical, water and wastewater, oil and gas,

chemical, transportation, pharmaceutical, pulp and paper, food and beverage, discrete manufacturing (e.g., automotive, aerospace, and durable goods) and experimental and research facilities such as nuclear fusion laboratories [MP06].

Throughout the remainder of this document, SCADA systems, DCSs, NCSs, SANs, WISNs, PCS and PLC systems will be referred to as ICSs unless a specific reference is made to one (e.g., field device used in a SCADA system).

Figure 2.1 shows a schema with the relation and main characteristics of different terms used for control systems. As it can be observed, PCSs are very closed to the process, while DCSs are used to refer to a distributed process control system, but in a single location. SCADA term refers to geographically extended control systems, and finally, *ICS* term is used as a general name for all of them.

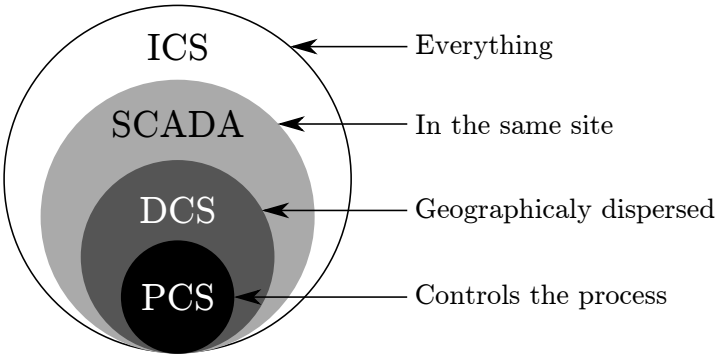


Figure 2.1: Relation between different types of ICSs.

2.2.1 Critical Infrastructures

Critical Infrastructures (CIs) are necessary for any modern society, as they are used to obtain energy, water, and so forth. There are two types of infrastructures, those that could be replaced and those that are indispensable, these later ones are also referred as CIs. As already discussed, CIs are managed by ICSs, thus, securing ICSs implies also to secure CIs. The European Commission defines CIs [Eur04; Fid12] as:

Definition 2.7. *Physical and information technology facilities, networks, services and assets which, if disrupted or destroyed, would have a serious impact on the health, safety, security or economic well-being of citizens or the effective functioning of governments.*

CIs are used by different sectors, which include:

- Communications and Information Technology (e.g. telecommunications, broadcasting systems, software, hardware and networks including the Internet).
- Energy installations and networks (e.g. electrical power, nuclear reactors, oil and gas production, storage facilities and refineries, transmission and distribution system).
- Banking and Finance (e.g. securities and investment).
- Health Care (e.g. hospitals, health care and blood supply facilities, laboratories and pharmaceuticals, search and rescue, emergency services).
- Emergency Services
- Agriculture and Food (e.g. safety, production means, wholesale distribution and food industry).
- Drinking Water and Water Treatment Systems (e.g. dams, storage, treatment and networks).
- Transport (e.g. airports, ports, intermodal facilities, railway and mass transit networks, traffic control systems).
- Critical Manufacturing, production, storage and transport of dangerous goods (e.g. chemical, biological, radiological and nuclear materials).
- Chemical
- Government (e.g. critical services, facilities, information networks, assets and key national sites and monuments).

These infrastructures are owned or operated by both the public and the private sector. For example, in the United States (US) approximately 90% of CIs are privately owned and operated [Sto13]. CIs are ofter referred to as "system of systems", because they are highly connected and highly interdependent. An incident in one infrastructure can directly and indirectly affect other infrastructures through cascading and escalating failures. Electric power is often thought to be one of the most prevalent sources of disruptions of interdependent CIs. Their have become more dependent on common

information technologies, including the internet and space-based radio-navigation and communication. Problems can cascade through these interdependent infrastructures, causing unexpected and increasingly more serious failures of essential services. For example an electric power substation failure could result in large area blackouts that could potentially affect oil and natural gas production, refinery operations, water treatment systems, wastewater collection systems, and pipeline transport systems that rely on the grid for electric power. Interconnectedness and interdependence make these infrastructures more vulnerable to disruption or destruction.

CIs have several security-related requirements [Kal09], which include:

- Secure cooperation. Different organizations should cooperate securely, allowing to share security policies and rules.
- Autonomous organizations. Each organization should be independent even if a global security policy is needed in order to manage the communication between partners.
- Consistency. Global and local security policies should be compatible as CIs are highly interconnected and interdependent.
- Distributed security. Security policies administration should be decentralized.
- Heterogeneity. Several components must be allowed to work together.
- Granularity vs. scalability. Security rules must be extensible while local access controls should be determined locally.
- Fine-grained, dynamic access control. Access control enforcement must be at a low granularity level to be efficient.
- User-friendliness and easiness of rule administration. Several users must be allowed and able to operate the infrastructures.
- External accesses. It is important to determine which would be the allowed external accesses and block all the unauthorized ones.
- Compliance with specific regulations. There are some security concerning regulations, being important to compliance as much as possible.

- Confidentiality, integrity and availability. These are the base of CI operation.
- Enforcement of permission, explicit prohibition as well as obligation rules.
- Audit and assessment. It must be defined which data must be logged, when, where.
- Support, enforcement and real time checking of the contracts that can be established between the different organizations.

2.2.2 The evolution of Industrial Control Systems

Industrial Control Systems have evolved from 1970s proprietary hardware and software, to modern computers, standard operating systems, TCP/IP communications and Internet connectivity. Initially, ICSs had little resemblance to traditional IT systems in that ICSs were isolated systems running proprietary control protocols using specialized hardware and software [Sto13]. In the 1990s, industry demanded corporate and production networks interconnection [Igu06]. Thus, widely available Internet Protocol (IP) devices started to replace proprietary solutions, which bring multiple benefits as well as the possibility of cybersecurity vulnerabilities and incidents [Cun10]. This integration supports new IT capabilities, but it provides significantly less isolation for ICSs from the outside world than predecessor systems, creating a greater need to secure these systems.

In addition, when ICS protocols were first developed, the goal was to provide good performance and the emphasis was placed on providing features that would ensure that the task constraints on the network would be met. Far from being a design requirement, network security was hardly even a concern. Until recently, the most common misconception regarding the security of ICS networks was that these networks were electronically isolated from all other networks and hence attackers could not access them. In addition, the use of standard IT devices has led to the development of a number of ICS protocols that can operate on traditional Ethernet networks and the TCP/IP stack. These protocols are often established serial-line based protocols encapsulated through some standard process prior to being placed in a TCP packet. Many of these protocols abandon any strict master/slave relationships traditionally seen in ICS networks, and devices designed for these networks often provide additional application-layer interfaces beyond the ICS messaging protocol.

It is important to mention that it is a usual trend to find new and legacy devices working together, even if those legacy devices have widely known vulnerabilities [MP06]. In fact, ICSs' lifetime is long compared with traditional IT devices, as they can last for several decades [Oma07]. Besides, the coexistence of different vendor devices is common [Cun10].

While security solutions have been designed to deal with these security issues in typical IT systems, special precautions must be taken when introducing these same solutions to ICS environments. In some cases, new ICS tailored security solutions are needed.

2.2.3 Components and operation of Industrial Control Systems

Industrial Control System components can be divided into two different groups. The first group includes all ICS operation control related components, while the second one consists of networking associated components. Some of the described elements are generally used by SCADA systems, DCSs and PLCs, while others are specific to each one [Düs10; MP06; Sto13; Fov09; Hen08; McG09].

ICS operation control related group is composed of the following components:

- **Operator**. A human operator monitors the ICS and performs supervisory control functions over plant operations.
- **Control Server**. It is responsible for the control software of DCSs and PLCs that communicates with subordinate devices over an ICS network.
- **SCADA Server** or **Master Terminal Unit (MTU)**. Both components act as Masters in ICSs. They contain the high-level control logic for the system and they are the responsible of the control of RTU and PLC devices.
- **Remote Terminal Unit (RTU)**. Also referred as *Remote Telemetry Unit*, it is a special purpose field device, designed to support remote stations. It acts as a slave in the master/slave architecture. It sends control signals to the device under control, acquires data from devices, receives commands from the MTU and transmits the data gathered to the MTU. RTUs provide the same control as PLCs but are designed for specific control applications.

- **Programmable Logic Controller (PLC)**. Small industrial computers with the capability of controlling complex processes. Initially, they were designed to perform the logic functions executed by electrical hardware. Nowadays PLCs are substantially used by DCSs and SCADA systems, especially as they are more economical, versatile, flexible, and configurable than special-purpose RTUs.

- **Intelligent Electronic Device (IED)**. Used to acquire data, communicate to other devices, and perform local processing and control, an IED allows for automatic control at the local level in SCADA systems and DCSs.

- **Human Machine Interface (HMI)**. Referred to the software and/or hardware that allows ICS operators to monitor the process state, modify control settings and respond with manual actions to emergency events. It displays process status information, historical information, reports, and other information to operators, administrators, managers, business partners, and other authorized users.

- **Data Historian**. It is a centralized database for logging all process information within an ICS. Information stored in this database can be accessed to support various analyses, from statistical process control to enterprise level planning.

- **Input/Output (IO) Server**. It is a control component responsible for collecting, buffering and providing access to process information from control sub-components such as PLCs, RTUs and IEDs. An IO server can reside on the control server or on a separate computer platform. IO servers are also used for interfacing third-party control components, such as an HMI and a control server.

Avoiding the network topologies in use, these are the major components of ICS networks:

- **Fieldbus Network**. This network links field components such as actuators and sensors to process controllers. In this way, they avoid the need for point-to-point wiring between each field device and the controller. Multiple communications protocols can be used, while all of them ensure that each field device is uniquely identified.

- **Control Network**. This network supports almost all ICS devices such as, control servers, MTUs, SCADA servers, RTUs, PLCs and so on. It connects the supervisory control level to lower-level control modules.

- **Communications Routers**. A router is a communications device that transfers messages between networks. In ICSs it is used to connect MTUs and RTUs to a long-distance network medium for SCADA communication.
- **Firewall**. A firewall protects devices on a network by monitoring and controlling communication packets using predefined filtering policies. Firewalls are also useful in managing ICS network segregation strategies.
- **Modems**. They are often used in SCADA systems to enable long-distance serial communications between MTUs and remote field devices. They are also used for gaining remote access for operational and maintenance functions.
- **Remote Access Points**. Remote access points are distinct devices, areas and locations of a control network for remotely configuring control systems and accessing process data.

The operation of an ICS is performed by different components, such as **Control Loop**, **HMI** and **Remote Diagnostics and Maintenance Utilities**. Control Loops consist of sensors, controllers, actuators and variables communication. Thus, the sensors measure the variable values and they transmit them to the controllers, which based on set points, interpret the signals. Finally, the controllers transmit to the actuators the actions they should perform. After those actions are performed, as a consequence the process suffer changes which result in new sensor signals that would be transmitted to the controller. Sometimes the Control Loops are nested or cascading, where the set point of a loop is based on the process variable of another loop. Supervisory-level and lower-level loops operate continuously over the duration of a process with cycle times ranging on the order of milliseconds to minutes.

The HMIs, are used by operators and engineers to monitor and configure set points, control algorithms, and to adjust and establish parameters in the controller. In addition they display information about process status and historical changes.

The Remote Diagnostics and Maintenance Utilities are used to prevent, identify and recover from abnormal operation or failures.

2.2.4 Architecture of Industrial Control Systems

Industrial Control System is a general term that involves several types of control systems, such as SCADA systems, DCSs, PLCs [Sto13; Gal13; Bay12; Yan12] and so forth. Even if we referred to all of them as ICS, it is important to clarify which are their main characteristics and differences.

SCADA systems

Supervisory Control And Data Acquisition systems are highly distributed systems which are used to control geographically dispersed assets. While the operation control and data acquisition is performed from a centralized station, these kinds of systems are often spread over thousands of square kilometers. These systems integrate data acquisition systems with data transmission systems and HMI software to provide a centralized monitoring and control system for numerous process inputs and outputs. Thus, the used communication networks are long in distance and usually with a low throughput and big delay. SCADA systems are usually designed to be fault-tolerant systems with significant redundancy built into the system architecture. Field devices are used to control local operations such as opening and closing valves and breakers, collecting data from sensor systems, and monitoring the local environment for alarm conditions. Those field devices communicate with the central station in order to send information and receive supervisory commands.

SCADA systems are usually used in distribution systems such as water distribution and wastewater collection systems, oil and natural gas pipelines, electrical power grids, and railway transportation systems.

SCADA systems consist of both hardware and software. Typical hardware includes a SCADA Server placed at a control center, communications equipment and one or more geographically distributed field sites consisting of either an RTU or a PLC, which controls actuators and/or monitors sensors. Additionally, an IED, that provides a direct interface to control and monitor equipment and sensors, may communicate directly to the SCADA Server, or a local RTU may poll the IEDs to collect the data and pass it to the SCADA Server.

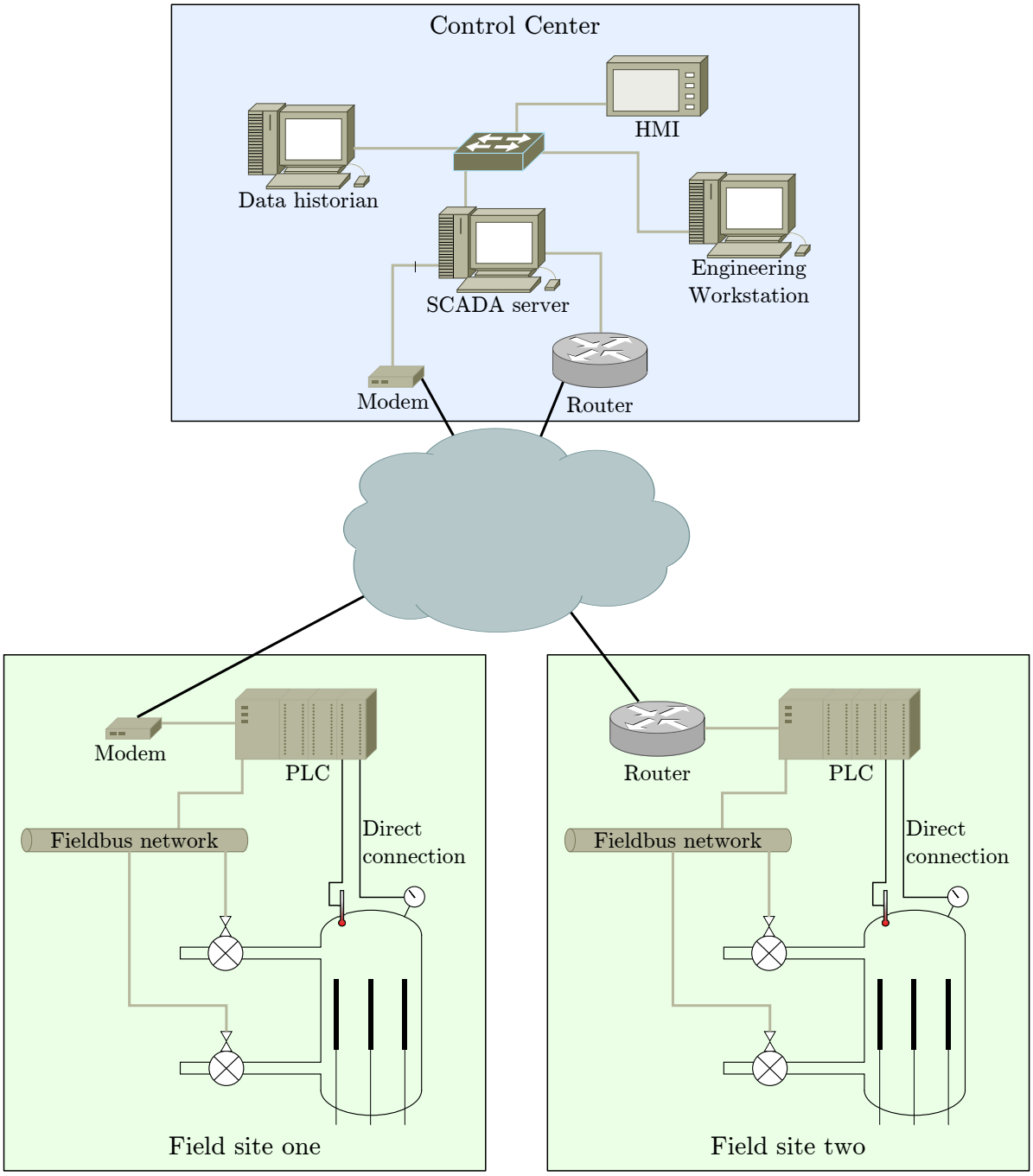


Figure 2.2: Example of SCADA system implementation.

Figure 2.2 shows an example of typical SCADA system implementation. As it can be observed, the installation is composed of two remote field sites and a centralized control center. The connection between field sites and the control center is done by using modem to modem communication or using Internet and several routers. In both field sites PLCs have been used as process controllers.

Distributed Control Systems

Distributed Control Systems consist of a supervisory level of control overseeing multiple, integrated sub-systems that are responsible for controlling the details of a localized process. DCSs are used extensively in process-based and within the same geographic location industries such as electric power generation, oil refineries, water and wastewater treatment, and chemical, food, and automotive production. The control of a process is usually done by feed back control loops, maintaining the process around an established set of points. PLCs are used to control the details of localized processes. As well as in SCADA systems, field devices are used to control local operations such as opening and closing valves and breakers, collecting data from sensor systems, and monitoring the local environment for alarm conditions.

Figure 2.3 shows an example of a typical DCS implementation. As it can be observed, the installation is composed of two PLCs and a common control server. One of the PLCs has been used to control a servo drive that is directly connected to a motor. The other is the responsible of a pressure tank, it has to control the pressure, the temperature and open and close a pair of valves. Similarly, the control network is connected to the Internet through a security firewall and a router. Thus, a remote maintenance could be done by connecting through a Virtual Private Network (VPN) to the control server.

Programmable Logic Controllers

Programmable Logic Controllers are used extensively in almost all industrial processes, including SCADA systems and DCSs, which use them to control local processes. Additionally, in the case of SCADA systems, PLCs provide the same functionality of RTUs. The DCSs use them as local controllers within a supervisory control scheme. When controlled systems are small configurations, PLCs are used to provide operational control

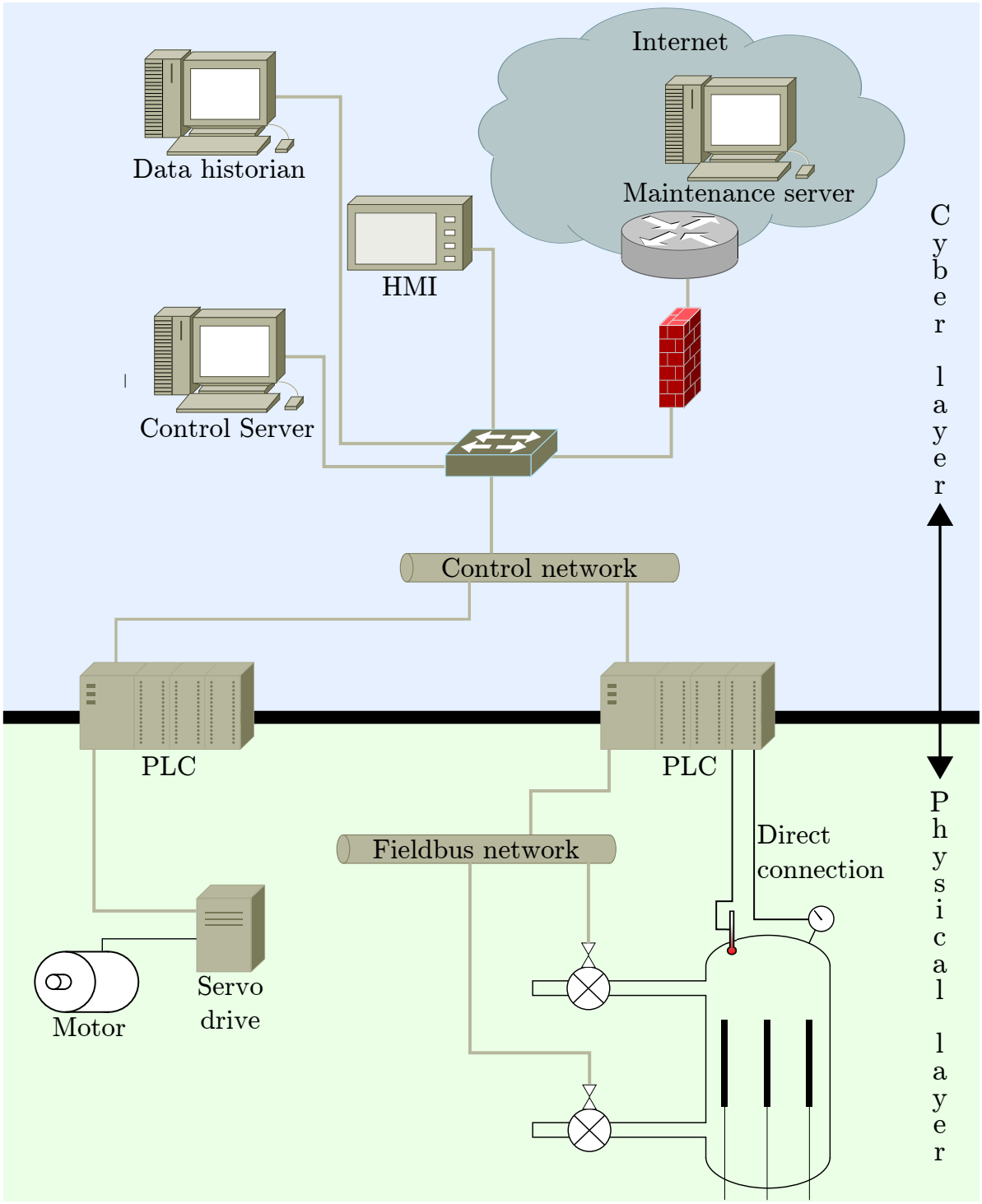


Figure 2.3: Example of DCS implementation.

of discrete processes. In order to provide the program or the control loop they should execute, PLCs are accessible via a programming interface located on an engineering workstation, and data is stored in a data historian. Therefore, PLCs have a user-programmable memory for storing instructions for the purpose of implementing specific functions such as I/O control, logic, timing, counting, three mode proportional-integral-derivative (PID) control, communication, arithmetic, and data and file processing.

As shown in Figure 2.4, the PLC control system implementation is composed of a PLC, some field devices, an Engineering Workstation, a Data historian and an HMI. In this kind of implementation, there is no MTU, SCADA server or control server, the process control is performed by the PLC. The installation can be divided into two different layers: the cyber layer and the physical layer. In modern PLC architectures, one can identify two different control layers: the physical layer composed of all the actuators, sensors, and generally speaking hardware devices that physically perform the actions on the system (e.g. open a valve, measure the voltage in a cable); the cyber layer composed of all the IT devices and software which acquire the data, elaborate low-level process strategies and deliver the commands to the physical layer.

2.2.5 Industrial Control Systems vs. Information Technologies

Information security is usually measured by three different concepts, **confidentiality**, **integrity** and **availability** [Yan12]. Due to the fact that ICSs and ITs are compared, among others, in terms of these three concepts, it is important to define them.

Bishop defines **confidentiality** [Bis02] as:

Definition 2.8. *Confidentiality is the guarantee that the information will only be shared among authorized personnel. Confidentiality presupposes the previous classification of users and of the information to be protected, and also requires adoption of reliable tools for identifying and authenticating users.*

In the same way, Wang *et al.* [Wan13] define it as:

Definition 2.9. *Preserving authorized restrictions on information access and disclosure is mainly to protect personal privacy and proprietary information. This is in particular necessary to prevent unauthorized disclosure of information that is not open to the public*

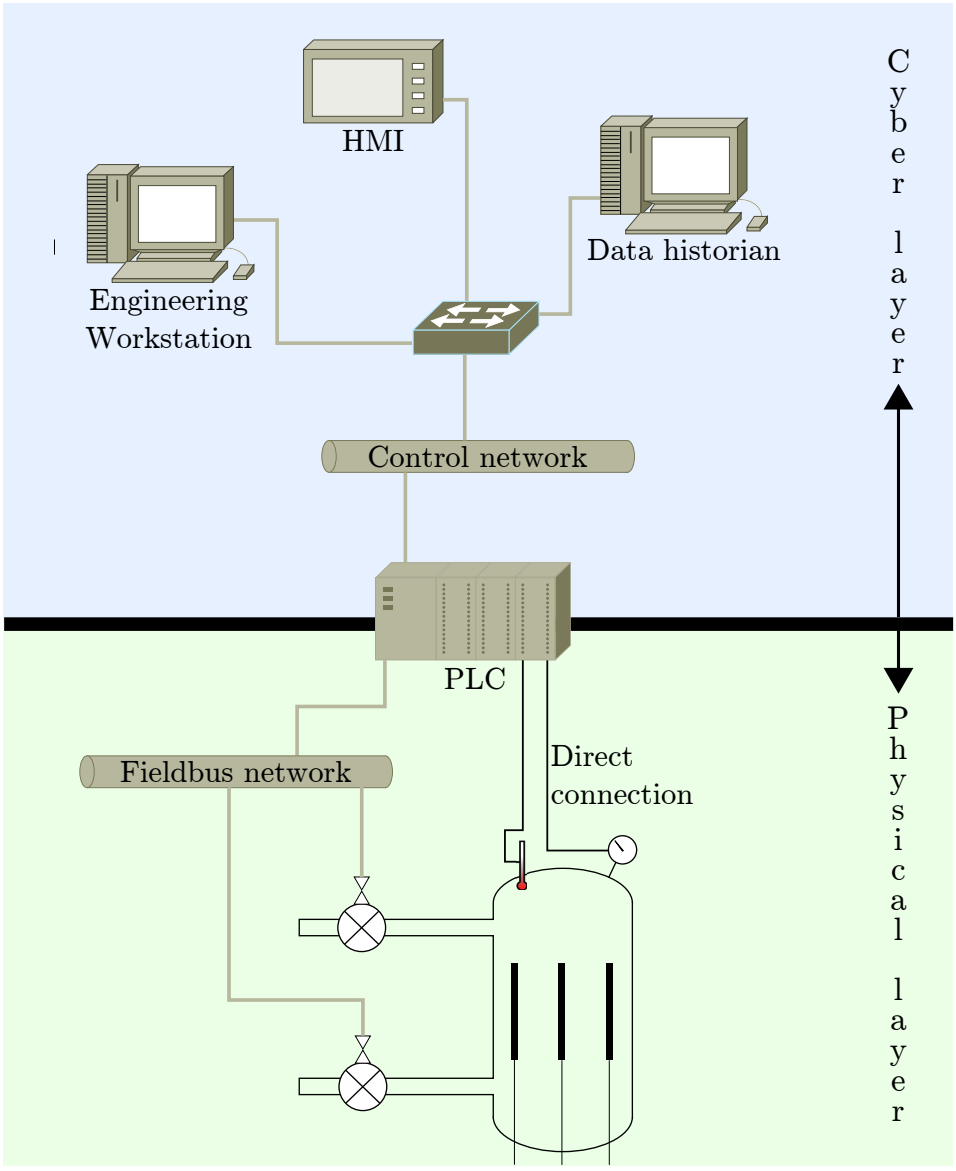


Figure 2.4: Example of PLC control system implementation.

and individuals.

In the same way, Bishop defines **integrity** [Bis02] as:

Definition 2.10. *Integrity is the guarantee that the information is authentic and complete. The available information can not suffer any kind of modification, accidental or malicious. This context includes the guarantee that the information was generated by a trustworthy source.*

In addition, Wang *et al.* [Wan13] define it as:

Definition 2.11. *Guarding against improper information modification or destruction is to ensure information non-repudiation and authenticity. A loss of integrity is the unauthorized modification or destruction of information and can further induce incorrect decision regarding power management.*

Moreover, Bishop defines **availability**[Bis02] as:

Definition 2.12. *Availability, is the guarantee that the information is accessible whenever it is necessary.*

Furthermore, Wang *et al.* [Wan13] define it as:

Definition 2.13. *Ensuring timely and reliable access to and use of information is of the most importance in the Smart Grid. This is because a loss of availability is the disruption of access to or use of information, which may further undermine the power delivery.*

Table 2.1: Differences between Industrial Control Systems and Information Technologies.

Category	**Industrial Control Systems**	**Information Technologies**
Lifetime	10-20 years	3-5 years
Standards	No international standards	International standards, ISO
Performance	Real-time No high throughput No high delay No high jitter	Not real-time High throughput High delay High jitter
Availability	60%	20%
Integrity	35%	30%
Confidentiality	5%	50%
Operating Systems	Legacy systems	Usually new O.S.
Architecture	No standard	Standard
Resources	Limited	Support security software
Networking	Proprietary and standard	Standard
Software updates	Not frequent	Patched systems

As already discussed in Section 2.2.2, ICSs have evolved from proprietary and isolated hardware and software into open and widely available IT solutions. Nowadays ICSs are adopting IT solutions in order to decrease costs as well as to connect to corporate networks, even to the Internet. Therefore, this integration supports new IT capabilities,

but it decreases ICSs isolation, exposing them to typical IT security issues.

Even if the usage of IT solutions is increasing in ICSs, many characteristics differ from traditional IT systems. These differences make necessary new security solutions specially tailored to the ICS environment. Therefore, it is important to discuss the main differences between these two worlds in order to elaborate the most adjusted security solutions for ICSs.

Table 2.1 summarizes the main differences between ICS and IT environments, divided in several characteristics [Kru05; Sto13; Sou13a; Wan13].

2.2.6 Major ICS cyber attacks and incidents

ICSs are widely used in energy, water supply, transport, financial services, health care and telecommunication sectors [Hen08]. CIs, like nuclear power plants, also rely on ICSs for their operation. As essential as those industries are for modern society to survive, a malicious and successful attack against a CI could cause catastrophic physical and economic losses, including adverse environmental effects together with the loss of human lifes.

During the last few decades, several attacks have interrupt ICSs causing diverse consequences. Some of those attacks have directly impacted while other did it indirectly. Due to the lack of evidence available in a number of the examples, it is difficult to determine with certainty if the problem was due to general IT network vulnerabilities or due to ICS related vulnerabilities. The following Table 2.2 some of the incidents occurred over the last decades in ICSs [Mil12; Tur05; Nic12; McA11; Fal11; Sym11; Sav12]:

2.3 IDS quality measures

A behavioral model describes the expected behavior of a system. Behavioral models are used by ADSs in order to detect unusual actions or patterns, in fact, ADSs use behavioral models to compare actual situation with them. Hence, the quality of an ADS depends or is closely related with the quality of the behavioral model it uses [Sve09a].

Table 2.2: Main ICS cyber attacks and incidents.

Year	Incident	*Modus operandi*	Consequences
1982	Siberian pipeline sabotage	Trojan	Economical losses System data modification
1992	Chevron Emergency Alert System	Fired employee resource compromise	Permissions manipulation
1994	The Salt River Project Hack	Trojan Back door	Economical losses Information disclosure
1997	Worcester airport telephone service	Telephone switch cyber attack	Economical losses Service disruption
1999	Gazprom gas pipeline	Trojan	Service disruption
1999	Bellingham gas pipeline	Overload of resources	Three human lives
2000	Maroochy Shire Water	Fired employee resource compromise	Economical losses
2001	California Independent System Operator	Remote access	Information disclosure
2003	Davis-Besse Nuclear Power Plant	Worm	Service disruption
2003	CSX Corporation	Virus	Service disruption
2007	Tehama Colusa Canal Authority	Fired employee resource compromise	Unknown
2010	Iranian nuclear facility	Worm (Stuxnet)	Service disruption
2011	Global energy cyberattacks code-name: Night Dragon	Trojan Exploits Social engineering	Information disclosure
2011	Targeted attacks on Iranian and Sudanese objects	Virus (Duqu)	Information disclosure
2012	Targeted attacks on Iran, Lebanon, Syria, and Sudan	Worm (Flame)	Information disclosure

Sample sets of analyzed systems are used to create behavioral models, even though not all kinds of behavioral models are created in this way. The quality, therefore, the validity of a behavioral model depends on the quality of the used sample set, it must be real, attack-free and diverse enough —all possibilities must be included.

ICSs are known to be repetitive and predictable systems [Gon07], hence, they are easy to be modeled. However, obtaining real and diverse enough data is a difficult task, which results in restrictive behavioral models. Attackers can benefit from a weak training

corpus to bypass an ADS.

The quality of ADSs is closely related with their usage. While a high quality ADS will be highly trusted, a poor one will be ignored and therefore never used. Thus, it is really important to define some common quality measures in order to compare different ADSs in the same terms.

The quality of an ADS is measured by three quality indicators: detection rate, false positive rate and false negative rate. While detection rate, also known as true positives, indicates the number of detected anomalies compared with the total number of anomalies, false positive rate, error type I, is used to show the amount of incorrectly flagged samples —samples flagged as anomalies when they are actually not. False negative rate, error type II, is also used as a quality measure, in fact, it is the opposite of detection rate; it indicates the number of undetected anomalies compared with the total number of them. Figure 2.5 shows all possible events including the normal ones, and the relationship between them.

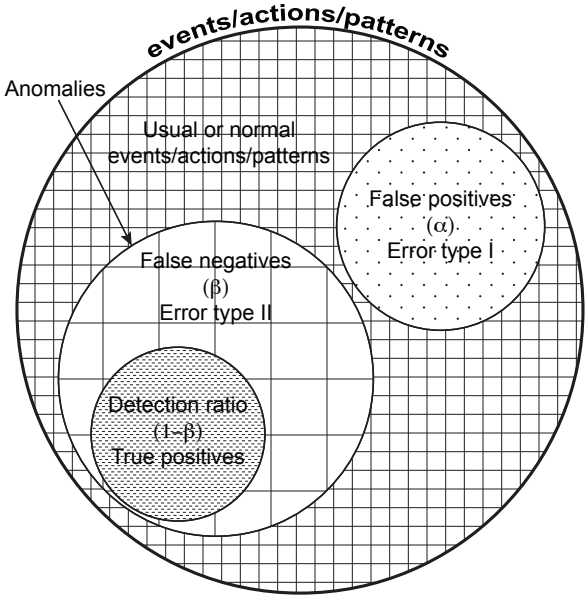


Figure 2.5: Anomaly detection quality measures

In the same way, in the literature Receiver Operating Characteristic (ROC) curves [Mit13] have been used as IDS quality measure. ROC curves describe the relationship between false negative probability and false positive probability obtained as a result of applying IDS techniques.

2.4 Industrial Control System Intrusion Detection Systems

According to [Deb00] Intrusion Detection Systems classification or taxonomy is carried out based on different functional characteristics:

- Detection method
- Behavior on detection
- Audit source
- Detection paradigm
- Usage frequency

In Industrial Control Systems there are two classification characteristics that have been used extensively, the detection method and the audit information source. The first characteristic defines which physical component misbehavior the IDS looks for to detect intrusions. The audit information source instead, defines the source of the information, that in the case of ICSs, could be network or host. In the literature multiple IDS solutions have been described for ICSs. In the following section, some of the works are described, classified by the detection method that they use together with the source of the information that they audit.

2.4.1 Behavior-based Intrusion Detection Systems

We begin by describing the work of Coutinho *et al.* [Cou09; Cou08] which uses **network packets** as the audit information source. The presented work extracts a reduced set of rules from an electrical data base knowledge using the Rough Sets Classification Algorithm described in [Cou07]. The extracted rules describe the traffic flows between the control center and the RTUs. Thus, the presented work is only able to detect the anomalies of the network traffic that affect the control center. Even if the authors emphasize the simplicity and the favorable performance of implementation, the need for a huge amount of attack-free collected data and the possibility to extract the rules just

in offline mode highlights its complexity.

D'Antonio *et al.* [DAn06] presented an IDS that relies on users' behavior extracted from network traffic flows. As well as the IDS, the authors describe a data mining process which is in charge of extracting a behavioral model from traffic flows. Later, the behavioral model is translated into a set of rules which is used by the IDS. In order to capture the traffic flows, the system uses a flow monitor component. This component classifies the traffic by source and destination IP addresses and source and destination port numbers. Even if the work emphasizes its capability, the proposed approach needs an extensive set up process as well as attack-free training data to create the behavioral model.

Shosha *et al.* [Sho11], described a distributed IDS. The presented work describes two different ADSs, the first one used to detect anomalies on usual IT traffic, while the second one has been designed to control specific ICS protocols. Both ADSs rely on Feed-Forward Artificial Neural Networks classifiers during the training process. Additionally, the authors propose an event correlation engine that carries out the process of coordinating between the different anomaly detection engines and correlating events of normal/ suspicious traffic from different sources with different protocols. The correlated values correspond to the protocol, service, source and destination port, event payload, and event timestamps. Even if the authors emphasize the capacity of the presented solution to detect well-known and zero-day attacks at both the control center and substation levels, the system is attack-free training data dependent, which as for the other presented solutions, slow down its implementation.

Gao *et al.* [Gao10] described a neural network based IDS. The presented solution motorizes the traffic between a MTU and a RTU and analyzes the protocol, in this case Modbus, in order to extract the water level value of a water tank. The behavioral model of this physical property is created by using neural networks. Then, the values obtained from network traffic packets are checked against the neural network in order to determine if there is an intrusion. Although the authors expose that neural networks are a promising mechanism to detect attacks, a malicious agent could slowly train the system and change the behavioral pattern in order to perform an attack.

Tsang and Kwong [Tsa05] describe a multi-agent IDS based on a network traffic clustering model. Even if they suggest their work as industrial network IDS, the traffic they

analyze is a compound of traditional IT protocols such as http, ftp, etc. The authors emphasize that their solution significantly improves the overall performance of existing ant-based clustering algorithms. The described work, as well as the others, requires a initial attack-free training data as well as a significant amount of resources to cluster the usual network traffic.

The extensive work of Mahamood *et al.* [Mah10] analyzed different traffic measurement methods and proposed different solutions to apply network traffic monitoring techniques to SCADA systems security. This work focused on the analysis of protocol headers and traffic flows, it grouped them in clusters and extracted useful information using data mining algorithms. Continuing with model-based techniques, Roosta *et al.* [Roo08] presented a design of model-based IDSs for sensor networks used for PCSs. This work defined models at different layers of the network stack, such as the physical, link and network layer. In this way, they identified the following communications patterns: Master to slave, Network manager to field devices, HMI to masters, and Node-to-node communications. Even though the previous approaches do not need to inspect the payload of packets in order to create the models, their two main disadvantages are the need of attack-free training data and the fact that they are unable to predict the size of the packets or the network capacity. These parameters can play an important role in the detection of attacks that are able to change packets, e.g. changing additional registers, without affecting the traffic flow.

Cheung *et al.* [Che06] described three anomaly detection techniques based on three different models. The first one, based on protocol-level models, specified Modbus/TCP protocols' single independent fields, dependent fields (cross-field relationships) and the relationship between multiple requests and responses in order to extract Snort rules. The second technique, called communication-pattern, generated Snort rules by describing communication models used to detect attacks that violate specific patterns. The third and last one, server/service availability technique, created a model of available server and services and detected changes that could indicate a malicious reconfiguration of a Modbus device. In the same way, Düssel *et al.* [Düs10] proposed a payload based real-time anomaly detection system. Their system was protocol independent and it was able to detect unknown attacks. This method took into account the similarity of the communication layer messages from a SCADA network. Although the solution presented by Cheung *et al.* and Düssel *et al.* inspects packets' payload, they are similar to the

work of Mahamood *et al.* in the sense that they need attack-free training data.

Next, we mention the work of Valdes *et al.* [Val09a] that focused on generating communication patterns and proposed an anomaly detection system for PCSs. Patterns that were taken into account included source/destination IP addresses as well as source/destination ports. The system classifies traffic as anomalous if a new pattern shows a lower probability than a specified threshold. In contrast to the previous methods, the approach of Valdes *et al.* does not need attack-free training data, similarly to the approach proposed in this paper. However, it is not able to create the required models without real data and it is not able to predict the size of the packets and the network capacity either.

Premarathne *et al.* [Pre10] present an ADS for Wireless Industrial Sensor Networks. The novelty of the described approach is the way they obtain training data; they use honeypots to obtain real attack traces to use them to launch simulated attacks against their installation, and thus, obtain the training data. The work underlines that the security of attacked hosts is compromised under controlled conditions based upon known attack methods. Later, the captured data during the simulated attacks is classified into different clusters using k-means and c-means clustering methods. The created clusters are used as a behavioral model by the ADS. Even though the authors highlight the increase of its performance compared to statistical outliers detectors, this approach is not able to detect zero-day attacks. In addition, the amount of required resources to perform all types of attacks and cluster their results make it significantly tedious to apply as a real solution.

All the methods that need attack-free training data also require time to obtain it. Every time the process changes, the pattern has to be re-created, which consumes a lot of resources with the impact that it has in the economical costs.

Continuing with behavioral-based IDSs, we describe some works that uses **host files** as the audit information.

The work presented by Dayu *et al.* [Yan06] applies the Autoassociative Kernel Regression (AAKR) methodology to intrusion detection. As the work explains, the AAKR is a nonparametric, empirical modeling technique that uses historical, exemplar observations to make predictions (corrections) for new observations. As the input data, the system uses traffic characteristics as well as system resource utilization statistics, such as link utilization, CPU usage, and login failure. This data is later classified into different

time periods such as day, weekends, holidays. Obviously, in order to create a correct pattern, this methodology needs attack-free training data, including an amount of time to obtain it.

Xue and Yan [Xue07] presented a model-based anomaly detection method for locomotive subsystems. Their system combines offline and online operational modes. Offline modes are used to train the neural network model of the system, while in an online reasoning mode the system calculates deviations between measured outputs and model outputs using three different decision reasoners: Mahalanobis distance, Gaussian mixture mode, and support vector machine. Even if the authors highlight that the approach can successfully detect four out of the five faulty cases while maintaining close to zero false positive based on 43 normal cases, an attack-free training data and the necessary resources to construct the behavioral model make its usage difficult.

Reeves *et al.* [Ree12] present a host-based IDS approach that operates within the operating system kernel. The described solution, which they call Autoscopy Jr., watches for a specific type of control-flow alteration commonly associated with malicious programs. The control flow of a program is defined as the sequence of code instructions that are executed by the host system when the program is executed. Thus, it looks for a certain type of pointer hijacking, where a malicious function interposes itself between a function pointer and the original function to which it pointed. The hijacking causes the pointer to point to the malicious function, which then calls the original target function of the pointer at some point within itself. Autoscopy Jr. operation is divided into learning and detection phases. During the learning phase, the system scans the kernel for function pointers to protect and collects information about "normal" behavior on the system. Later on, the created pattern is used to check the veracity of all system calls. Although the authors highlight the power of the presented approach to reduce the overhead of the host as benefit, this solution is attack-free training data dependent. Additionally, as well as the other presented solutions, it imposes a high initialization overhead every time the analyzed process is changed.

Next, Hadžiosmanović *et al.* [Had10] present MEDUSA system, a host-based IDS which is focused on the detection of process-related threats. MEDUSA automatically analyzes system logs, detects and alerts users about situations in which the system behaves inconsistently with past behavior. The process is divided into two steps: learning phase and detection phase. During the learning steps, first of all, it uses algorithms for mi-

ning outliers to detect individual actions and events that are significantly different from previous entries. Then, it analyzes sequences of events and actions in the logs to provide a better view on the situation context. During the detection phase, they use the created model to analyze the situation in real-time. The presented approach is based on historical attack-free training data to construct the behavioral model. In addition, every time the process is changed the behavioral model has to be reconstructed which requires time to create historical data.

Finally, Rabatel *et al.* [Rab10] presented an ADS that monitors field sensors state in order to detect process anomalies. In fact, the authors do not present an IDS itself, but their solutions could be used for that purpose. They focus their work on system maintenance, and the cost saving originated from an efficient sensor maintenance. Their contribution is divided into three different parts. First of all, they propose an adequate representation of data in order to extract knowledge based on sensor data describing the past normal behavior of systems. The behavioral pattern is created based on historical data saved in logs. Second, they study the possibility of extracting sequential patterns in historical data thus improve understanding of systems for experts, but also to provide a knowledge database for use in the development of an anomaly detection method. Finally, they propose an approach to compare new journeys with sequential patterns describing normal behavior. The proposed solution trusts on the historical data in order to create a behavioral pattern. Thus, it is attack-free training data dependent.

2.4.2 Behavior-specification-based Intrusion Detection Systems

We continue describing some IDSs that use behavior-specification-based detection method and use **network packets** as audit information.

Svendsen *et al.* [Sve09a] presented a statistical ADS for SCADA systems. The described system is focused on statistical variations of a liquefied natural gas system measurements. The system compares the actual system measurements against fixed limits, historical observations, correlation between two or more metrics, and created Markov models, which give the probability of moving from one state to another. Thus, every time a measurement is out of the predefined limits an alarm is raised indicating that an anomaly has occurred. Although the authors underline that the presented approach is well suited to detect anomalies, the system requires a lot of initial resources to create

the behavioral model. Additionally, attack-free training data is necessary during the pattern creation process.

Valdes *et al.* [Val03; Val09a; Val09b] present a work to demonstrate that anomaly detection, and specifically methods based on adaptive learning, can provide a useful intrusion detection capability in process control network. They describe two anomaly detection techniques, pattern based anomaly detection and flow-based anomaly detection. In pattern based anomaly detection they use patterns formed from source and destination IP addresses and destination port. They evaluate patterns against a pattern library in order to find the more similar one. The most important feature of this technique is that it does not need attack-free training data. In the case of flow-based anomaly detection, they define a flow in terms of its source and destination IP address and destination port. Also, they have established that flows are unidirectional. They maintain a database of active and historical flow records and these records are evaluated against learned historical norms. In order to test the two approaches, they have used a test environment that is based on DCS from Invensys Process Systems. The obtained results indicate that the flow-based anomaly detection technique is able to detect anomalous flows effectively. The experiments have been conducted in a simulated SCADA system composed of several SUN servers and workstations on a local network. As a conclusion, the experiments have demonstrated that this methodology can quickly detect anomalous behavior. The main disadvantage of this solution is than an attacker could slowly train the system in order to include attack patterns as common ones. This would bypass the anomaly detection engine.

In the same way, there are some works that use **host files** as audit information source.

Bigham *et al.* [Big03] described two different ADSs. The first one is based on the n-gram technique described by Damashek *et al.* [Dam95] while the second ADS is based on invariant induction technique. Both techniques were tested in an electricity management network. The invariant induction technique builds up a normal model of the data by looking for relationships between the real and reactive power flow measurements for a six bus network. In the same way, the n-gram technique with up to four characters has been used to create the model by applying it to real and reactive power flow measurements for a six bus network. The presented results remark that while the performance of the n-gram technique is better with small numbers of changes of measurements, the invariant induction technique has better overall performance on the identification of

errors. The main disadvantage of both techniques is that they are attack-free training data dependent. They also require a lot of initial resources to set up the system.

Continuing with invariant induction technique, McEvoy *et al.* [McE10] exposed an ADS that uses as input the ICS sensor readings. In fact, they define relationships between sensor readings using non-linear structural equations to build a causal model. Later on, this causal model is used to detect anomalies of sensor readings as well as manipulated computational states, actuator settings and data values. As the authors expose, although the presented approach have a good anomaly detection performance, it needs extensive modeling for design and implementation process.

Finally, Oman and Phillips [Oma07] presented an IDS monitoring system that permits the process engineers to identify malicious and anomalous device configurations. The system is capable to automatically obtain the configuration of devices and compare them with other configurations which is useful for SCADA operators. The presented IDS is able to automatically generate rules based on devices configuration expressed in XML format. As the authors expose the current prototype automates intrusion detection and settings retrieval only for RTUs.

2.4.3 Knowledge-based Intrusion Detection Systems

We start by describing those works that use **network packets** as audit information source.

Verba and Milvich [Ver08] presented an hybrid IDS for ICS networks. Their approach use traditional signature matching together with the analysis of data and command flows to detect potential exploitation vectors, unauthorized commands and Man In The Middle (MITM) attacks. They emphasize the strength of the combination of multiple approaches in order to create a unique solution that could detect multiple attack sources. Additionally their work uses DPI techniques in order to extract protocol state information. Their work is only able to detect anomalous connections and malformed packets together with MITM attacks that can compromise the control server. However, in order to detect anomalous connections and malformed packets, the IDS have to be able to understand the used ICS protocols and the matching rules have to be deployed for each installation. Furthermore, the detection of MITM attacks also makes it necessary to

specify all the connections that the control server can support. These handicaps make the real implementation of the solution slow and lazy.

The extensive work of Mitchell and Chen [Mit13], describes an IDS based on behavioral rules. The presented approach uses several IDS modules that run in every device. Before the IDS starts working, it is necessary to define a set of rules describing the system behavior. Those rules describe different system states as well as each devices network traffic characteristics. Within network characteristics, the number of received and sent packets as well as the networking conditions are included. Although the presented solution is able to detect zero-day attacks, it forces to define with rules all the system state possibilities in order to not dismiss any possible attack trace. The definition of those rules demands a vast initial work together with the amount of resources needed each time the process changes to redefine the rules.

Next, Carcano *et al.* [Car10; Car11; Fov10] exposed an IDS based on process specification. The presented approach is able to detect the process state of the system and use it in order to identify Modbus and DNP3 protocols commands that could produce a critical situation. The presented solution is composed of five elements: a Modbus and DNP3 protocols sensor, a rule data base, a virtual image of the system, a system state analyzer and another data base with critical rules. The authors also present a rule language specifically designed in order to describe Modbus and DNP3 signatures and field device states. The system operates by following the real system state by the analysis of the captured network traffic and the update of the virtual system state. This traffic is also analyzed in order to determine if the actual commands could push the real system into a critical state based on the state of the virtual system. Although the presented results validate the approach, its main disadvantage is the amount of work that is required to create a virtual system together with all the necessary rules. Additionally, each time the process is changed, the rules as well as the virtual system have to be updated. As a positive point, the presented approach does not need attack-free training data.

Gonzalez *et al.* [Gon07] described a Modbus protocol passive scanner which provides valuable information about the state of Modbus networks. The presented scanner analyzes Modbus protocol function codes, transaction states, memory access and memory contents in order to detect anomalous activity. The passive scanner is composed of three different components: a network scanner, a Modbus transaction checker, and an incremental network mapper. The first component is used to capture network traffic.

The Modbus transaction checker, pairs matching sets of messages while the incremental network mapper uses the collected information to populate dynamic data structures that store network topology and status information. The passive network scanner requires an initial set up where all the Modbus protocol possible transactions have to be defined. Although this system is able to detect protocol-based intrusions, the required amount of work for each installation make its implementation difficult.

Rrushi and Campbell [Rru09] presented a rule-based IDS able to detect attacks on nuclear power plants. They follow a process in which Stochastic Activity Networks (SANs) are used to model and analyze the operation of a boiling water reactor. Later, those SAN models are derived into intrusion detection rules that describe the values of Modbus protocol data items and Protocol Data Unit (PDU) fields to determine whether incoming PDUs are legitimate or malicious. Therefore, even if they do not emphasize, their work uses Deep Packet Inspection, and as well as for Modbus protocol, it could be extended to use other ICS protocols. The main disadvantage of the proposed approach is the vast amount of work that supposes the creation of the operational model specification of the boiling water reactor.

And finally, there are some works which analyze **host files** to detect intrusions.

The work presented by Svendsen and Wolthusen [Sve09b] describes an ADS based on a physical system model. The exposed work uses a non-linear model of a real hydroelectric power plant to detect the anomalies originated by some attack vectors used to test the system. Particularly, deviations in the expected correlations between variables in the model are used to detect direct manipulations of sensors. Even if the presented approach is able to detect different types of attacks, it needs extensive resources to design and implement the non-linear model.

Gaspary, *et al.* [Gas05] present and SNMP protocol based intrusion detection system. They also present the protocol trace specification language Protocol Trace Specification Language (PTSL), a language for the representation of protocol traces based on the concept of finite-state machines (FSMs). Thus, they represent attack signatures using PTSL language. They rely on a distributed architecture of several intrusion sensors and they explored, through a case study, the platform to validate its applicability. The main issues of this approach are that they rely on the protocol traces of the SNMP protocol that they generate in order to detect attacks. Thus, in order to apply it in an

ICS they should analyze and create all the communication patterns of SNMP protocol of an installation, which highly slows down its applicability. In addition, not all the ICS installations have SNMP protocol enabled and even if they have it, the presented approach could not detect all the attack events because it is just focused on one protocol.

Zhang, Yu and Hardy [Zha11] present a distributed network sensor-based intrusion detection framework that monitors network traffic and hosts. They correlate events from both information sources, host based IDS and network traffic, in order to efficiently detect attack traces. Also, they generated a prototype to demonstrate its efficiency. Even if they use Deep Packet Inspection (DPI) [Byr12] to efficiently process network traffic, they do not analyze industrial protocols. In addition, they focus just on packets headers. The presented work is not able to detect unknown attacks, or zero day attacks.

Chapter 3

An Anomaly Detection System for ICS network binary private protocols

The objective of this thesis work is to explore, design and develop an Anomaly Detection System (ADS) for Industrial Control Systems (ICSs). There is a huge variety of ICSs around the world. They can be classified according to the geographical area they cover, the types of network protocols and devices they use, etc. The existing variety of ICSs requires different types of ADSs. This diversifies the research areas as well as demands a greater effort to construct robust security mechanisms.

Currently, approximately ten network protocols dominate the industrial marketplace; MODBUS, DNP3, EtherNET/IP , PROFIBUS and Foundation Fieldbus are the most used ones among them [Byr04]. ICS network protocols can be divided into two main groups: public and private ones. Private protocols, also called *unknown* protocols, are those whose details/specifications are only known by its creators, thereby, unknown for the rest, e.g. Siemens S7 protocol. In contrast, public protocols have open specifications, providing the possibility to be implemented by each vendor and thus, being widely used by different manufacturers, e.g. Modbus, DNP3. Even if most of these industrial marketplace dominating protocols are public, there also are private ones, requiring research and security solutions.

In this chapter, a new approach for an Anomaly Detection System for binary private protocols of Industrial Control Systems is presented. The introduced ADS is divided into six components, thus, a detailed description of each one is given. In addition, its implementation and followed steps for its validation are presented. Due to the fact that presented ADS is based on the assumption that network traffic is repetitive for ICS, in the first section a real network traffic capture is analyzed.

3.1 ICS network traffic repetitiveness

In the following section a real ICS network traffic capture is explored. By focusing on some features such as where each packet is originated, which is its destination and payload content, a clear repetition of packets can be observed. The stationary and repetitiveness characteristics of ICS network traffic establishes the bases for the ADS presented in this chapter.

Table 3.1 shows an example of features extracted from a network traffic capture file. Note that different colors and symbols have been used in order to highlight the small variations within the repetitive patterns. The captured file was obtained in a real manufacturing plant, where the ICS network is composed of a Siemens S7 PLC, a Siemens switch and a SCADA server that also has capturing software in it. During the traffic capture, no anomalies have been observed, keeping the captured data free from attacks. The following Figure 3.1 shows the network architecture used for the traffic capture. It is clear that in a switched network each device will only obtain traffic originated from it or traffic that has its address as destination, including broadcast traffic. Even if this hides what is going on in the rest of the network, the intention of this traffic capture is to show how repetitive and, therefore, predictable an ICS traffic is. Problems originated by switching environments could easily be solved by most switches' port mirroring feature. This characteristics allows to obtain all the traffic that goes through the switch.

The network traffic capture file includes all kind of packets, including common network protocols (ARP, ICMP), as well as industrial protocols, in this case Siemens S7 protocol. Table 3.1 shows some selected features of Siemens S7 protocol packets, while, the rest of protocols are filtered out. The presented features include:

1. Captured packet number
2. Timestamp
3. IP address of the source device
4. IP address of the destination device
5. Payload length
6. Payload

Table 3.1: Example of captured data.

Packet number	Timestamp (seconds)	Source IP	Destination IP	Payload length (bytes)	Payload
1	1311687257	192.168.1.240	192.168.1.2	84	32010007 š 68004A00000406120A10020020012C840005C012 ...
2	1311687257	192.168.1.240	192.168.1.2	96	32010007 ȯ 68005600000407120A1002000A012C84000BC012 ...
3	1311687257	192.168.1.2	192.168.1.240	222	320300007 5̄ 68000200D000000406FF0401000000000000000 ...
4	1311687257	192.168.1.2	192.168.1.240	158	320300007 ö 6800020090000000407FF0400500000000000042 ...
5	1311687257	192.168.1.2	192.168.1.240	236	320700000000000C00D60001120812020111000000000FF0900 ...
6	1311687257	192.168.1.2	192.168.1.240	85	320700000000000C003F0001120812020010D00000000FF0900 ...
7	1311687257	192.168.1.240	192.168.1.2	84	32010007 ž 68004A00000406120A10020020012C840005C012 ...
8	1311687257	192.168.1.240	192.168.1.2	96	32010007 ȯ 68005600000407120A1002000A012C84000BC012 ...
9	1311687257	192.168.1.2	192.168.1.240	222	320300007 7̄ 68000200D000000406FF0401000000000000000 ...
10	1311687257	192.168.1.2	192.168.1.240	158	320300007 ö 6800020090000000407FF0400500000000000042 ...
11	1311687257	192.168.1.2	192.168.1.240	236	320700000000000C00D60001120812020111000000000FF0900 ...
12	1311687258	192.168.1.240	192.168.1.2	144	32010007968008600000040B120A10020004012C8400004012 ...
13	1311687258	192.168.1.240	192.168.1.2	48	32010007A68002600000403120A1002001E012C84002A8012 ...
14	1311687258	192.168.1.2	192.168.1.240	224	320300007968000200D20000040BFF040020C21311852892244 ...
15	1311687258	192.168.1.2	192.168.1.240	90	320300007A680002004C00000403FF0400F044218082220A0A ...
16	1311687258	192.168.1.240	192.168.1.2	84	32010007 B̌ 68004A00000406120A10020020012C840005C012 ...
17	1311687258	192.168.1.240	192.168.1.2	96	32010007 Ċ 68005600000407120A1002000A012C84000BC012 ...
18	1311687258	192.168.1.2	192.168.1.240	222	320300007 B̄ 68000200D000000406FF0401000000000000000 ...
19	1311687258	192.168.1.2	192.168.1.240	158	320300007 C̈ 6800020090000000407FF0400500000000000042 ...
20	1311687258	192.168.1.2	192.168.1.240	236	320700000000000C00D60001120812020111000000000FF0900 ...
21	1311687258	192.168.1.2	192.168.1.240	85	320700000000000C003F0001120812020010D00000000FF0900 ...
22	1311687258	192.168.1.240	192.168.1.2	84	32010007 Ď 68004A00000406120A10020020012C840005C012 ...
23	1311687258	192.168.1.240	192.168.1.2	96	32010007 Ė 68005600000407120A1002000A012C84000BC012 ...
24	1311687258	192.168.1.2	192.168.1.240	222	320300007 D̄ 68000200D000000406FF0401000000000000000 ...
25	1311687258	192.168.1.2	192.168.1.240	158	320300007 Ë 6800020090000000407FF0400500000000000042 ...
26	1311687258	192.168.1.2	192.168.1.240	236	320700000000000C00D60001120812020111000000000FF0900 ...

group of repeated packets repeated packet individual packets

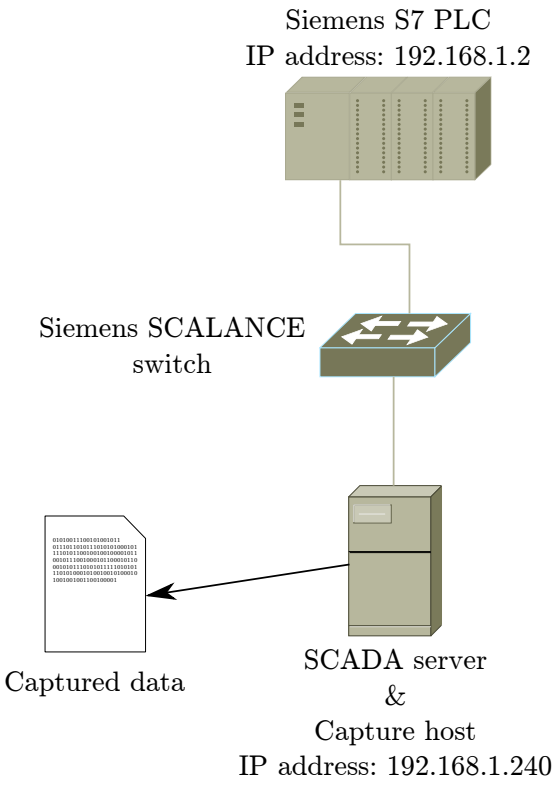


Figure 3.1: Industrial Control System where network traffic was captured.

Just a small section of the payload is shown as we consider it is enough to explain the main ideas that concern this traffic capture. The following figure 3.2 shows the protocol stack of Siemens S7 protocol as well as the layers from where illustrated features, excluding packet number and packet's timestamp, are extracted.

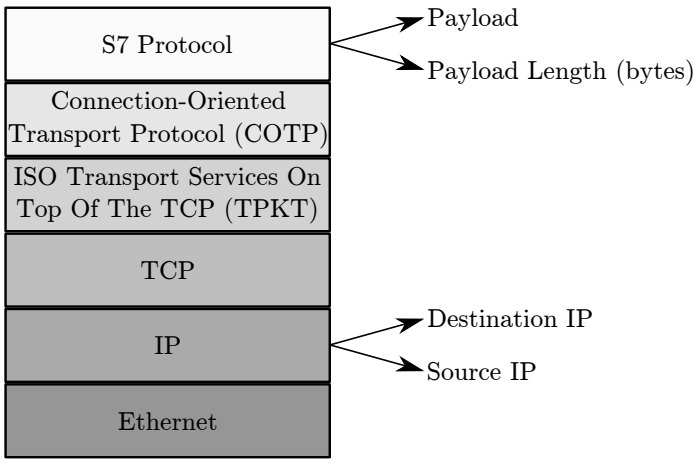


Figure 3.2: Siemens S7 protocol over TCP.

Looking at the extracted features, it can be observed that the 26 captured packets were cached in a time interval of two seconds. In addition, all the packets correspond to a communication between just two devices, the PLC and the SCADA server. Those packets whose source IP address is 192.168.1.2 were originated by the PLC, while 192.168.1.240 source IP address corresponds to packets originated by the SCADA server. The PLC sent 16 out of the total 26 packets, which represents the 62% of the traffic. Further, we can observe that packets originated by the PLC are bigger in size (236, 224, 222, 158, 90, 85 bytes) than those sent by the SCADA server (144, 96, 84, 48 bytes). This is mainly because the PLC sends process value updates to the SCADA server.

Focusing on the size of packet payloads and their content, we can observe that there are several packets of the same payload length. In fact, packets with a payload length of 236, 84, 96, 222, 158 bytes are repeated four times during the capture. Other packets, like those with 85 bytes' payload length are repeated twice and the rest of the packets (144, 48, 224, 90 bytes) are unique. Considering just repeated packets, and focusing on their payload, packets can be classified into two different groups; those whose payload change over repetitions and those that are always the same. 236 and 85 bytes payload length packets are those whose payload does not change over the time. The rest of the repeated packets change the content of their payload, but as shown in Table 3.1, differences between the same length packets are minimal, changing only one byte in the shown section of the payload.

The traffic of ICS networks, as discussed in Chapter 2, is considered repetitive and quite static. The example of captured data showed in table 3.1 is a clear illustration of how repetitive is an ICS traffic. These are the main features that support this idea:

1. Static payload packets are repeated over time.
2. Some packet are repeated in an established order.
3. Even if payload of different packets is not the same the differences are small.

3.2 Anomaly Detection System for ICS binary private protocol

In this section, an Anomaly Detection System for ICS binary private protocols is described. One of the positive aspects of the designed ADS is that due to the fact that it does not try to understand the content of the packets, it is able to work with different binary protocols.

Most Anomaly Detection Systems require learning normal traffic patterns as a reference to make a comparison with the monitoring traffic, hence looking for differences that will show behavioral anomalies. Hence, the behavioral model has to be constructed before an ADS starts to work. The presented ADS has two ways of operation, offline and online. Offline mode is made up of elements that permit the creation of the behavioral model and synthesize it on rules, it is used each time the monitored process changes and a new behavioral model has to be created. In addition, the online mode is composed of elements that compare the actual situation with the behavioral model created in offline mode and generates alerts in case an anomaly has been detected.

Figure 3.3 summarizes the two ways of operation as well as the six components that compose the ADS. The offline mode is composed of the network traffic capturer, the feature extractor, the data assembler and the rule generator. In the other hand, the online mode is made up of two components, the anomaly detection system and the notification center.

3.2.1 Components of the Anomaly Detection System

This section presents a detailed description of each ADS module or component. In order to make it easier to understand the design of each component, an example is used to clarify what the purpose of the component is.

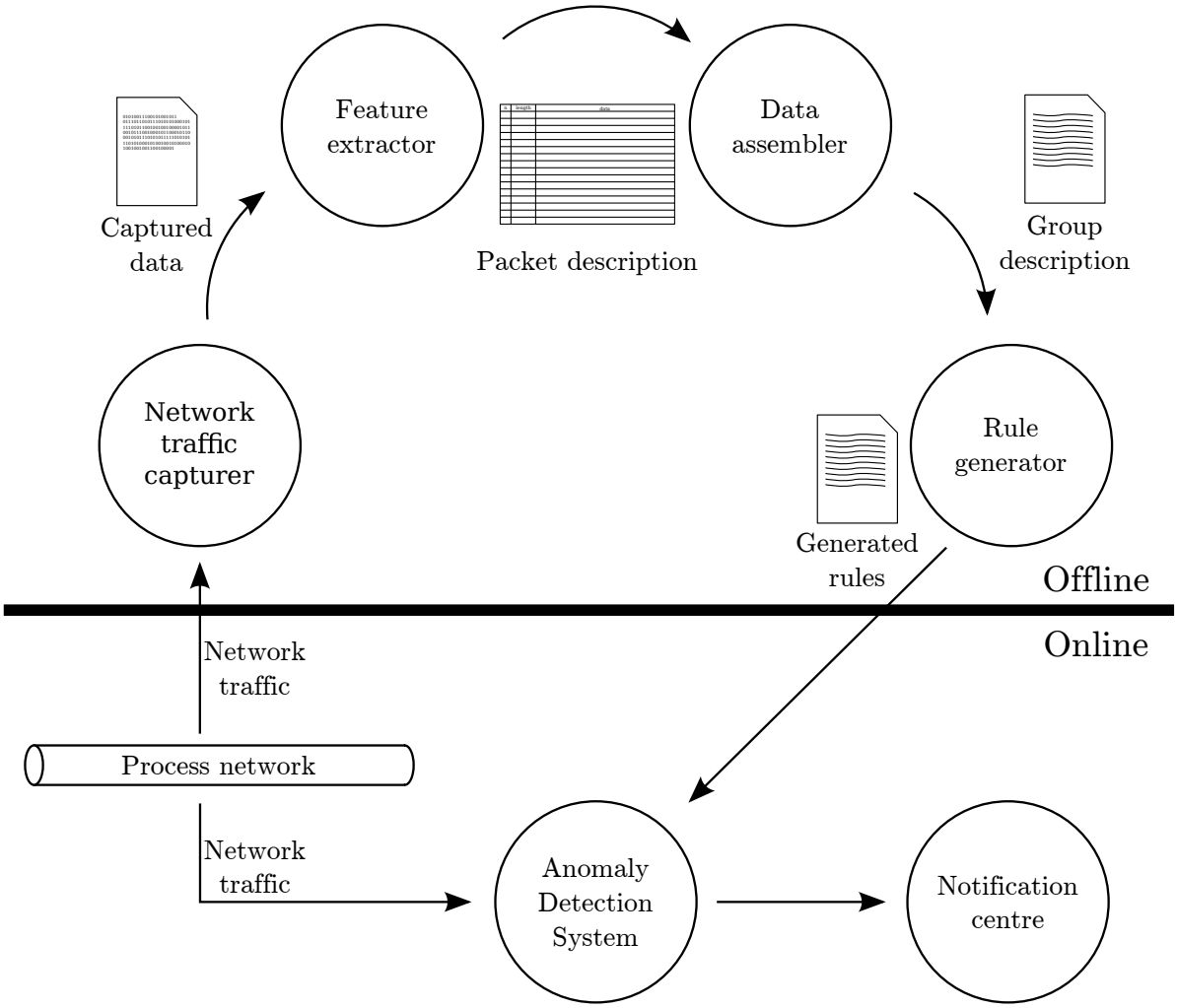


Figure 3.3: Component diagram of the Anomaly Detection System.

Network traffic capturer

The network traffic capturer component, part of the offline mode, is responsible for capturing network traffic, filtering out and saving all the data in a capture file. Depending on the network section where the network traffic capturer is located, the behavioral model will be different. Therefore, it is really important to select the network section from which the anomaly detection system module will monitor the traffic.

In addition, the network traffic capturer filters all the captured packets and discards those that do not satisfy the established conditions. In this case, due to the fact that we are interested just in Siemens S7 protocol, it filters out all common network traffic,

including ARP and DNS packets. To do so, and because Siemens S7 protocol uses 102 port number, it leaves just those packets that have the 102 port as source or destination port.

Finally, the network traffic capturer stores all the packets in a binary file that can be used by other components to read captured data in offline mode.

Feature extractor

The feature extractor component takes as its input the capture file, created by the network traffic capturer component, and extracts all the features needed to analyze the traffic and generate the behavioral model. To do so, it reads the packets one by one and extracts the following features.

- Packet number
- Timestamp
- Source IP address
- Destination IP address
- Payload length
- Payload

As shown in Table 3.2, the feature extractor assigns sequentially a packet number to each captured packet. The timestamp is the time the packet is read by the capture, represented in Unix time, the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970, not counting leap seconds. Source IP address refers to the IP address of the device where the packet was created. In the same way, Destination IP address refers to the IP address of the device which packets were to. The payload length refers to the length of the portion of the packet that corresponds to Siemens S7 protocol. Payload length is calculated taking the entire packet length and subtracting the header sizes of the rest of the protocols.

Finally, the feature extractor component saves all the extracted features, so the assembler module can use them.

Table 3.2: Example of captured data manipulated by the feature extractor.

Packet number	Timestamp (seconds)	Source IP	Destination IP	Payload length (bytes)	Payload	
1	1311687257	192.168.1.240	192.168.1.2	84	32010007568004A00000406120A10020020012C840005C01	...
2	1311687257	192.168.1.240	192.168.1.2	96	3201000766800560000040712 0A1002000A012C84000BC01	...
3	1311687257	192.168.1.2	192.168.1.240	222	32030000756800020 0D000000406FF040100000000000000	...
4	1311687257	192.168.1.2	192.168.1.240	158	32030000766800020090000000407FF040050000000000004	...
5	1311687257	192.168.1.2	192.168.1.240	236	320700000000000C00D60001120812020111000000000FF09	...
6	1311687257	192.168.1.2	192.168.1.240	85	320700000000000C003F000112081202010D00000000FF09	...
7	1311687257	192.168.1.240	192.168.1.2	84	32010007768004A00000406120A10020020012C840005C01	...
8	1311687257	192.168.1.240	192.168.1.2	96	3201000786800560000040712 0A1002000A012C84000BC01	...
9	1311687257	192.168.1.2	192.168.1.240	222	32030000776800020 0D000000406FF040100000000000000	...
10	1311687257	192.168.1.2	192.168.1.240	158	32030000786800020090000000407FF040050000000000004	...
11	1311687257	192.168.1.2	192.168.1.240	236	320700000000000C00D60001120812020111000000000FF09	...
12	1311687258	192.168.1.240	192.168.1.2	144	3201000796800860000040B120A10020004012C840000401	...
13	1311687258	192.168.1.240	192.168.1.2	48	32010007A68002600000403120A1002001E012C84002A801	...
14	1311687258	192.168.1.2	192.168.1.240	224	32030000796800020 0D20000040BFF040020C21311852892	...
15	1311687258	192.168.1.2	192.168.1.240	90	320300007A680002004C00000403FF0400F044218082220A	...
16	1311687258	192.168.1.240	192.168.1.2	84	32010007B68004A00000406120A10020020012C840005C01	...
17	1311687258	192.168.1.240	192.168.1.2	96	32010007C68005600000407120A1002000A012C84000BC01	...
18	1311687258	192.168.1.2	192.168.1.240	222	320300007B68000200D000000406FF040100000000000000	...
19	1311687258	192.168.1.2	192.168.1.240	158	320300007C68000200900000040 7FF040050000000000004	...
20	1311687258	192.168.1.2	192.168.1.240	236	320700000000000C00D60001120812020111000000000FF09	...

Data assembler

The data assembler component is responsible for analyzing the captured data, extracting the useful information and generating groups of bytes that will be used by the next component, the rule generator.

The data assembler component focuses on the payload of packets extracted by the feature extractor and it analyses all the packet payloads at the same time. Following are the steps to analyze data and create the groups; including an example.

1. Divide packets by their source and destination IP addresses and select those which have common source and destination IP addresses. An example is shown in Table 3.3.

Table 3.3: Example of the packet classification.

Packet number	**Payload**
1	32010007568004A00000406120A10020020012C840005C01 ...
2	3201000766800560000407120A1002000A012C84000BC01 ...
7	32010007768004A00000406120A10020020012C840005C01 ...

Packets with 192.168.1.240 as source IP address and 192.168.1.2 as destination IP address.

Packet number	**Payload**
3	320300007568000200D000000406FF040100000000000000 ...
4	3203000076680002009000000407FF040050000000000004 ...
5	320700000000000C00D60001120812020111000000000FF09 ...

Packets with 192.168.1.2 as source IP address and 192.168.1.240 as destination IP address.

2. Organize all packet payloads in rows and columns. Each row will contain individual packet payload. Each column will correspond to one byte of the payload. An example is shown in Table 3.4.

Table 3.4: Example of the organization of the payloads.

Packet	**Payload (Bytes)**													
number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	32	01	00	07	56	80	04	A0	00	00	40	61	20	...
2	32	01	00	07	66	80	05	60	00	00	40	71	20	...
7	32	01	00	07	76	80	04	A0	00	00	40	61	20	...

Packets with 192.168.1.240 as source IP address address and 192.168.1.2 as destination IP address.

Packet	**Payload (Bytes)**													
number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	32	03	00	00	75	68	00	02	00	D0	00	00	04	...
4	32	03	00	00	76	68	00	02	00	90	00	00	04	...
5	32	07	00	00	00	00	00	0C	00	D6	00	01	12	...

Packets with 192.168.1.2 as source IP address and 192.168.1.240 as destination IP address.

3. Starting from the first byte until the last one, indicate those bytes that change their value across different packets. An example is shown in Table 3.5.

Table 3.5: Example of the changing bits identification.

Packet	**Payload (Bytes)**													
number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	32	01	00	07	56	80	04	A0	00	00	40	61	20	...
2	32	01	00	07	66	80	05	60	00	00	40	71	20	...
7	32	01	00	07	76	80	04	A0	00	00	40	61	20	...

Packets with 192.168.1.240 as source IP address and 192.168.1.2 as destination IP address.

Packet	**Payload (Bytes)**													
number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	32	03	00	00	75	68	00	02	00	D0	00	00	04	...
4	32	03	00	00	76	68	00	02	00	90	00	00	04	...
5	32	07	00	00	00	00	00	0C	00	D6	00	01	12	...

Packets with 192.168.1.2 as source IP address and 192.168.1.240 as destination IP address.

4. Group consecutive changing bytes. Table 3.6 shows two groups of consecutive bytes.

Table 3.6: Example of the group of bytes.

Packet number	Payload (Bytes)													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	32	01	00	07	56	80	04	A0	00	00	40	61	20	...
2	32	01	00	07	66	80	05	60	00	00	40	71	20	...
7	32	01	00	07	76	80	04	A0	00	00	40	61	20	...

Packets with 192.168.1.240 as source IP address and 192.168.1.2 as destination IP address.

Packet number	Payload (Bytes)													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	32	03	00	00	75	68	00	02	00	D0	00	00	04	...
4	32	03	00	00	76	68	00	02	00	90	00	00	04	...
5	32	07	00	00	00	00	00	0C	00	D6	00	01	12	...

Packets with 192.168.1.2 as source IP address and 192.168.1.240 as destination IP address.

Group 1 Group 2 Group 3 Group 4 Group 5

5. Make note of each group starting byte number and its length in bytes. Table 3.7 shows the starting byte number and the length in bytes of each group.

Table 3.7: Example of the starting byte number and length of groups.

Group number	Starting bit number	Length (bits)
1	5	1
2	7	2
3	12	1

Packets with 192.168.1.240 as source IP address and 192.168.1.2 as destination IP address.

Group number	Starting bit number	Length (bits)
1	2	1
2	5	2
3	8	1
4	10	1
5	12	2

Packets with 192.168.1.2 as source IP address and 192.168.1.240 as destination IP address.

6. Note down the possible members of each group and the number of repetitions of group members. Table 3.8 shows an example of each group members and their cardinality.

Table 3.8: Example of members of the groups divided by packets source and destination.

Members of three groups of the packets which source IP address is 192.168.1.2 and 192.168.1.240 their destination IP.

Group member	Member cardinality
56	1
66	1
76	1

Group 1

Group member	Member cardinality
04 A0	2
05 60	1

Group 2

Group member	Member cardinality
61	2
71	1

Group 3

Members of three groups of the packets which source IP address is 192.168.1.240 and 192.168.1.2 their destination IP.

Group member	Member cardinality
03	2
07	1

Group 1

Group member	Member cardinality
75 68	1
76 68	1
00 00	1

Group 2

Group member	Member cardinality
02	2
0C	1

Group 3

Group member	Member cardinality
D0	1
90	1
D6	1

Group 4

Group member	Member cardinality
00 04	2
01 12	1

Group 5

Once all groups are created, their information is transferred to the next module, the rule generator.

Rule generator

The rule generator component is responsible for generating a set of rules that describes the possible groups and their features. It creates a behavioral pattern of the ICS network traffic. Note that each set of rules will describe the ICS network traffic of an individual industrial process, being necessary to create a new set of rules each time the process changes.

The language used to describe the rules is the Snort's rules description language [Roe13]. Simple, lightweight, flexible and powerful, the rules description language of Snort permits the use of Snort to check the content of the payload, enabling the possibility to check if the behavioral pattern is satisfied.

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header is formed by the following features:

- Rule action: what to do in the event that a packet with all the attributes indicated in the rule would show up.
- Protocol: protocol that the rule looks for.
- Source IP address: source IP address of the packet. It can be *any*.
- Source netmask address: source netmask address of the packet.
- Source port number: source port number of the packet. It can be *any*.
- The direction operator: indicates the orientation, or direction, of the traffic that the rule applies to.
- Destination IP address: destination IP address of the packet. It can be *any*.
- Destination netmask destination: source netmask address of the packet.
- Destination port number: destination port number of the packet. It can be *any*.

On the other hand, the rule option section, enclosed in parenthesis, contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken. The words before the colons in the rule options section are called option keywords. All of the elements within must be true for the indicated rule action to be taken. When specified in conjunction, the elements can be considered to form a logical AND statement. At the same time, the various rules in a Snort rules library file can be considered to form a large logical OR statement. In this context, even if there are 89 possible keywords, the rule generator module uses the following ones:

- Message: tells the logging and alerting engine the message to print along with a packet dump or to an alert.
- Packet payload size: used to test the packet payload size.
- Payload content: allows the user to set rules that search for specific content in the packet payload and trigger a response based on that data.

- Pattern starting byte: specifies where to start searching for a pattern within a packet.
- Pattern length: specifies how far within a packet Snort should search for the specified pattern.
- TCP flags: used to check if specific TCP flag bits are present.
- Rule identification number: used to uniquely identify Snort rules.
- Rule version number: used to uniquely identify revisions of Snort rules.

The rule generator takes as input the description of the groups and as output it generates a set of rules that describe the network traffic of an ICS. Therefore, following the example explained above, Table 3.9 and Table 3.10 show the set of rules generated for the description of the groups presented respectively by Table 3.7 and Table 3.8.

Table 3.9: Example of rules generated for the communication between the PLC and the SCADA server.

```
alert tcp 192.168.1.2 102 -> 192.168.1.240 any (msg:"plc_group1"; dsize:>8; \
content:!"|56|"; \
content:!"|66|"; \
content:!"|76|"; \
offset:8; depth:1; flags:PA; sid:1000001; rev:1;)

alert tcp 192.168.1.2 102 -> 192.168.1.240 any (msg:"plc_group2"; dsize:>10; \
content:!"|04A0|"; \
content:!"|0560|"; \
offset:10; depth:2; flags:PA; sid:1000002; rev:1;)

alert tcp 192.168.1.2 102 -> 192.168.1.240 any (msg:"plc_group3"; dsize:>15; \
content:!"|61|"; \
content:!"|71|"; \
offset:15; depth:1; flags:PA; sid:1000003; rev:1;)
```

As shown in Table 3.9, the communication between the PLC and SCADA server is composed of three rules, each one corresponding to an individual group. The first rule sets to be applied is to check if there is a packet that satisfies the following requirements:

- Protocol: TCP

- Source IP address: 192.168.1.2 (the PLC)
- Source port number: 102
- Destination IP address: 192.168.1.240 (the SCADA Server)
- Destination port number: any
- Payload size: bigger than eight bytes
- TCP flags: PUSH and ACK set
- Eighth byte's content does not correspond to 0x56, 0x66 or 0x76 values.

In the case that an alert is raised, it sets the alert message to *plc_group1*, providing an easy way to identify the alerting rule. The second and third rules are quite similar to the first one except for the values; the minimum payload size and the position of inspected bytes. Thus, the second rule looks for the tenth to twelfth bytes positions to check if the payload content matches to 0x04A0 or 0x0560 values. Otherwise an alert with *plc_group2* message is raised. For the third rule, the changes are the message, inspected content, its minimum size and its position. Note that the three rules have a different *sid* or rule identification number. This is due to the fact that it is not possible to have different rules with the same rule identification number.

Table 3.10 shows the set of rules that correspond to the communication between the SCADA server and the PLC. In this case, each rule inspects the packets that have the IP address of the SCADA server as source and the IP address of the PLC as destination together with 102 as destination port number. The following keywords change their value among each rule:

1. Message
2. Content
3. Position or offset
4. Number of inspected bytes or depth
5. Payload's minimum size or dsize
6. Rule identification number

Table 3.10: Example of rules generated for the communication between the server and PLC.

```
alert tcp 192.168.1.240 any -> 192.168.1.2 102 (msg:"server_group1"; dsize:>5; \
content:!"|03|"; \
content:!"|07|"; \
offset:5; depth:1; flags:PA; sid:1000004; rev:1;)

alert tcp 192.168.1.240 any -> 192.168.1.2 102 (msg:"server_group2"; dsize:>8; \
content:!"|7568|"; \
content:!"|7668|"; \
content:!"|0000|"; \
offset:8; depth:2; flags:PA; sid:1000005; rev:1;)

alert tcp 192.168.1.240 any -> 192.168.1.2 102 (msg:"server_group3"; dsize:>11; \
content:!"|02|"; \
content:!"|0C|"; \
offset:11; depth:1; flags:PA; sid:1000006; rev:1;)

alert tcp 192.168.1.240 any -> 192.168.1.2 102 (msg:"server_group4"; dsize:>13; \
content:!"|D0|"; \
content:!"|90|"; \
content:!"|D6|"; \
offset:13; depth:1; flags:PA; sid:1000007; rev:1;)

alert tcp 192.168.1.240 any -> 192.168.1.2 102 (msg:"server_group5"; dsize:>15; \
content:!"|0004|"; \
content:!"|0112|"; \
offset:15; depth:2; flags:PA; sid:1000008; rev:1;)
```

The rule set generated by the rule generator component is stored in an individual file which will be the input of the next module, the ADS.

Anomaly Detection System

The goal of an Anomaly Detection System is to compare the current behavior patterns with a behavioral model of normality, classifying as anomaly every situation that is out of the usual behavior. A misuse based Intrusion Detection System instead, based on rules that describe intrusion patterns, tries to search for uncommon situations, which are classified as intrusions. Therefore, a misuse based IDS searches for patterns that match those of an attack, while an ADS looks for traffic behavior that stands out from

common traffic.

The presented module uses a modified version of Snort [Sou13b], a well known IDS. Snort, based on libpcap [Tcp13a], is a lightweight and open source IDS, composed of three modules: a packet decoder, a detection engine and an alert and register subsystem. Although Snort comes with a default set of rules, the modified version of Snort is loaded with the set of rules generated by the previous component, the rule generator component.

Snort has some limitations that prevents us from using it as an ADS. First, a limitation on the maximum number of rules, and second, the impossibility to use the *Content* keyword (see below) to analyze the same position are the main limitations.

Snort, at the initialization process, reads the configuration and rule files and loads all the rules generating the necessary rule in memory. Rule chains are the binary translation of the rules, loaded in memory and used in real time for pattern matching. In order to avoid getting blocked while reading and translating malformed rules or keeping without memory due to endless rules, Snort defines a maximum rule length. However, some of the rules generated by the rule generator component are longer than the predefined maximum rule length. Therefore, in order to use Snort as an ADS, it has been modified by removing the maximum rule length limitation. This modification enables Snort to load in memory the set of rules for normal behavior generated by the corresponding module.

The rule description language of Snort defines a set of keywords such as *content*. Some of the keywords can be used more than once in each rule, enabling to write complex rules and in the same way preventing the usage of multiple rules when they can be described using one. All the rules shown in Tables 3.9 and 3.10 have a keyword in common, the *content* keyword. Looking at each rule it can be observed that all the *content* keywords are focused on the same byte or group of bytes of the payload. In the proposed model, there are multiple options of the content of the same byte or group of bytes. By default, Snort does not check the same byte multiple times, even if the *content* keyword can be used more than once. Therefore, Snort has been modified by enabling to check the same byte or group of bytes.

Table 3.11: Example of rules with intercalated logical ANDs and ORs.

```
alert tcp 192.168.1.2 102 -> 192.168.1.240 any (msg:"plc_group1"; AND dsize:>8; \
AND content:"|56|"; \
AND content:"|66|"; \
AND content:"|76|"; \
AND offset:8; AND depth:1; AND flags:PA; AND sid:1000001; AND rev:1;)

OR

alert tcp 192.168.1.2 102 -> 192.168.1.240 any (msg:"plc_group2"; AND dsize:>10; \
AND content:"|04A0|"; \
AND content:"|0560|"; \
AND offset:10; AND depth:2; AND flags:PA; AND sid:1000002; AND rev:1;)
```

As shown in Table 3.11, the rule option section, enclosed in parenthesis, contains the alert message as well as the information about which parts of the packet should be inspected to determine if the rule action should be taken. By default, all the elements of the rule option section can be considered to form a logical AND statement. In the same way, consecutive Snort rules can be considered to form a logical OR statement. The problem comes when the intention is to use Snort to check if a predetermined payload position contains an specific value from multiple options. Obviously, if none of the possible options is satisfied, the global answer of the rule will be false, generating a false negative.

Table 3.12: Example of rules with intercalated logical ANDs.

```
alert tcp 192.168.1.2 102 -> 192.168.1.240 any (msg:"plc_group1"; AND dsize:>8; \
AND content:!"|56|"; \
AND content:!"|66|"; \
AND content:!"|76|"; \
AND offset:8; AND depth:1; AND flags:PA; AND sid:1000001; AND rev:1;)
```

The solution for the previous problem comes from negating all the content possibilities. As shown in Table 3.12, by converting all the content options in negative, Snort must set all the options to true in order to raise an alert. Hence, all the rules generated by the rule generator module have the content option negate. In consequence, first normal behavior of the ICS network is translated to Snort rules. The rule generator transforms

those into an unlimited set of concatenated, multi-content and opposite-meaning rules in order to raise an alert when the traffic does not contain such normal behavior patterns.

Notification centre

The notificatoin center module is a graphical user interface which notifies system users about the anomalies encountered by Snort. This system uses Basic Analysis and Security Engine (BASE) [Flo10] as a notification engine. In order to analyse the alarms raised by Snort, the module offers a simple and intuitive web interface.

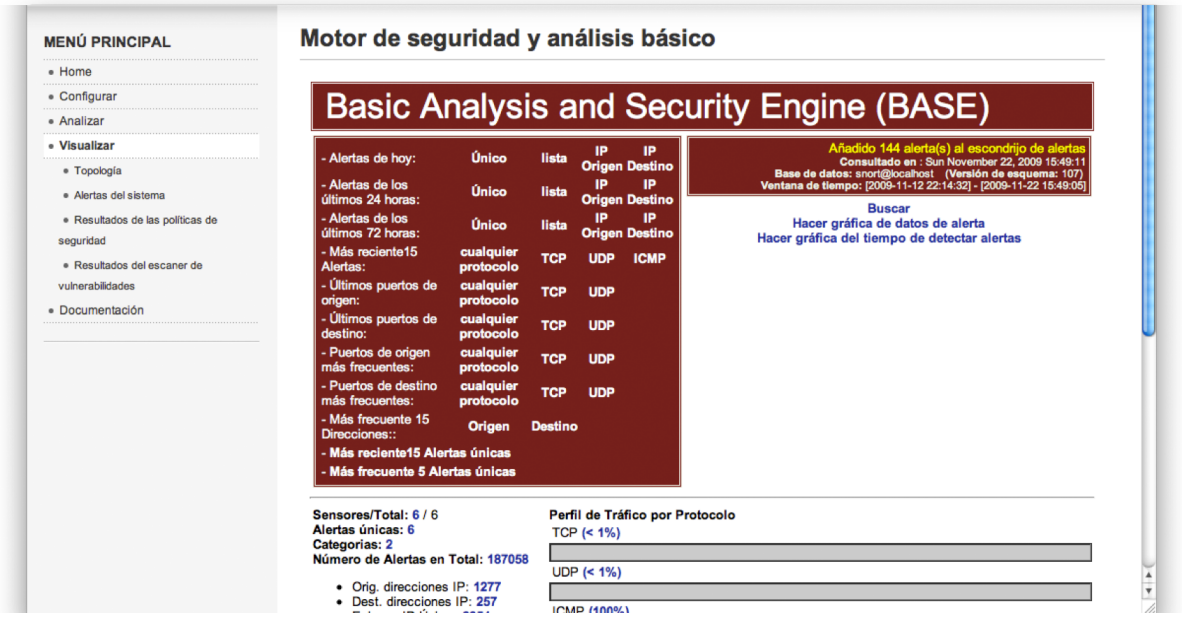


Figure 3.4: Graphical user interface of BASE.

Figure 3.4 shows the graphical user interface of BASE. As shown, BASE classifies the traffic based on protocol, source and destination IP addresses and it shows some statistics related with raised alerts by Snort. It organizes different kinds of alerts offering the user an easy and intuitive way to access the information.

3.2.2 Implementation

The Anomaly Detection System for ICS binary private protocols is composed of six modules: the network traffic capturer, the feature extractor, the data assembler, the

rule generator, the anomaly detection system and finally, the notification centre. Some of the modules or components are modified versions of well known open source software, such as tcpdump [Tcp13b], Snort [Sou13b] or BASE [Flo10].

The network traffic capturer module is based on the tcpdump open source software. Tcpdump, as well as Snort, works together with libpcap library [Tcp13a]. In fact, tcpdump and libpcap have been developed by the same team. Libpcap, is a portable library written in C/C++ for network traffic capturing, while tcpdump is a powerful command-line packet analyzer. The network traffic capturer component uses both of them, libpcap in order to capture packets that go through the network, and tcpdump to filter and store them in a file. Packets can be filtered at the reception time as well as later on, during the packet processing step. In the case of the network traffic component, it filters the packets at the beginning, at the reception time, avoiding any processing time demanded by unnecessary packets. Due to the fact that the traffic we are interested in corresponds to TCP protocol and 102 port number, tcpdump is loaded with a preconfigured filter following Berkeley Packet Filter (BPF) style [McC91; McC93]. Table 3.13 shows an example of the filter.

Table 3.13: Example of BPF rule used by tcpdump.

tcp port 102

The feature extractor module has been developed using Python [Pyt13] programming language and specifically Scapy interactive packet manipulation program [Bio11]. Scapy offers a wide range of possibilities including the reading of packets from a file, manipulation of them and saving them in another type of file such as Comma Separated Values (CSV) file. The extracted features are the packet number, timestamp, source and destination IP addresses, payload length in bytes and the payload itself.

The next module, the data assembler module, has been developed using MATrix LABoratory (MATLAB) [The13]. MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. It allows the user to analyze data, develop algorithms, and create models and applications. The language, tools, and built-in math functions enable the user to explore multiple approaches and reach a solution faster than using spreadsheets or traditional programming languages, such as C/C++ or Java. The data assembler module, entirely developed in MATLAB,

reads a file in CSV format, analyses data, generates the corresponding group and finally saves all groups' details in another CSV file.

The rule generator component, as well as the feature extractor module, has been entirely developed in MATLAB. At the beginning it reads a CSV file from which it takes all the relative information about groups. This information is used later on to write the corresponding rules in Snort rule language format. As an output, the module generates an ASCII file containing all generated rules each one followed by the others.

Snort, written in the C programming language, is the base system of the anomaly detection system module. In fact, the developed anomaly detection system is a modification of Snort. The modifications done enable us to use Snort as an ADS. As is explained on Subsection 3.2.1, the maximum rule length and the impossibility of using the *Content* keyword to analyze the same bytes' position have to be overcome. The solution comes from changing the source code of Snort and recompiling it generating a personalized version of the well known application. Therefore, the implementation of the anomaly detection system module has been done in C language.

Finally, the notification centre, as explained in Subsection 3.2.1, is an adaptation of BASE [Flo10] software. In this case, the development of the module has not altered the source code of BASE, but only included as a part of an integrated solution.

3.2.3 Validation

The validation of the system presented has been done by using network traffic of a real ICS. The control system that has been used controls a steel rolling process. Even if it is not a Critical Infrastructure, the steel rolling process is quite critical due to the temperatures, the necessary amount of electrical power and the weight of the materials that are involved. Moreover, a critical infrastructure such as those used for transport, energy or grids, etc. use the same type of components as this process. A successful attack against this kind of process could be the origin of many disasters such as considerable economical losses, environmental disasters, including the loss of human lives. The PCSs used in the installation are manufactured by Siemens. In fact, the PLCs and network switches used correspond respectively to Siemens S7 and Siemens SCALANCE families.

Figure 3.1 shows the network diagram where traffic captures were carried out. As shown,

the host used to capture network data, the SCADA server, is part of a switched network, thus, the capturing node is able to capture only traffic corresponding to broadcast, multicast and traffic originated from and to the node.

A validation conducted using a real ICS is more realistic than employing traffic generated in a simulated environment. Many times the reality is far away from simulations, even if the models used in both cases are practically the same. ICSs are critical processes which must ensure their availability in front of diverse threats such as software patches. Thus, patches are tested before being deployed. In addition, there is an established thought that it is better not to touch a control process device if it is already working.

The network traffic captures used for the validation purposes have been captured in two different network segments, in order to check the system response with different traffic. Both traffic captures were performed in attack-free conditions; there were no anomalies before, during or after the capturing process. In total, two network traffic captures have been used for system training and validation. As shown in Table 3.14, the first and second captures are two hours and a half and two hours long respectively, and they contain 309,830 and 329,742 packets. Filtering out all packets not corresponding to Siemens S7 protocol, the first capture contains 99,808 valid or useful packets while the second capture consist of 138,855 filtered packets.

Table 3.14: Captured data files for system training and validation.

Capture number	**Duration (seconds)**	**Total number of packets**	**Useful number of packets**
1	9,013	309,830	99,808
2	7,198	329,742	138,855

The difference of capturing time between both captured files is contradictory compared with the number of captured packets. The first capture file, even if it is longer in time, consists of less packets than the second one, shorter in time. This divergence is related with the network segment where the traffic has been captured. Even if both packet acquisitions have been made in the same ICS, the communication between different devices could require more or less packets.

The presented system is divided in six modules or components that are at the same way organized into two operational modes, offline and online. The objective of the offline mode is to generate the behavioral pattern of the ICS network traffic. To do so, it

captures network traffic packets and it uses them to extract knowledge that later on will be converted to rules describing normal operations. The online mode uses the generated rules to compare with the traffic model of the actual situation, raising alerts in case the anomaly detection system detects an anomaly. Therefore, a piece of each network traffic capture file has been used to train the system and generate the rules, while the entire capture file has been used to check if the anomaly detection systems detects a deviation from the normality.

The 75% of each capture has been used to train the system and generate the corresponding rules. The obtained results differ significantly depending on the network segment where the traffic capture has been performed. As shown in Table 3.15, in the first case 74,856 packets have been used to create groups, giving a total number of 52 while the minimum length of a group is one byte and the maximum 71 bytes. In the second case, a total of 104,141 packets have been used, generating six groups of a minimum length of one byte and a maximum of 227 bytes. As shown, the difference between the generated number of groups is high. Anyhow this does not mean that behavioral models are inappropriate. Due to the fact that each generated group equates to a rule, in the first case a total of 52 rules have been created while in the second case we obtained 6.

Table 3.15: Characteristics of generated behavioral patterns.

Capture number	**Training num. of packets**	**Number of groups**	**Minimum length (bytes)**	**Maximum length (bytes)**
1	74,856	52	1	71
2	104,141	6	1	227

After the rules of each traffic capture have been generated, each capture file has been tested with the corresponding set of rules. To do so, using the network configuration showed in Figure 3.5, the ADS module has been loaded with the rules, and using Tcpreplay software [Tur13], the traffic capture files have been replayed. The ADS module analyzed all the traffic and every time it detected an anomaly, it raised an alert to the notification centre component.

Table 3.16 shows the obtained results. As it can be observed, the number of raised alerts or false positives are 49 for the first capture and 173 for the second. If the number of false positives is compared with the number of packets not used for system training purposes, the results are 0.2% for the first capture file and 0.5% for the second. These

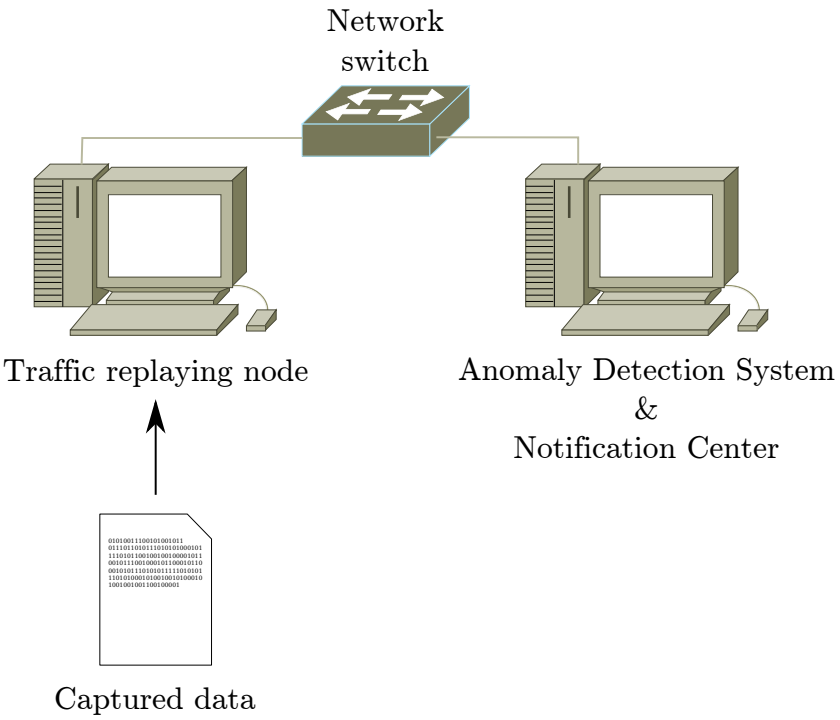


Figure 3.5: Validation network diagram.

numbers are relatively small, which indicate that the created set of rules have a high accuracy. Obviously the rate decreases if the number of false positives are compared with the total number of packets. In that case the obtained results are 0.016% for the first capture and 0.052% for the second.

Table 3.16: Results of anomaly detection system component.

	Capture number	
	1	2
False Positive packets	49	173
F.P. vs untrained packets	0.2%	0.5%
F.P. vs all packets	0.016%	0.052%
False Positive rules	7	2
F.P. vs all rules	13.46%	33.33%

Another interesting result is the number of different rules that raised an alert. In the case of the first capture, from the 52 rules created by the rule generator module, just seven different rules raised an alert. This means that the 49 packets corresponding to false positives are concentrated in these seven rules. Thus, the system created 45 rules that do not raise any alert, showing that they are correctly designed. For the second capture

just two different rules raised the alerts, compared with the total number or generated rules, it can be observed that four rules have included all the possible members. If the number of activated rules is compared with the total number of rules, the results for the first case indicate 13.46% of the rules raised an alert, while in the second case the proportion grows to 33.33%.

Several network protocols contain a packet identification number that grows each time a packet is sent. Usually the identification number has an upper limit imposed by the number of bytes dedicated to represent it. When the number of sent packets reaches the upper limit, it is returned to zero, beginning again the loop. The presented system is based on the network traffic capture time that the security engineer decides to use to create the behavioral pattern. Even if used time is equivalent to a unique or various process times, it could be that the number of sent messages do not reach the packet identification upper limit. Therefore, the behavioral pattern will not reflect all the possibilities, leaving a rule with a high probability to raise false positives.

3.3 Conclusions of the chapter

Proprietary binary protocols complicate the creation of ADSs as well as their understanding. Despite this, the payload of binary protocol packets can be treated as raw data and with the usage of different techniques try to extract some knowledge. This chapter presents a payload-based ADS for binary protocols based on ICSs repetitiveness. The entire process is described together with the implemented system.

The presented system is composed of six components that together discompose the packet payloads and extract a set of rules that later on are used to detect attack patterns. The described methodology is validated through two different real ICS traffic captures. The obtained results show its performance in terms of false positive rate.

The main advantage of the proposed approach is that is capable of working with different kinds of protocols, thus, diversifying its applicability.

Chapter 4

A method to construct ICS network traffic models for public protocols

Anomaly Detection Systems require behavioral models to detect anomalies, thus, the first step is to create those models. The vast majority of ADSs require attack-free training data to create behavioral models; chapter 3 describes a training-data dependent ADS. The generation of a behavioral model based on training-data, whether attack-free or not, requires the preparation of data, which in fact consumes time and resources.

Many ICSs control processes with a high availability level, must be running 24 hours, 365 days per year. Any process interruption could cause significant economical losses, therefore, their operation is critical. Multiple reasons could cause a process interruption, hence, the shorter the downtime, the smaller the possible effects.

Every time a process is changed or an old or malfunctioning device is replaced with a new one, the behavioral model of the network traffic changes, forcing the creation of a new pattern. A training-data dependent ADS will require a lot of time and resources to prepare a good sample of data; the economical costs could be avoided by not creating this data but sacrificing the quality of the ADS and compromising the security of the ICS.

In this chapter, a method to generate realistic network traffic in laboratory conditions without the need for a real ICS installation is introduced. The presented method defines an algorithm, which eliminates the necessity of training-data during the behavioral model creation process. Such a method could support experimentation through the recreation of realistic traffic in simulated environments. The defined algorithm uses ICS application characteristics as input and generates a description of the network traffic. The resulting traffic could be used for multiple applications, such as behavioral pattern

generation and network planning process.

4.1 ICS operation overview

As already mentioned in chapter 2, Industrial Control System is a general term used to refer to several types of control systems, such as SCADA systems, DCSs and PLCs. SCADA systems are used to control geographically dispersed assets, where the system control is usually done from a centralized location. DCSs are used to control system that are concentrated in a smaller area than SCADA systems, such as water and wastewater treatment, oil refineries and so forth. PLCs, designed to control industrial equipment and processes, are extensively used in almost all industrial processes. Moreover, even if PLCs are the principal components of DCSs and SCADA systems, usually, in small control system configurations, they are the primary components used to provide operational control of discrete processes.

The operation of ICSs can be divided into two abstraction levels. The lower level is the one which controls the physical actions and where they take place, while the operators or control process maintainers are located in a higher level of abstraction. As shown by figure 4.1, at the lower level we can find PLCs or RTUs as well as operation control loops. At the higher level resides Human Machine Interfaces (HMIs) together with Control servers and SCADA servers or Master Terminal Units (MTUs).

In order to clarify some terms, these are the descriptions of the most used devices in ICSs, ordered from the lower to the higher abstraction layer:

- The control loops, the basis of ICS operations, consist of sensors, actuators, controller and communication hardware. The sensors transmit information about the actual state of controlled variables to the controllers. The controllers in turn, interpret or understand the received information and based on the local strategy, established by the ICS application, send to the actuators the information about the actions to perform. The actuators receive the information and they execute the corresponding action while the sensors begin again with the control loop by sending the new state of the controlled variables to the controllers or PLCs.
- The PLCs are small industrial computers, used substantially in SCADA systems

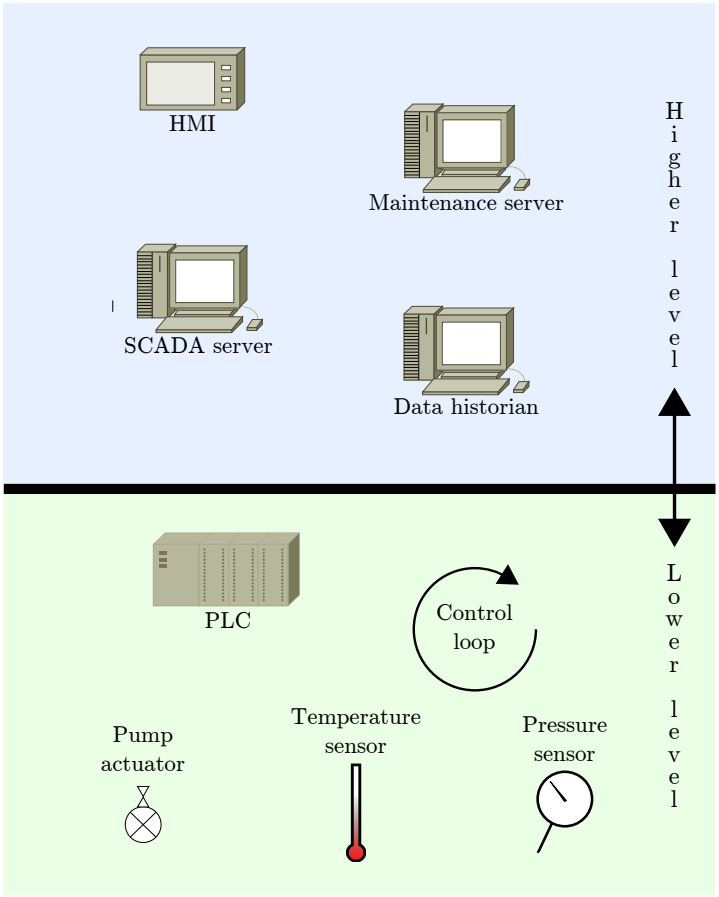


Figure 4.1: Components of each abstraction level.

and DCSs. In SCADA environments, PLCs are often used as field devices because they are more economical, versatile, flexible, and configurable than special-purpose RTUs.

- The RTUs are special purpose data acquisition and control units designed to support SCADA remote stations; they provide the same control as PLCs but are designed for specific control applications. Sometimes PLCs are implemented as field devices to serve as RTUs; in this case, the PLC is often referred to as an RTU.

- The SCADA server, also referred as MTU, is the master device in SCADA systems, where PLCs and RTUs act as slaves.

- The Control Server is the device which hosts the supervisory application of the PLC. To communicate with subordinate devices, it uses the ICS network.

- The HMI, as its name suggests, is the interaction point between process operators, engineers and the process control application. It displays process status information and historical information, and it permits monitoring and configuring a set of points, control algorithms, and adjust and establish parameters in the controller.
- The Data historian is a centralized database for logging all process information within an ICS. Information stored in this database can be accessed to support various analyses, from statistical process control to enterprise level planning.
- The maintenance server is a node for maintainance of the ICS after its implementation. Usually, the maintenance server is located in another network, allowing an external operator to perform remote maintenance avoiding physical presence and the its costs.

The algorithm presented in this chapter is focused in the ICS network traffic between the Control Server or SCADA server and the PLC or RTU. Therefore, the communication between field devices and PLC or RTU is out of the scope of the presented algorithm. In addition, the communication with the HMI is also out of interest.

ICSs are characterized by being time-critical systems. One of their requirements is to give a result in a deterministic time. The acceptable delay and jitter are dictated by the individual installations, established as well by the process characteristics. In addition, PLCs are small computers with limited computation resources able to accomplish only the task they were designed for. The installation of additional software, such as an antivirus or a host based IDS, is not feasible due to the lack of resources. Control and SCADA servers instead, are usually hosted in computers with enough resources and standard operating systems. Thus, PLCs act as slaves, leaving Control or SCADA servers to behave as masters. The variable values represent physical states as well as intermediate calculus results. The PLCs, which carry out the corresponding calculus operations, have real and updated values. Control and SCADA servers in contrast, need to update their variable values by querying the PLCs. The update of the values is performed across the network. The communication between Control or SCADA servers and PLCs follow a *pull* strategy, that is, PLCs act as servers waiting for incoming connections while Control or SCADA servers ask about the status of the variables they need to be updated. The *pull* strategy permits PLCs to focus on the control of the processes; they do not have to control the update rates of variables. In case of *push*

strategy, PLCs will have to send variable value updates to Control or SCADA servers in a predefined amount of time, in the update rate of each variable. This strategy demands more PLC resources than the *pull* strategy.

Following the *pulling* strategy, it is important to differentiate the two types of update rates used by ICSs: system update rates and variable update rates, both of them set in seconds. Control and SCADA servers define a set of system update rates, lets call them U_s. Each system defines a different set of system update rates. In the same way, each set can be composed of a variable number of system update rates. A set of system update rates defines the time intervals in which the system will ask the PLC or the RTU for the values of variables. Thus, the originated network traffic will be directly associated to the set of system update rates. The applications of ICSs are composed, among other things, by different variables. The engineer, or the application developer, defines the variables of the application and the update rate of each variable, lets call U_v. The variable update rate establishes the time interval in which each variable needs to be updated. Even if variable update rates can be set freely, without considering system update rates, due to the fact that system update rates take precedence over variable update rates, the last ones have to be fitted into the system update rates. This, makes the calculations easier and it reduces the generated network traffic by simplifying the number of necessary requests. Therefore, even if there are multiple variables, each one with a different variable update rate, all of them will be fitted into the set of system update rates, resulting in a smaller combination of possibilities. This, eases the necessary calculations as well as simplifies the resulting behavioral model.

4.2 Industrial Control System Applications

Industrial Control Systems are controlled by specially made (personalized) applications for each industrial installation. An application is a combination of logic and values. The logic is represented as value dependent conditional actions that the controller should execute. The values in turn, are stored on variables, the storage location and an associated symbolic name. These applications, together with the controlled industrial systems, are unique. Even if the purpose of the application is the same, depending on the programmer, it could have a different logic and it would use distinct values, giving a large

number of possibilities.

ICS application values can represent different types of physical states, a value could represent the state of a switch while another value could give the pressure of a tank. Thus, the variables can be classified in different ways depending on the nature of the value they store. In the same way, variables can be used to store logical values, values that are not a physical state representation. These variables make the logic of an application easier. Hence, the total number of variables could be bigger than the number of interesting variables from the point of view of the operator or SCADA server. Depending on the nature of the value stored by a variable or how frequently it is changed by the application logic, it would be necessary to adjust the variable update rate to follow its value.

Although the number of possible applications is enormous, the traffic that results from these can be described by the following three components: variable classes, number of variables and variable update rates. These components are defined by ICS application designers and implemented by developers. The components are part of ICS application specifications.

- **Variable classes:** refers to the kinds of variable that the system or the application development tools are able to manage, e.g. `boolean`, `integer`, `real` and `string`. The application programmer should select the appropriate one for each purpose, e.g. store the status of a switch in a `boolean` variable, store the temperature value in a `real` variable. Depending on the class, variables are allocated a specific number of bytes. Thus, the assigned memory will depend on variable classes.

- **Number of variables:** from the operator's point of view, in order for the HMI to show the status of updated variable values, the SCADA server needs to update them. Due to the fact that variables consume memory and process resources, application developers try to use the minimum number of variables possible in order to save resources. Variables can be connected to controller ports, where the transformation between physical and cyber values is done. Even if an ICS application consist of more variables than just those that are updated by the SCADA server, from the generated network traffic point of view, only the updated ones are interesting. Hence, the number of variables refers to the number of updated variables across the network.

- **Variable update rates:** is the time rate at which variable values are refreshed. Depending on their purpose, as defined by engineers, update rates can be the same for all variables, or they can be defined per variable class or even per variable. On the other hand, application developers should use the maximum Update Rate (UR) possible in order to save resources. For example, in the case of a variable designed to store the time in seconds, it would be enough to update it once per second. If the value is meaningful in milliseconds, it should be updated every millisecond. However, this would also increase the network traffic, and calculation times in both end-nodes, i.e. SCADA server and PLC.

In ICSs, following the *pulling* strategy, SCADA servers interrogate PLCs using SCADA communications protocols in order to refresh the value of local variables. After all, SCADA servers hold the data that would be presented in a human readable form to operators behind HMI devices. Even if the purpose of two applications is totally different, the number of variables, variable classes and update rates could be the same, and would generate an identical communication pattern.

SCADA communications protocols such as Modbus/TCP, and DNP3 could be object oriented or variable oriented protocols. During this project only the second type was analyzed. In the case of variable oriented protocols, servers keep the memory address in which variables are stored. In order to preserve the network bandwidth and system's resources, SCADA servers ask for several variables at the same time; variables fitted into the same system update rate will be grouped in order to form a unique value status request. Thus, for each system update rate, a single request that encompass multiple variables will be generated instead of multiple request packets. This means that the requested packet sizes are proportional to the number of requested variables.

4.3 Methodology

The goal of the proposed algorithm is to generate a network traffic model starting with a real ICS application description. It focuses on the communication between the SCADA server and PLCs with applications in the field of IDS. Before the actual construction of the algorithm we conducted a thorough analysis of real traffic between SCADA servers and PLCs from ABB [Abb13]. The analyzed network traffic was captured in the

Experimental Platform for Internet Contingencies (EPIC), the emulation testbed of the Institute for the Protection and Security of the Citizen (IPSC), Joint Research Centre (JRC) of the European Commission. The EPIC testbed is based on the Emulab software [The12], that is able to recreate a wide range of experimentation environments in which researchers can develop, debug and evaluate complex systems [Whi02]. The testbed characteristics have been studied in several works [Sia12; Gen12; Pér10].

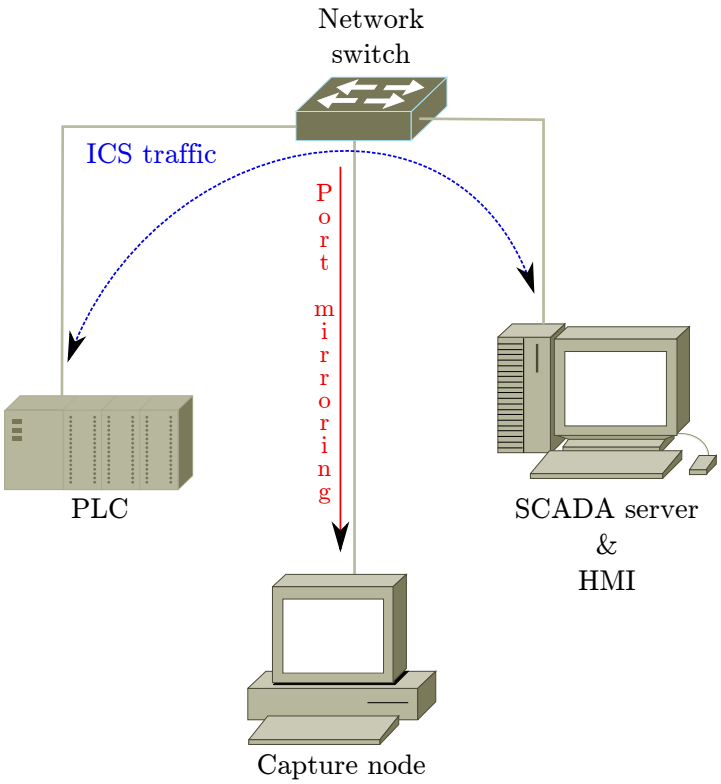


Figure 4.2: Experimental network topology

By using the EPIC platform, we constructed the topology shown in Figure 4.2. This topology included a real PLC, a SCADA server together with HMI software, a network switch and a dedicated node to capture network traffic. All the devices were connected to the same commuted LAN. The switch was configured with the port mirroring feature turned ON in order to capture all the traffic between the SCADA server and the PLC.

The PLC we used was ABB's 800M with the Manufacturing Message Specification (MMS) protocol. As shown in Figure 4.3, the MMS protocol is located above other protocols such as the Connection-Oriented Transport Protocol (COTP) and ISO Transport Services on top of the TCP (TPKT).

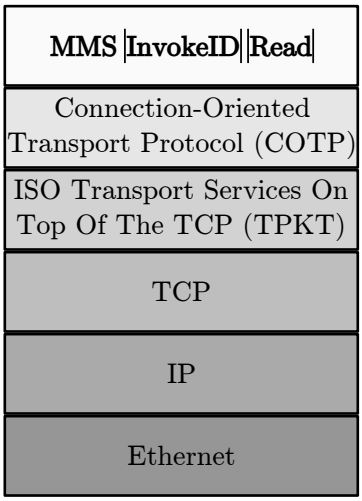


Figure 4.3: MMS protocol layers used by SCADA servers and PLCs from ABB

The communication between the SCADA server and PLC begins with a setup or initialization process. Then, it continues with a loop that issues requests at each update rate in order to refresh the variable values. At this point we assume that the initialization process can be different for each communication protocol and device, and specific to each vendor. Our work focuses on the second part of the communication in which variables are updated. Figure 4.4 shows in detail the initialization process and the loop we are focusing on, i.e. *Repeat until Process duration*.

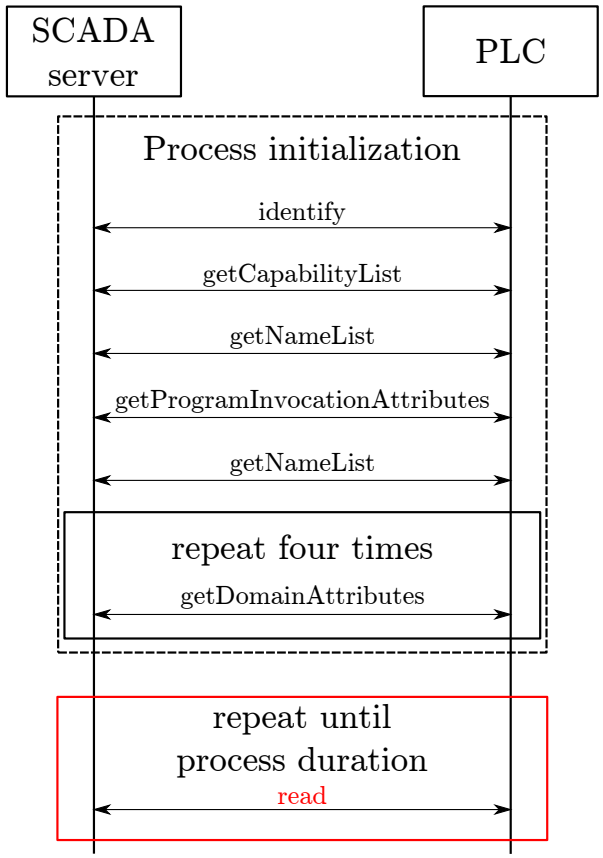


Figure 4.4: The sequence of messages exchanged by the SCADA server and PLC

ICS applications composed of different numbers of variables, variable classes and variable update rates generate distinct network traffic patterns. Thus, in order to analyze the output of each performed change, several ICS applications have been used to generate diverse network traffic patterns. Each application ran for 20 minutes and the generated traffic was captured in separate files using Tcpdump [Tcp13b]. The analyzed information was extracted from captured files using Scapy [Bio11]. Each capture was analyzed in terms of:

- **Number of packets:** number of packets captured.
- **Number of different requests:** number of different SCADA server requests sent to the PLC.
- **Size of each request (bytes):** size in bytes of each request, including all layers.
- **Packet inter-arrival time (seconds):** average time in seconds between packets

arrival time.

- **Packet inter-arrival time for each request (seconds):** average time in seconds between packets arrival time for each request type.
- **Size of each packet based on variable classes (bytes):** packet size by taking into account the number of variables and their specifications (class, update rate).
- **Number of different requests based on variable classes:** number of different requests due to the fact that several variable classes could be combined in the same request.
- **Number of different requests based on update rates:** number of different requests due to the fact that several variable update rates could be combined in the same system update rates.

Table 4.1: Description of applications used for algorithm construction

Size	**Number of variables**	**Variable classes**	U_v [1]	U_s [2]	**Max. No. variables per U_s [1]**	**Number of applications**
Small	1-6	1-4	1-13	6	1	7
Medium	7-60	1-4	1-12	6	10	4
Large	61-120	1-4	1-6	6	20	8
Total						**19**

[1] U_v = Variable update rate.
[2] U_s = System update rate.

Table 4.1 summarizes the number of applications and their specifications. As shown, three types of applications were used, depending on the number of variables, classified as small, medium or large. The small ones, composed of up to 6 variables, were used to analyze the effect of different variable classes and variable update rates into the network traffic. By combining distinct variable classes in the same variable update rate, we could see that distinct variable classes are not joined in a single packet. In the same way, using multiple variable update rates we could see that all variable update rates are fitted into system update rates. The medium size applications, with up to 60 variables, were used to analyze the impact of multiple variables of the same variable class and distinct variable update rate into the generated traffic. Even if different variable update

rates were used, they were chosen in order to see how the system fit them into system update rates. Finally, large applications, with up to 120 variables, were used to analyze the effect of numerous variables into network packets. In total, 19 application were created and analyzed. The analysis of these features for all the analyzed applications was used to construct the proposed algorithm.

4.4 Generating Traffic Models

The proposed algorithm generates the traffic model based on application specifications instead of statistical traffic analysis. As explained in section 4.2, each application can be described in terms of number of variables, class of variables and variable update rates. These specifications are application dependent and are used to calculate the communication pattern between the SCADA server and the PLC. Although the analysis we made is specific to the MMS protocol, it can be adapted to other protocols as well, e.g. Modbus.

The algorithm uses the "generic" Gaussian distribution in order to illustrate the applicability of the approach. However, users can choose other distributions as well, e.g. Poisson, in order to best fit their specific network patterns.

The proposed algorithm takes the following sets and variables as input:

- V = set of variables
- C = set of variable classes
- U_v = set of variable update rates
- U_s = set of system update rates
- d = duration of the trace (in seconds)
- b = network bandwidth

Each variable of the set of variables, V, is a pair consisting of variable class and variable update rate, denoted by (c, u_v). In the same way, the set of variable classes, C, consist of elements where each one is a pair of variable class and length in bytes, denoted by

(c, l), e.g. $(\texttt{integer}, 2)$.

As output, the algorithm generates a set of packet descriptions P, where each packet is defined as a pair including the transmission time and the size of the packet, denoted by (θ, σ).

For simplicity, throughout the chapter we use the X_y^z notation to denote the y component of the z-th element in set X. For instance, taking the C set, the j-th element is denoted by C^j, and the c component of the j-th element is denoted by C_c^j.

The algorithm starts with the *MAIN* function by initializing the set of packet descriptions P and $numP$, where the later one is used to count the number of packets. Next, it loops through all system update rates U_s^j and variable classes C_c^k and calls the $Packet(j, C_c^k)$ function to get the number of variables that should be placed in a packet. Then, based on the returned value and the size of the variable class C_l^k, it calculates the packet size σ at line #20.

As users can explicitly specify the length of the trace, i.e. through the d variable, we calculate the number of transmissions by dividing d to each update rate U_s^j. For each transmission we increase $numP$ and we generate a random number r using a Gaussian distribution function. The role of this random number is to introduce a realistic deviation from the configured update rate. Such deviations are caused by multi-process OSs, delays in network communications, etc., and are frequent in real environments. Based on this value we calculate the predicted transmission time θ and we add a new packet description to P at line #25.

Starting with line #26 we sort the P set in a time order and we change the transmission time of packets in order to take into account the bandwidth of the system. More specifically, we calculate the time t needed to send each packet P^{j-1} by dividing the size of the packet P_σ^{j-1} by the bandwidth b. Then, if the time needed to send the previous packet exceeds the actual packet transmission time, we update the transmission time of the current packet, as shown in line #31.

Finally, we briefly describe the $Packet(i, c)$ function, used to count the number of variables sent out in the i-th system update rate. This function returns the number of variables of class c for which the update rate satisfies one of the next three conditions. The *first condition* given at line #7 is a particular case and counts all variables for

Algorithm 1 Traffic model generator

1: $input :< U_v, C, V, U_s, d, b >$, $output :< P >$
2: /* *The Packet function counts all variables of class C that satisfy a set of conditions. This number is used by the MAIN function to calculate the size of the packet.* */
3: **function** $Packet(i, c)$
4: $X := 0$
5: **for** $(j := 1$ to $|V|)$ **do**
6: **if** $(V_c^j = c)$ **then**
7: **if** $((i = 1)$ AND $(V_u^j < U_s^2))$ **then**
8: $X := X + 1$
9: **if** $((1 < i < |U_s|)$ AND
10: $(U_s^i \leq V_u^j < U_s^{i+1}))$ **then**
11: $X := X + 1$
12: **if** $((i = |U_s|)$ AND $(U_s^i \leq V_u^j))$ **then**
13: $X := X + 1$
14: **return** X
15: **function** $MAIN$
16: $P := \emptyset$, $numP := 0$
17: /* *The first phase calculates for each system Update Rate (U_s) and each variable class (C) the set of packet descriptions (P) from the request size (σ) and the packet transmission time (θ).* */
18: **for** $j := 1$ to $|U_s|$ **do**
19: **for** $k := 1$ to $|C|$ **do**
20: $\sigma := C_l^k \cdot Packet(j, C_c^k)$
21: **for** $g := 1$ to (d/U_s^j) **do**
22: $numP := numP + 1$
23: $r := rand('Gauss', 0, 10^{-3})$
24: $\theta := (g - 1)U_s^j + r$
25: $P := P \cup (\theta, \sigma)$
26: sort_by_transmission_time(P)
27: /* *The second phase uses the set P to calculate the transmission time taking into account the available bandwidth (b).* */
28: **for** $j := 2$ to $numP$ **do**
29: $t := P_\sigma^{j-1}/b$
30: **if** $P_\theta^j < P_\theta^{j-1} + t$ **then**
31: $P_\theta^j := P_\theta^{j-1} + t$
32: **return** P

which the configured update rate is smaller than the second system update rate. Requests for all such variables will be sent out during the first system update rate. The *second condition*, given at line #9, counts variables that are between two sequent system update rates, i.e. U_s^i and U_s^{i+1}. Finally, the *third condition*, given at line #12, counts all variables that have a greater update rate than the last system update rate, i.e. $U_s^i \leq V_u^j$ for $i = |U_s|$.

As a result, the algorithm generates a set of packet descriptions, P, denoted by the transmission time and the size of each packet.

4.5 Validation and experimental results

The proposed algorithm generates an ordered sequence of packets described by time and size. A prototype of the algorithm was implemented in Matlab [The13] and it was used in the validation process. The validation included several statistical methods comparing the predicted traffic model with traces captured from a real SCADA installation.

Three different applications were used to validate the proposed approach. Each one included a different number of variables, variable classes and variable update rates. The real traffic for each application was then compared to the generated traffic models in terms of: number of packets, packet sizes, throughput, and packet inter-arrival time. The comparison of packet inter-arrival time between real and predicted traffic was done separately for each system update rate. The results show the level of accuracy of the predicted traces. The throughput analysis permits the validation of packet sizes and packet transmission time.

For evaluating errors between the predicted and the real traffic we used the Mean Absolute Percentage Error (MAPE). Error is defined as actual or observed value minus the forecasted value. Percentage errors are summed without regard to sign to compute MAPE. According to Lewis (1982) [Law08], the lower the MAPE the more accurate the forecast. This way, it is considered that a MAPE value less than 10% shows a high accuracy, while between 11% and 20% we can assume a good result. A result between 21% and 50% is considered reasonable. However, in case the error is larger than 51% it can be a clear indicator of inaccuracy. Throughout the validation process we used the

following equation to calculate MAPE:

$$\frac{100}{n}\sum_{t=1}^{n}\left|\frac{R_t - P_t}{R_t}\right|, \tag{4.1}$$

where R_t is the real value in the period t, P_t is the predicted value in period t and n is the number of periods used in the calculations.

Table 4.2: Experimental results comparing the predicted and real network traffic

Application		Number of packets	Packets sizes	Throughput (bytes/sec.)			Packets inter-arrival time error				
				Avg.	Max.	Min.	50ms	500ms	1s	3s	6s
1	Predicted	29400	114	2793	2964	2736	**4.88%**	**0.70%**	**0.32%**	**0.37%**	**0.12%**
	Real	29165	113/114	2758	5928	113					
	Error	**0.80%**	**48.16%**	**1.27%**	**50%**	**2321%**					
2	Predicted	29400	132	3234	3432	3168	**8.34%**	**0.28%**	**0.32%**	**0.06%**	**0.20%**
	Real	29421	132	3236	6336	660					
	Error	**0.07%**	**0%**	**0.07%**	**46%**	**380%**					
3	Predicted	29400	152	3724	3952	3648	**4.53%**	**0.15%**	**0.08%**	**0.01%**	**0.01%**
	Real	29407	152	3725	6688	760					
	Error	**0.02%**	**0%**	**0.02%**	**41%**	**380%**					

4.5.1 Number of packets

As shown in Table 4.2 the calculated error for the predicted number of packets is smaller than 1%. This is because even if the number of variables, variable classes and variable update rates are different for each application, variables can be placed in the same request, and will generate the same number of packets.

4.5.2 Packets sizes

For the second and third application, packet sizes are exactly the same in the predicted and real applications, giving an error of 0%. Due to the fact that the analyzed protocol has a dynamic field, i.e. the sequence number, which can change its size, for the first application we recorded an error of 48.16%. The same can be observed in Figure 4.5. The second and the third applications have the same maximum and minimum peaks while the first application shows that predicted traffic's throughput is higher than the real one. At this point the proposed algorithm does not take into account the variation of packet sizes, and this is why almost half of the real packets from the first application have a different size. However, an improved version of the algorithm can take dynamic sizes into account, but this would need more investigation and we consider it as part of our future work.

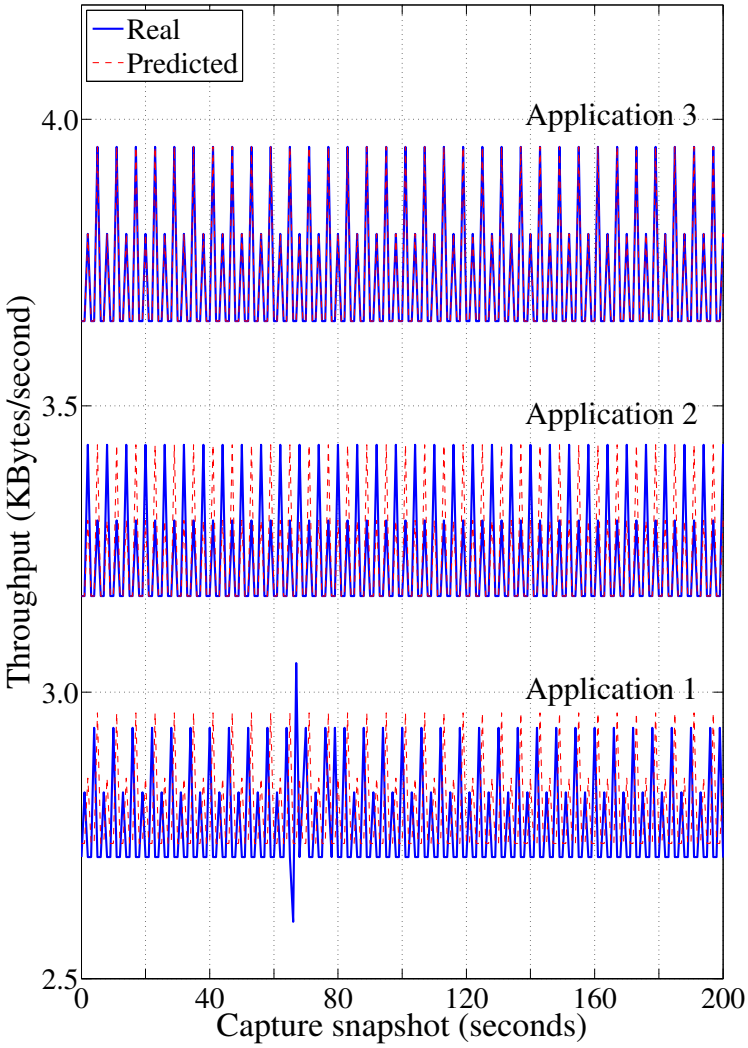


Figure 4.5: Comparison of throughput between predicted & real traffic

4.5.3 Throughput

The predicted throughput can be extracted by splitting the output of the algorithm in blocks of one second and summing the number of bytes. The following equation shows

how to obtain the throughput for a specific second:

$$Throughput(second_x) = \sum |P_{length}| : \\ second_x \leq P_{time} < second_{x+1}. \tag{4.2}$$

As shown in Table 4.2, in all cases the average throughput error is smaller than 2%, demonstrating the accuracy of the proposed algorithm. However, the errors for maximum throughput are bigger than 41%, while the error for minimum throughput for the first application is about 2321%. These maximum values are due to real system operations in which transmission times can be delayed, leading to a drop in the throughput value. In the next seconds these are followed by transmissions of the missed packets, leading to an increase in the throughput value. Consequently, the measured errors in some cases increase to large values. However, in the next seconds the error drops back to values smaller than 2%. A clear example of this behavior is also shown in Figure 4.5, where in the case of application 1 we can see a drop in the throughput, followed by a peak.

4.5.4 Packet inter-arrival time

All the errors we measured for packet inter-arrival time were smaller than 10% and in most cases the error was less than 1%. As expected, the largest error was measured for the smallest update rate, i.e. 50ms, that is mainly due to the non-real-time OS we used to run the SCADA server and due to network delays. In order to estimate the distribution of packets in the predicted and real traffic, we also calculated the Cumulative Distribution Function (CDF), shown in Figure 4.6. These figures show the accuracy of the predicted and of real applications for the configured system update rates. In each figure the horizontal axis is a ten millisecond time interval (given in seconds), while the vertical axis represents the percentage of packets for each system update rate.

As we can clearly see in Figure 4.6, the largest difference between the predicted and real packet distributions is for a 50ms update rate. The reason for this was already discussed in the previous paragraph and it can also be justified by looking at the measured errors in Table 4.2. The following figures show a more accurate CDF with the increase of the system update rate. By inspecting these figures we also notice that packet distributions

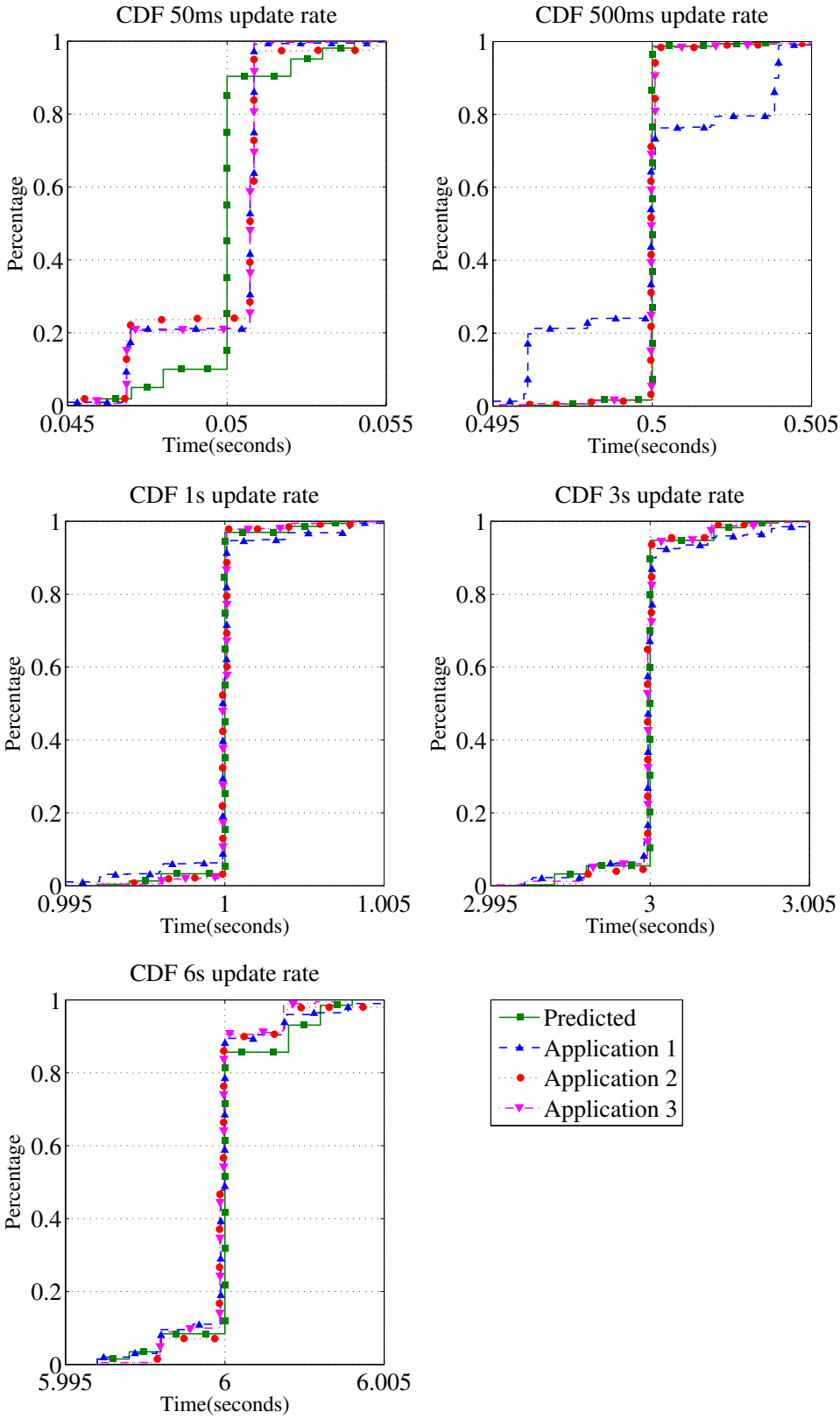


Figure 4.6: The calculated Cumulative Distribution Function (CDF) for each system update rate.

can be specific to each application. In fact, in order to predict more accurately the small variations in the distribution that we can see in each case, the proposed algorithm used a Gaussian distribution function. This way we increased the accuracy of the approach and, as already discussed, we managed to obtain an error of less than 1% in most of the cases. Nevertheless, we believe that the algorithm can be further improved in order to take into account more application-specific variations and to further reduce all errors below 1%.

4.6 Conclusions of the chapter

Recent approaches in the field of ICS traffic modeling [Mah10; Roo08; Che06] proved that the construction of accurate network traffic models is a difficult task. The time and resources required to construct realistic models constitutes one of the biggest issues that engineers must solve. The automated model construction based on SCADA application characteristics can effectively avoid these issues. Therefore, this chapter proposes an approach to generate network traffic models for ICSs, starting from application-specific data such as the number of variables, variable types and variable update rates. The presented algorithm takes as input an application description and generates a traffic model of a given duration. The algorithm was constructed by inspecting the network traffic taken from a real SCADA installation including several applications. As shown by experimental results, the algorithm is highly accurate and it can predict the number of packets by an error $< 1\%$, the packet sizes by an error of 0%, the throughput by an error $< 2\%$ and the packet inter-arrival time by an error $< 10\%$.

The main advantage of the proposed approach is that traffic models can be re-generated easily if needed, without having to analyze real traffic or to develop new models. Furthermore, the approach can be applied in several domains, starting with ADSs, to the recreation of network traffic in simulated/experimental environments.

Chapter 5

A method to correct delay and jitter issues

In Chapter 4, a method to construct ICS network traffic is presented. The exposed method uses an algorithm that having an ICS application description, gives the expected packet size for a certain period. The algorithm output has different applications, such as a base for a behavioral model for an ADS or planning of network traffic.

ICS devices, as already discussed in chapter 2, are designed with limited computational resources, and therefore, they are only able to satisfy process control tasks. Thus, the installation of additional software such an anti-virus or an ADS, is not feasible due to the lack of resources. Usually, ADSs are hosted in additional computers with common operating systems and with access to the network segments desired to monitor.

Computers use time as a reference for multiple applications such as event logs, network communication, etc. Log applications save the timestamp of each event in order to register when they happened. Applications using network communication, for instance email, need the time reference to know when an action took place.The system clock in a computer is based on registers and oscillators. Registers, composed of a number of bits, hold the time and oscillators, which add bits to registers, keep track of the ever-changing time. Far from being stable, oscillators suffer from changes in temperature, air pressure, magnetic field and so forth. For this reason, real system clocks have a frequency error, hence, even if they can keep the track of time within a specified accuracy they can be unreliable in some cases.

The time reference is essential in computer environments, but the frequency error of real clocks makes it untrustworthy. In a single computer, some applications like databases have difficulties when the time jumps backwards. Due to these errors, in a networked

environment where many computer applications have to work simultaneously, it is not feasible to have several computers each one with its own time reference. Therefore, computers not only need a time reference, they need a synchronized common time reference.

Time synchronization between several computers can be achieved following different strategies. On one hand, in centralized environments, where there is a main computer and many subordinates, the central server can establish the time reference. Berkeley Algorithm [Gus89] and Cristian's algorithm [Fla89] are two examples of time synchronization in centralized environments. On the other hand, in distributed systems, time synchronization is a complex task; one of the most used clock synchronization solution is the Network Time Protocol (NTP) which is based on the UDP protocol [Mil85; Mil88; Mil89; Mil92; Mil10]. Nowadays the NTP protocol is in its fourth version (NTPv4) and it is able to keep the clocks synchronized with a maximum difference of ten milliseconds across the Internet and with a minimum of two-hundred microseconds in ideal conditions [Mil10]. Another time synchronization solution is the Precision time protocol (PTP) [Wan11] defined by the standard IEEE 1588 [IEEE02; IEC04] which provides time synchronization with up to nanosecond precision over ethernet networks.

In the last decades ICSs have evolved from isolated networks and proprietary hardware and software to the usage of COTS hardware, standard communication protocols, common software, also to be connected to corporate networks and/or to the Internet. As already discussed in Chapter 2, this evolution brings together several positive as well as negative aspects. Although most modern ICSs are connected to corporate networks and/or Internet, best practice guidelines and standards of well known entities such as the Centre for the Protection of National Infrastructure (CPNI) suggest to limit as much as possible network interconnections in order to keep ICSs as secure as possible [Cen10]. Therefore, most ICSs are still isolated from other networks or they just permit necessary traffic such as SCADA traffic. In isolated or restricted environments, it is uncommon to find time synchronization protocols, specially those like NTP which require an external server as reference. Thus, on ICSs, it is usual to find unsynchronized clocks which fortunately does not affect the process itself.

Even if unsynchronized system clocks do not affect ICSs normal operation, in the case of an ADS with a detection engine based on time, it is necessary to maintain the clocks of different devices in good harmony; the false positive rate could be increased due to

device clock differences, decreasing the ADS performance.

In this chapter, a method to correct packet delay and jitter issues is presented. The exposed method is based on an algorithm that does not change the system clock, it works together with the ADS in order to modify its time reference to be synchronized with the ICS network traffic source device.

5.1 Methodology

The goal of the proposed method is to correct the ADS clock reference based on the degradation of static system update rates. It focuses on the variable value update requests between the SCADA or Control server and PLCs or RTU. Before the actual construction of the algorithm, a thorough analysis of real traffic between SCADA servers and ABB PLCs [Abb13] was conducted. The analyzed network traffic was captured in the Experimental Platform for Internet Contingencies (EPIC), the emulation testbed of the Institute for the Protection and Security of the Citizen (IPSC), Joint Research Centre (JRC) of the European Commission. The EPIC testbed is based on the Emulab software [The12], that is able to recreate a wide range of experimentation environments in which researchers can develop, debug and evaluate complex systems [Whi02]. The testbed characteristics and performance has been studied and tested in several research works [Sia12; Gen12; Pér10].

By using the EPIC platform, we constructed the topology shown in Figure 5.1. This topology included a real PLC, a SCADA server together with HMI software, a network switch and a dedicated node to detect anomalies and capture network traffic. All the devices were connected to the same commuted LAN. In order to capture all the traffic between the SCADA server and the PLC, the switch port mirroring feature has been used.

The PLC we used was ABB's 800M with the Manufacturing Message Specification (MMS) protocol. As shown in Figure 5.2, the MMS protocol is located above other protocols such as the Connection-Oriented Transport Protocol (COTP) and ISO Transport Services on top of the TCP (TPKT).

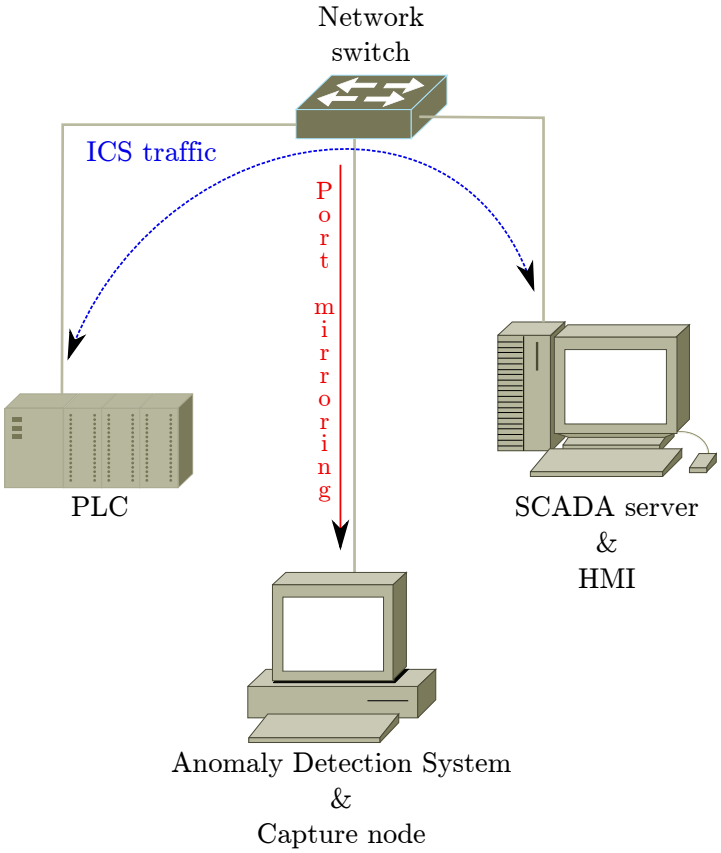


Figure 5.1: Experimental network topology

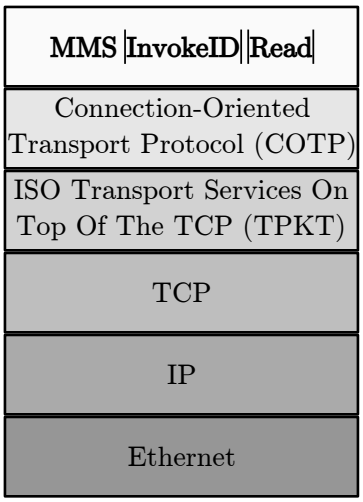


Figure 5.2: MMS protocol layers used by SCADA servers and PLCs from ABB

The communication between the SCADA server and PLC begins with a setup or initia-

lization process. Then, it continues with a loop that issues requests at each update rate in order to refresh the variable values. At this point we assume that the initialization process can be different for each communication protocol and device, and specific to each vendor. Our work focuses on the second part of the communication in which variables are updated. Figure 5.3 shows in detail the initialization process and the loop we are focusing on, i.e. *Repeat until Process duration.*

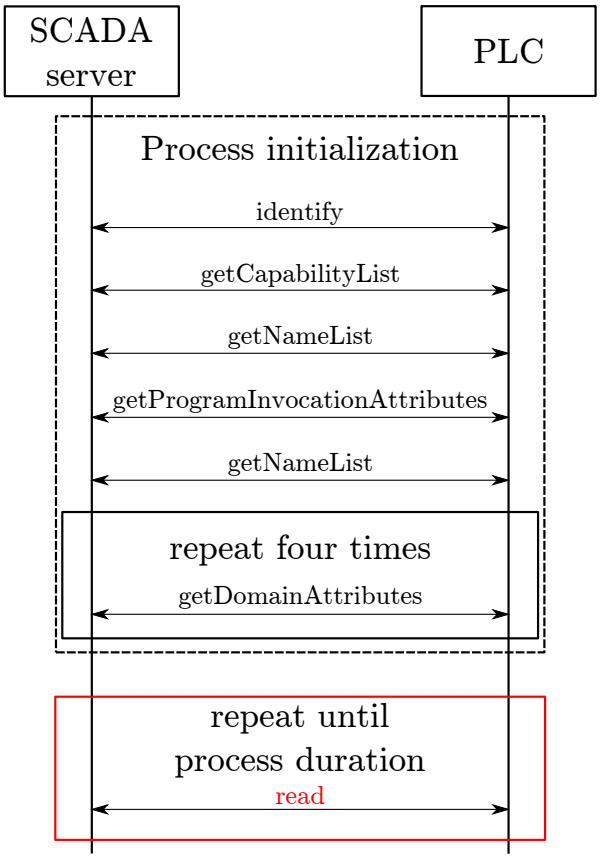


Figure 5.3: The sequence of messages exchanged by the SCADA server and PLC

In Chapter 4, as shown in Table 4.1, several applications, each one with different characteristics, were generated in order to analyze the created traffic in terms of size and quantity of packets. The same traffic has been analyzed in this chapter but with timing issues in mind. The inspected traffic has been broken down into different system update rates and its analysis has been used to construct the proposed algorithm.

5.2 Delay and Jitter of Analyzed Traffic Packets

As already discussed in Chapter 4, Control or SCADA servers that follow a *pulling* strategy, send variable value update requests to PLCs or RTUs. These update requests are performed with a repetitive pattern following system update rates. For each system update rate, all variables that satisfy two conditions are joined to form a unique request. The first condition is that variables must correspond to a certain variable class. The second one specifies a range of variable update rates that joint variables must satisfy. Thus, for each system update rate there would be a set of requests, each one corresponding to a different variable class. If a single variable class and a specific system update rate are considered, there will be a packet repeated continuously among process duration with a predefined time interval between two consecutive transmissions.

At this time, it is important to clarify some concepts that could induce an error. The delay of a packet, also referred to as latency and measured in fractions of seconds, specifies how long it takes for data to travel across a network from one computer to another [Com08]. There are various types of delay:

- ***Propagation Delay:*** The time required for a signal to travel across a transmission medium. Some delay in a network arises because a signal requires a small amount of time to travel across a transmission medium.

- ***Access Delay:*** The time needed to obtain access to a transmission medium. It depends on the number of stations that contend for access and the amount of traffic each station sends.

- ***Switching Delay:*** The time required to forward a packet. The time required to compute a next hop and begin transmission. An electronic device in a network (e.g., a Layer 2 switch or router) must compute a next-hop for each packet before transmitting the packet over an output interface.

- ***Queuing Delay:*** The time a packet spends in the memory of a switch or router waiting to be selected for transmission.

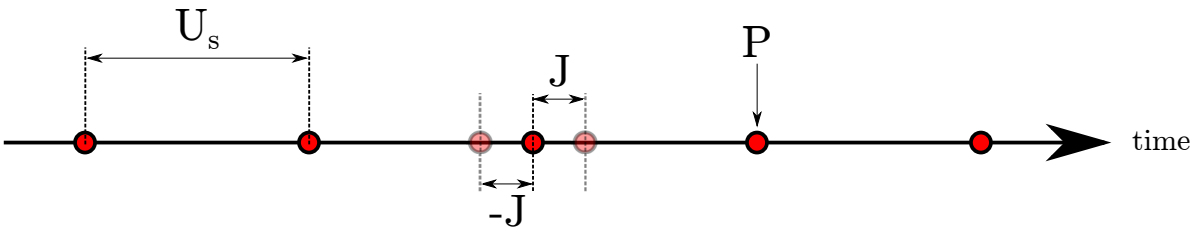
- ***Server Delay:*** The time required for a server to respond to a request and send a reply. Although not part of a network per se, servers are essential to most communication. The time required for a server to examine a request, compute it

and send a response constitutes a significant part of overall delay. Servers queue incoming requests, which means that server delay is variable and depends on the load of the moment.

The variation of packet delay is commonly known as *jitter* [Dem02]. However, the term jitter is used in different ways by different communities, hence, it is important to know which are the different meanings:

- The first meaning is the variation of a signal with respect to some clock signal, where the arrival time of the signal is expected to coincide with the arrival of the clock signal.
- The second meaning has to do with the variation of a metric, in our case the delay, with respect to some reference metric, in this case the minimum delay. It is also named as Packet Delay Variation (PDV) [Int11].

The second meaning is frequently used by computer scientists and frequently (but not always) refers to variation in delay. In our case, and for the rest of the document, we will use the term jitter to refer to the variation of packet delay.



P = Packet
U_s = System Update Rate
J = Packet Jitter
-J = Negative Packet Jitter

Figure 5.4: Packets, System Update Rate and Delay explanation.

As shown in Figure 5.4, the time between two consecutive packets is called system update rate. Depending on which system update rate is used by a specific packet type, the time between consecutive packets will be longer or shorter. In a networked environment packets are created by an application, transmitted by a Network Interface Controller (NIC), propagated across the wire, processed and retransmitted by networking devices, received by the destination NIC and finally processed by the destination application. In

ideal conditions, the time between packets is a constant value. However, in real traffic, it suffers small variations even if it pretends to maintain constant. Therefore, jitter can be positive or negative. Positive jitter takes place when a packet is transmitted or received after it was expected. On the contrary, negative jitter happens when a packet is transmitted or received before it was expected. While IT systems are typically focused first on confidentiality, then on integrity and finally on availability, ICSs emphasize availability more than integrity and confidentiality. In addition, ICSs are real-time or time-critical systems, which must deliver packets in a deterministic time. Thus, in IT systems packets are created depending on the current load when it is possible, in ICSs packets must be created when it corresponds, even if the current load can introduce small variabilities. During the life cycle of a packet, it can suffer delays at the origin, during the transmission and finally at the reception. Multiple factors could be the source of the suffered delay, acting individually or in conjunction. These are some of the factors that could influence a packet delay:

- At the source
  - The application which originates the packet cannot create it due to current load. Server delay.
  - The source application cannot deliver the packet to the NIC. Server delay.
  - The packet waits in the buffer of the NIC. Queuing delay.
  - The NIC cannot send the message because of a network congestion. Access delay.
- During the transmission
  - Packet has to be retransmitted due to network collision.
  - Network devices such as switches introduce a latency due to packet processing time. Switching delay.
  - Packet needs time to go across the wire. Propagation delay.
- At the destination
  - Network card buffer is out of space and cannot process the packet, it has to be discarded and retransmitted. Queuing delay.

– The packet waits in the buffer of the NIC. Queuing delay.

– The application of packet destination cannot process the packet due to congestion. Server delay.

As shown, multiple factors can be the origin of packet delay, but none of the presented causes could generate a negative packet jitter. All the given reasons can only cause a positive delay and jitter. In fact, the delay, also known as latency, usually is understood as an incremental value. A system with a low load, which generates periodical packets and delivers them to another host using always the same network, without congestion, the delays should be constant. Thus, there will not be any jitter. But as already mentioned, no jitter situations only happen in ideal conditions; in real networks there is always jitter, even if it is small.

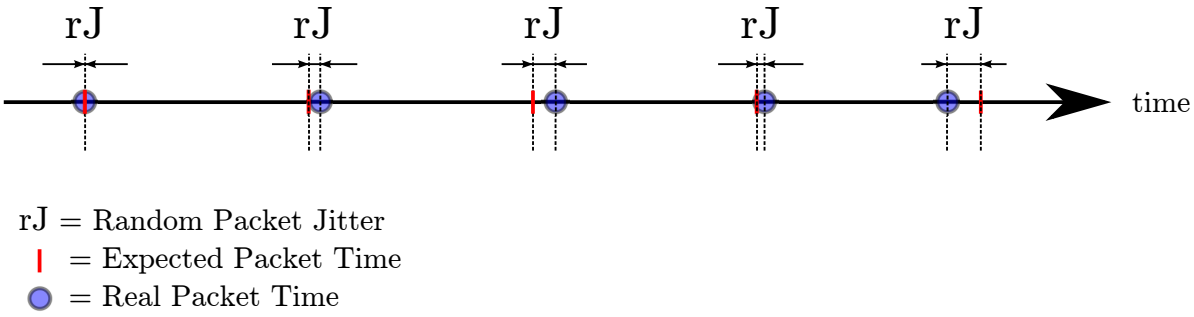


Figure 5.5: Random delay of Packets.

An ADS, as well as a PLC located in separated hosts as shown in Figure 5.1, expects a succession of packets with an interval of system update rate. In fact, for each system update rate, a set of packets will be delivered. Thus, a combination of all dispatched packets at all system update rates, generates a set of multiple packets that for an outside observer could be a fuzzy conjunction of packets. Figure 5.5 shows a succession of expected packet times combined with real packet times. It can be observed that just the first packet arrives when it was expected; the rest have a small delay variation that in this case it can be explained as a random packet delay. A random packet delay generates a random jitter that it depends on multiple factors as has been already explained.

Figure 5.6 shows a different situation. In this case, only the first packet arrives at the expected time. The rest accumulate a constant positive delay that in each iteration makes the real packet time get away from the reference time. Multiple reasons could be the origin of this accumulated delay, these are some of them:

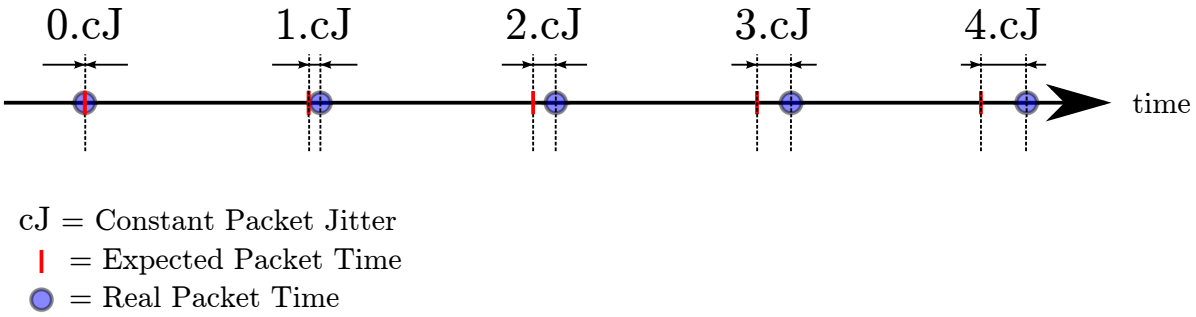


Figure 5.6: Packets with accumulated constant delay.

- The expected packet time is not correctly calculated, it is smaller than real.
- In the SCADA server, the trigger to create and send a new variable update request is set every time the last message has been sent. Thus, to the system update rate the queuing, access and propagation delays must be added.
- The clock oscillator of a SCADA server is imprecise, therefore, the system clock slowly lags behind. Thus, the time difference between packets is greater than the system update rate.
- The clock oscillator of the ADS host is imprecise, therefore, the system clock slowly steps forward. Hence, for the ADS the time between packets is greater than system update rate.
- A combination of some or all of them.

The fourth case describes a possible reason in which the delay is originated by the clock reference of the destination host. In this case, for the rest of devices that receive the same packets, they do not suffer any jitter. Thus, this is not jitter itself, but the consequences in a real-time system could be the same.

As shown, jitter can originate for multiple reasons, some of them would create random jitter while others would generate a constant one.

5.2.1 Delay of real ICS traffic packets

The algorithm presented in Chapter 4, generates ICS network traffic descriptions based on ICS application specifications. The explained algorithm classifies the application

variables based on variable classes and variable update rates. Then, depending on a set of system update rates and variable classes, it calculates the size of each variable value update request. Although the main idea is to use all system update rates at once and generate an output combining all of them, an individual output can be obtained for each system update rate. This provides an easier understanding of the process.

As described in Section 5.1, an ICS has been used to generate real network traffic. The used ICS is composed of six system update rates. In order to check the differences between predicted and real traffic in terms of delay and jitter, several ICS applications, each one with different characteristics, have been used to generate the network traffic. The ICS applications have run for twenty minutes and then, the captured traffic has been broken down into six different system update rates.

As already described, the algorithm presented in the previous chapter can be used to generate individual traffic descriptions of each system update rate. Thus, the Matlab based implementation of the algorithm has been used to generate the predicted traffic of each application divided into different system update rates. The selected duration as the algorithm input has been twenty minutes. Later, the predicted and real traffic have been compared giving as a result an accumulative delay for each packet.

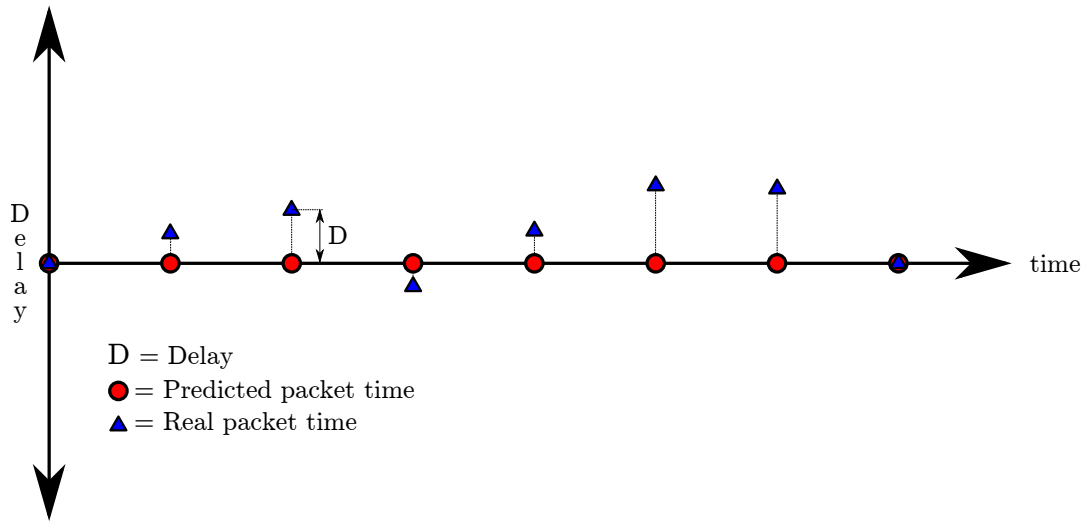


Figure 5.7: Explanation of the kind of figures used to describe the delay of each packet.

Figure 5.7 shows a time line in which there are placed the predicted packets times, illustrated as red circles, and the real packet times, illustrated as blue triangles. The

difference of height between predicted and real packets symbolize the delay of the real packet with respect to the predicted one. Note that a positive height means a positive delay, i.e. the real packet has been captured after it was expected, while a negative height means that the real packet has been captured before is was expected. The next figures use the same terminology to show the delay between predicted and real packets for each system update rate.

Figures 5.8, 5.9, 5.10, 5.11 and 5.12 show the time difference of each real packet with respect to the predicted one. As can be seen, all of them present a similar shape of delay. It can be observed that in each iteration a small portion of delay is accumulated, resulting in a total of near twenty milliseconds in twenty minutes. But the accumulated delay is negative in all cases, indicating that packets are received sooner than the corresponding predicted time.

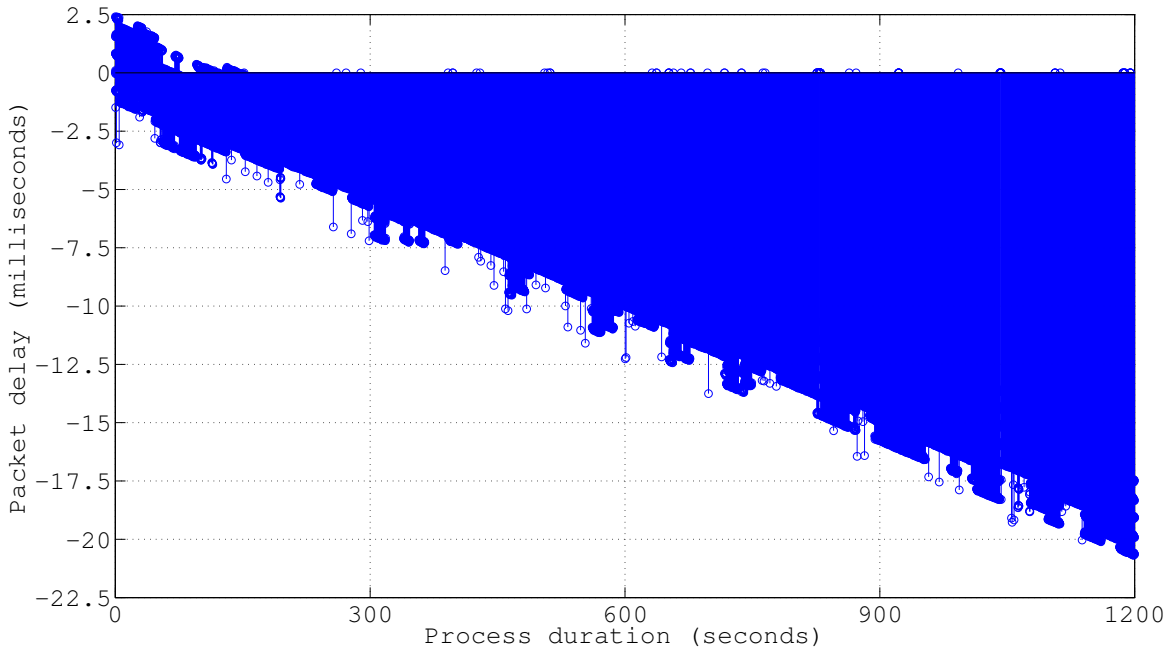


Figure 5.8: Delay of real packets respect to predicted traffic for 50ms system update rate.

There are multiple reasons that could explain why the packets are generated earlier than the corresponding system update rate. One of the reasons could be that in order to ensure that variable values are updated into the time system update rate, the SCADA server requests the updates a little bit earlier than established system update rate. Thus, even if there are small delays due to network congestion or the PLC is busy, it will be

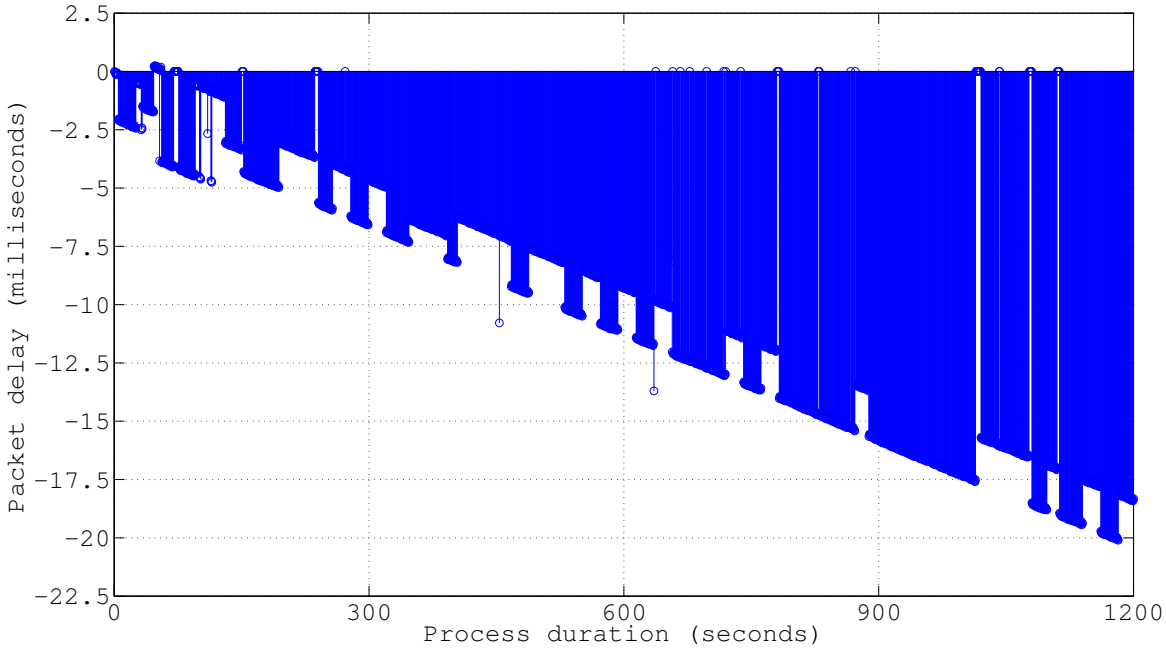


Figure 5.9: Delay of real packets respect to predicted traffic for 500ms system update rate.

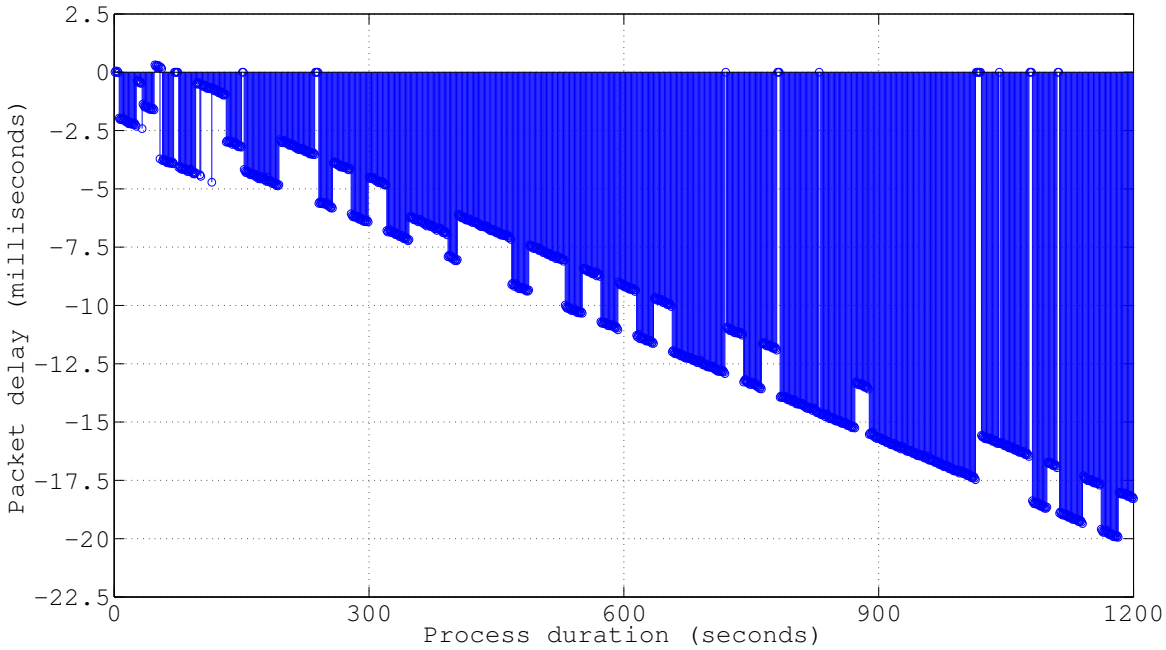


Figure 5.10: Delay of real packets respect to predicted traffic for 1000ms system update rate.

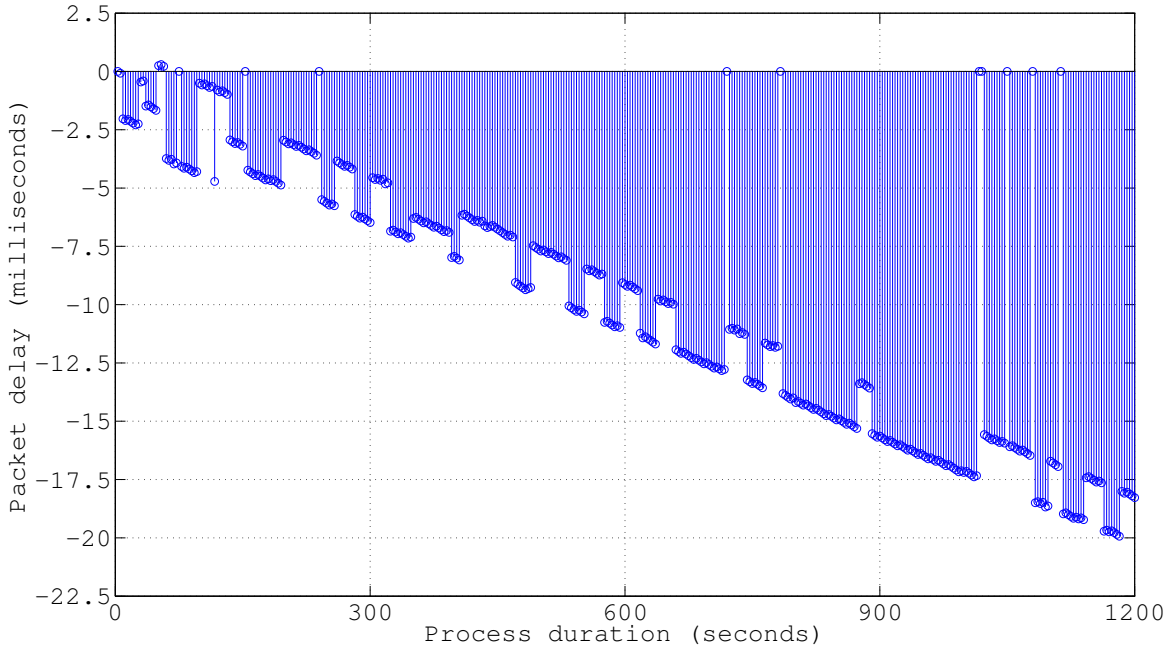


Figure 5.11: Delay of real packets respect to predicted traffic for 3000ms system update rate.

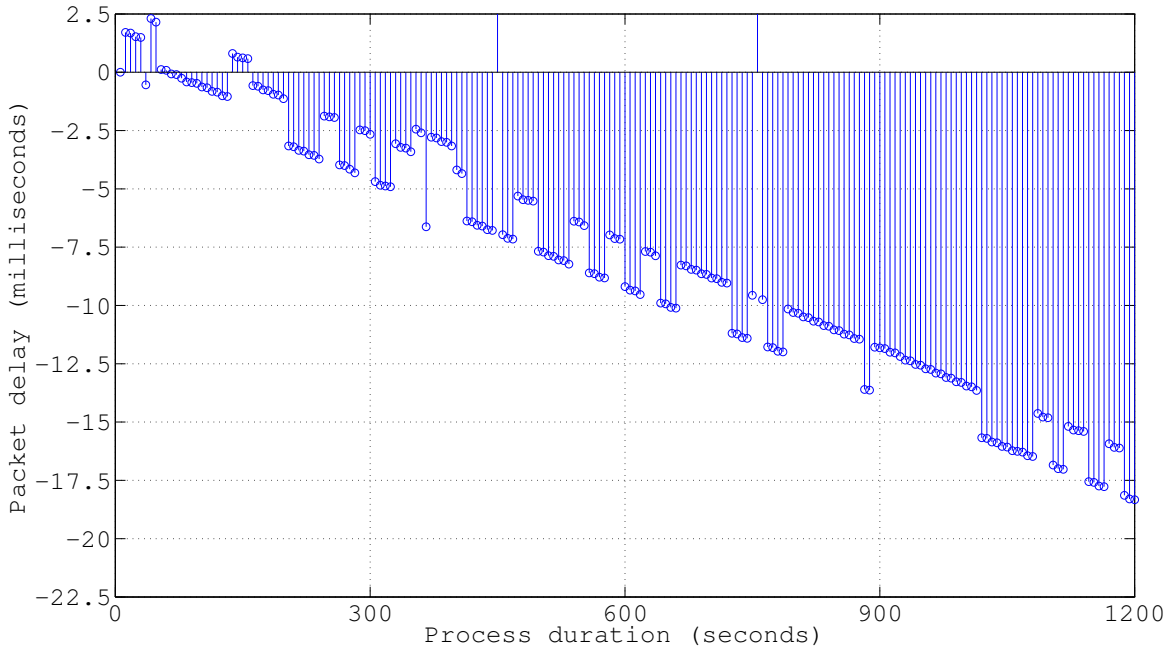


Figure 5.12: Delay of real packets respect to predicted traffic for 6000ms system update rate.

easier to meet the set up system update rate. Another possible reason is the inaccuracy of the clock oscillators of both nodes or at least one of them. On one hand, if the clock oscillator of the SCADA server is imprecise and it steps forward in each oscillation, the real system update rate will be shorter than the predicted one, thus, packets will arrive to the ADS earlier than they should. Even if the clock oscillator of the ADS system steps forward too but with a smaller step than SCADA server, the result would be the same. On the other hand, if the clock oscillator of the ADS is imprecise and it lags behind in each oscillation, the predicted system update rate will be longer than the real one, hence, for the ADS, packets will arrive earlier than they should. In addition, the clock oscillator of the SCADA server could lag behind, but in order to have the same consequence, the lags must be smaller than the clock oscillator of the ADS. Obviously, a combination of different options could increase or decrease the delay of each packet.

To the best of our knowledge, there is no evidence of any SCADA server that sends variable value update requests earlier than established system update rate. In addition, the EPIC emulation tested is composed of computers based on AMD Dual-Core family processors. These processors are known to have a timing issue which generates, together with other effects, a negative Packet InterNet Groper (PING) [Mil83] time measurements. The manufacturer released a patch to solve the issue, even if it does not openly admit it [Adv08]. Therefore, it suggest that the clock oscillator would be the most probable factor of packets' delay.

Even if for ICS devices the presented differences between real and predicted traffic apparently do not have negative effects, for time based ADSs the same cannot be said. The algorithm presented in Chapter 4 generates an output that can be used as reference for a time based ADS. The algorithm output establishes when the SCADA server should send each packet and what must be the throughput. Thus, it is important that the SCADA server and the ADS work in synchronization in order to decrease the false positive rate. As already discussed in Chapter 2, the quality of an ADS is measured in terms of false positive, false negative and detection rates. In addition, the usual approach to document the performance of an ADS is to relate the false positive rate to the detection rate. This is done by drawing Receiver Operating Characteristic (ROC) curves.

5.3 Correcting the Time Reference

As already discussed in the introductory part of this chapter, depending on the ICS installation, the ICS network can be totally isolated from external networks, connected to corporate networks or even connected to the Internet. In addition, there could be a time synchronization protocol working in the network or each system could be responsible for its own time. A time synchronization protocol such as NTP, could synchronize the system clocks in small variations. As a consequence it would reduce the possible effects of an imprecise clock oscillator. Hence, in an ICS installation there could be different strategies for clock synchronization:

- In an Internet connected ICS a time synchronization protocol such as NTP could be used.
- Even if the ICS is not connected to Internet, there could be an internal synchronization protocol to synchronize device clocks.
- As it happens with the serial line, specific applications could include a time synchronization mechanism.
- An ICS could be totally unsynchronized, in fact, for several years this has been the most usual situation.

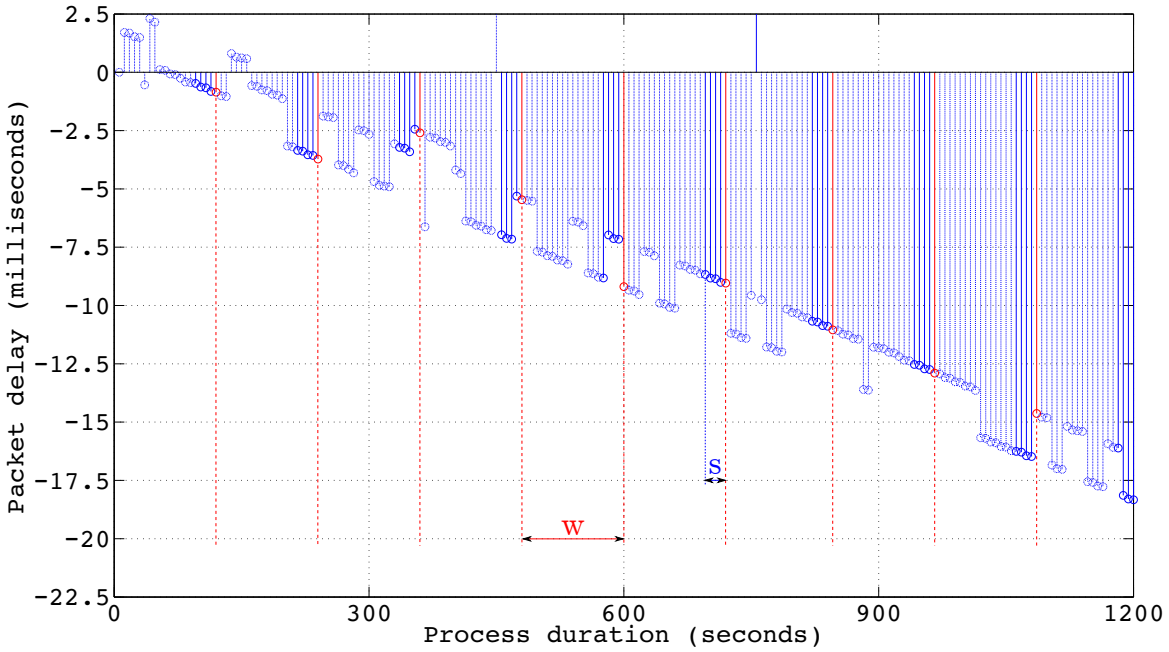
In this chapter, an algorithm is described that using SCADA server characteristics as well as generated packets, corrects behavioral model timing reference, thus, increasing the quality of the ADSs by reducing the false positives rate. The proposed algorithm corrects the traffic model based on an iterative process among the ICS process duration. Each ICS can be characterized by multiple factors, including the system update rates together with the process duration. These characteristics will determine the singularities of ICSs, and therefore, they must be included as the input of the presented algorithm.

The proposed algorithm takes the following sets and variables as input:

- P_R = set of real packet descriptions
- U_s = set of system update rates
- C = set of variable classes

- w = windows size (in packets)
- s = number of samples (packets) to calculate the mean delay
- f = correction factor

Each variable of the set of real packet descriptions, P_R, is a pair consisting of packet time and packet size, denoted by (θ, σ). As shown in Figure 5.13, the window size variable, w, denotes the interval (number of packets) between each reference correction calculation. In the same way, the number of samples, s, corresponds to the number of packets that each correction takes into account. The correction factor, f, is used to reduce or amplify the effect of each correction into the next group of packets.



W = window size (packets)
S = number of samples (packets) to calculate the mean delay

Figure 5.13: Window size and number of samples variables explanation.

As output, the algorithm generates a set of modified time packet descriptions P_M, where each packet is defined as a pair including the transmission time and the size of the packet, denoted by (θ, σ).

For simplicity, throughout the chapter we use the X_y^z notation to denote the y component of the z-th element in set X. For instance, taking the C set, the j-th element is denoted

by C^j, and the c component of the j-th element is denoted by C^j_c.

The algorithm starts with the *MAIN* function and it loops though all system update rates U^j_s, all variable classes C^h_l and all packet descriptions P^h_R. Next, it initializes u and $numP$ variables; the first one is used as the relative system update rate, while the later one is used to count the number of packets. It checks if each packet corresponds to the actual variable class, at line #9, and if the condition is satisfied, it continues by increasing the number of packets. Then, the time difference between the actual and the first packets is calculated, as well as the predicted packet time, and both of them are compared to calculate the delay error at line #13.

Whenever the number of packets $numP$ equals the window size w, the delay mean error is calculated from the last number of samples s quantity at line #17. Finally, the relative system update rate u is calculated, using the delay mean error and the correction factor f.

Algorithm 2 Packet delay correction

```
1: input :< U_s, P_R, C, w, s, f >, output :< P_M >
2:
3: function MAIN
4:     for j := 1 to |U_s| do // for each system update rate
5:         u := U_s^j // initialize update rate variable
6:         for k := 1 to |C| do // for each variable class
7:             numP := 0
8:             for h := 1 to |P_R| do // for each variable
9:                 if P_Rσ^h = C_l^h then // check if variable corresponds with actual variable class
10:                    numP := numP + 1
11:                    φ^numP := P_Rθ^h − P_Rθ^1 // calculate time difference respect first packet
12:                    P_Mθ^h := u · k // calculate theoretical packet time
13:                    ε^numP := φ^numP − P_Mθ^h // compare real and theoretical packet times
14:                    if numP = w then // if packet number is equal to windows size
15:                        /*calculate mean of real and theoretical packets' time difference
16:                        for 's' amount of packets*/
17:                        ρ := mean(ε^(numP−s) : ε^numP)/w
18:                        u := u + ρ · f // actualize u variable depending on correction factor
19: return P_M
```

As already explained, as a result the algorithm generates a set of modified time packet descriptions P_M, that could be used by a time based ADS.

5.4 Validation

The proposed algorithm generates an ordered sequence of packets described by corrected time and sizes. A prototype of the algorithm was implemented in Matlab [The13] and it was used in the validation process.

Different applications were used to validate the proposed approach. The applications were used both to create real ICS network traffic as well as to generate the predicted traffic by the algorithm presented in the previous Chapter 4. Then, the predicted traffic has been used as the input for the presented algorithm, obtaining a set of modified time packet descriptions. The output of the last algorithm has been compared with the real traffic in order to see if the accumulated delay is corrected and if the presented algorithm is able to follow the timing issues.

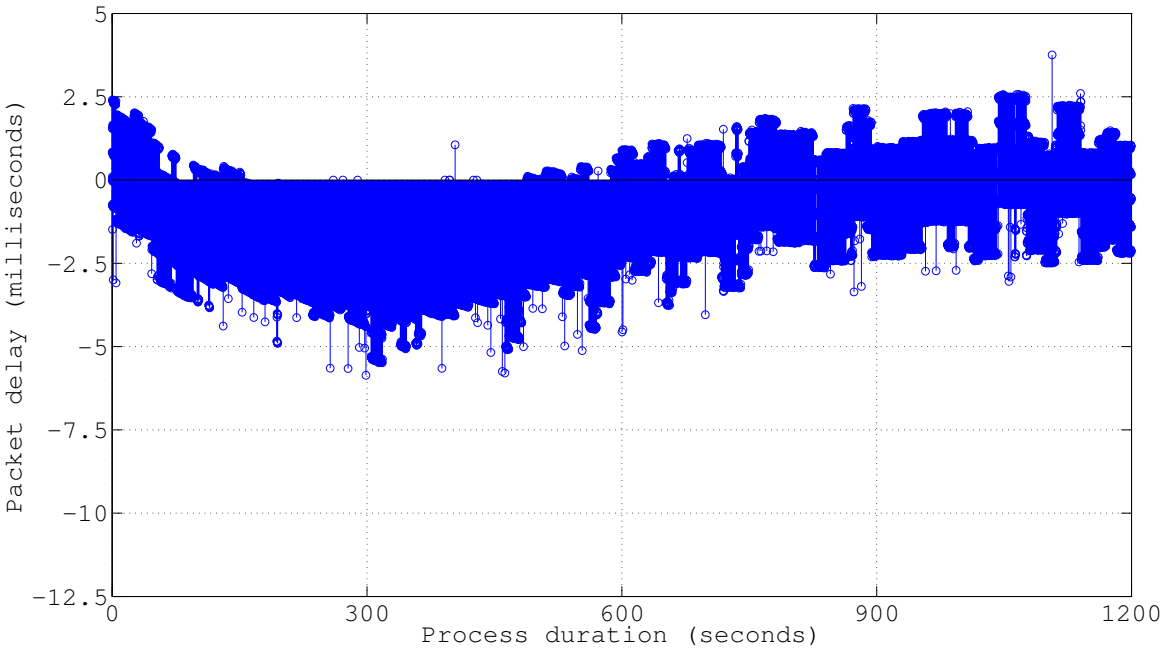


Figure 5.14: Delay of real packets with respect to corrected traffic for 50ms system update rate.

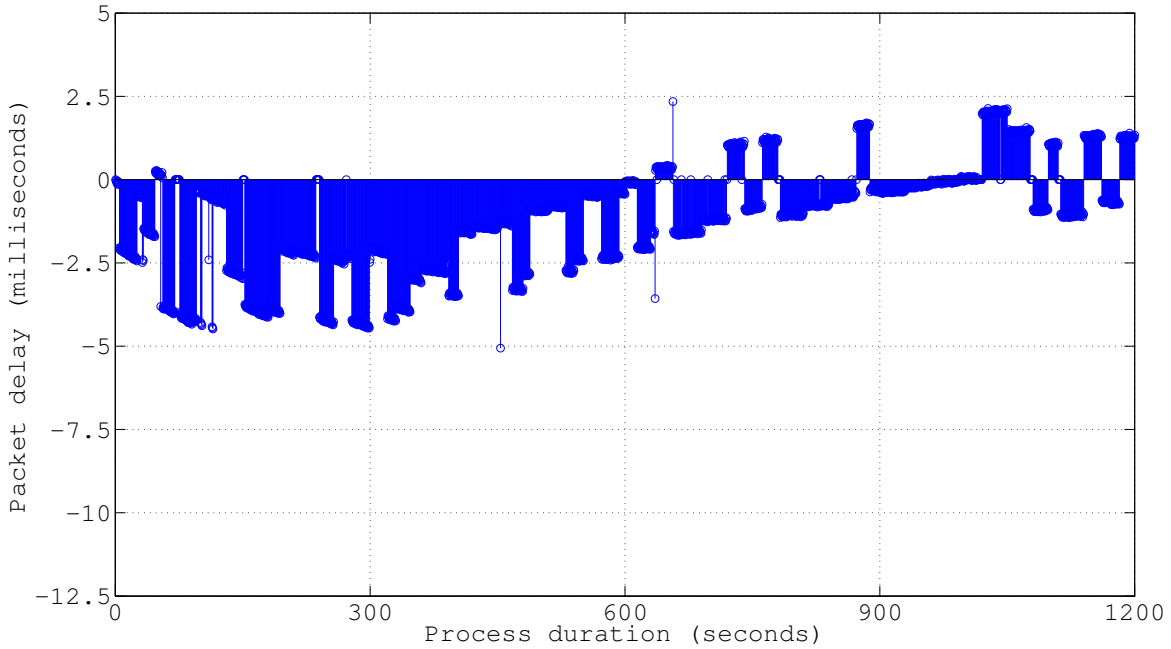


Figure 5.15: Delay of real packets with respect to corrected traffic for 500ms system update rate.

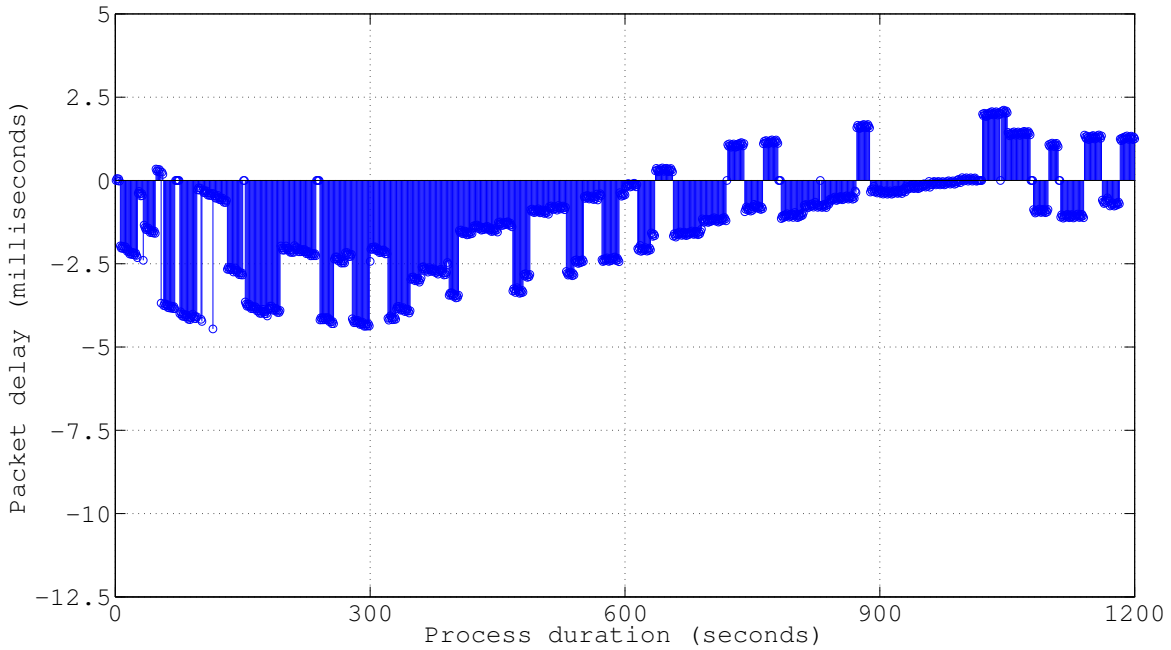


Figure 5.16: Delay of real packets with respect to corrected traffic for 1000ms system update rate.

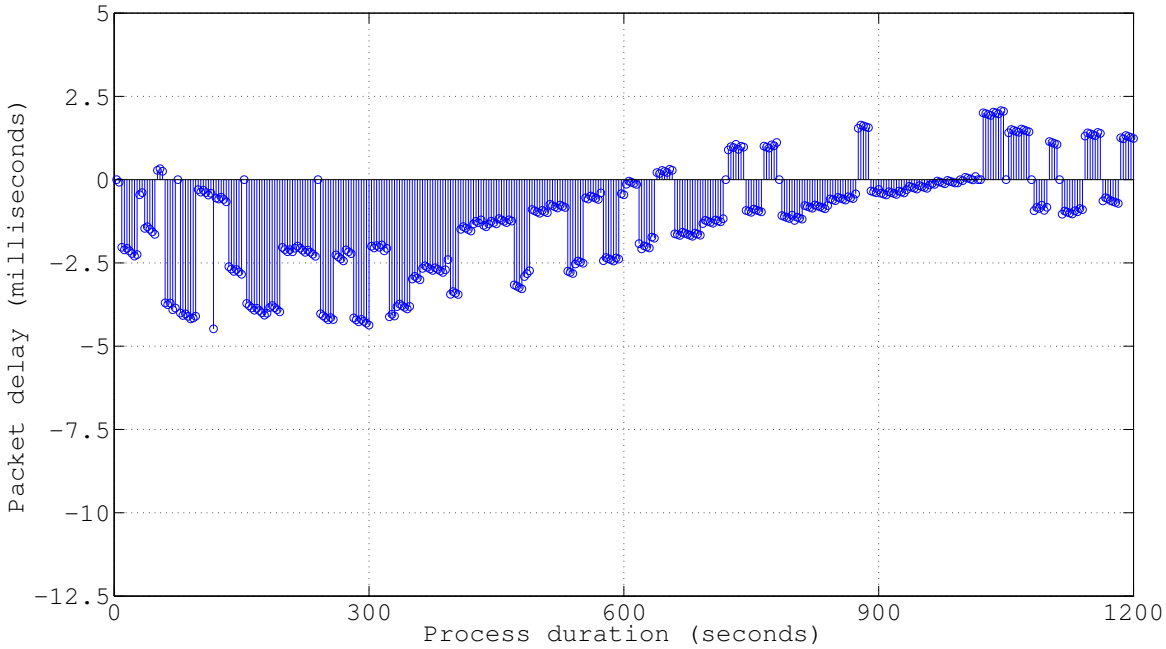


Figure 5.17: Delay of real packets with respect to corrected traffic for 3000ms system update rate.

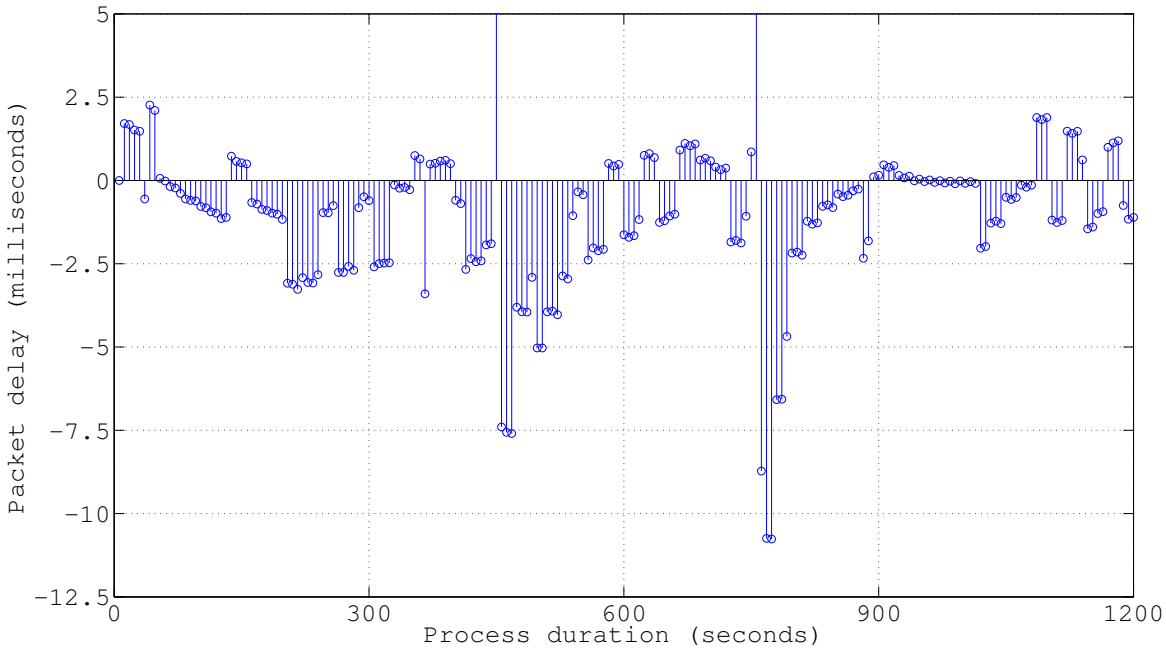


Figure 5.18: Delay of real packets with respect to corrected traffic for 6000ms system update rate.

Figures 5.14, 5.15, 5.16, 5.17 and 5.18 show the time difference of each real packet with respect to the corrected one. Excluding Figure 5.18, the rest show how the delay difference decreases in an iterative way, thus, compensating the accumulative error that has been showed by uncorrected traffic. In addition, the delay has its minimum approximately three hundred seconds after process was started, and the maximum after 18 minutes. Moreover, the delay is enclosed between -5 and 2.5 milliseconds. This, compared with uncorrected traffic, shows a 75% smaller error, which obviously will decrease the probability of a false positive.

Although Figure 5.18 shows a delay improvement with respect to Figure 5.12, due to the big positive delays at 450 and 756 seconds from the beginning of the process, the algorithm tries to compensate them by relatively big system update rate. Thus, a big positive delay is followed by a big but negative one. In this case, the error is not minimized as much as in the previous cases, even though the result improves the original delay. In the worst case, the minimum error is around minus eleven milliseconds, which shows an improvement of 40%.

The idea of the presented algorithm is to correct with small and iterative hops the reference of the system update rate. Hence, if the accumulative delay has small variations the algorithm will be able to follow them without any problem. The windows size ($\boldsymbol{w}$), the number of samples ($\boldsymbol{s}$) and the correction factor ($\boldsymbol{f}$) variables permit to adjust the algorithm behavior to the delay characteristics. The values of those variables used for the presented results have been selected in order to obtain the minimum delay error in average. Due to the fact that the number of possible combinations is significantly large, a Matlab program was developed in order to calculate which combination gives the minimum delay error in average. The calculations were performed individually for each system update rate. Table 5.1 shows the $\boldsymbol{w}$, $\boldsymbol{s}$ and $\boldsymbol{f}$ values used for each system update rate.

Table 5.1: Selected windows size, number of samples and correction factor values for each system update rate.

	System Update Rate				
	50ms	**500ms**	**1000ms**	**3000ms**	**6000ms**
Window Size ($\boldsymbol{w}$)	2	2	7	14	125
Number of Samples ($\boldsymbol{s}$)	1	1	1	2	9
Correction Factor ($\boldsymbol{f}$)	2	1	1	1	1

5.5 Conclusions of the chapter

The inaccuracy of system clock oscillators as well as SCADA server variable value update processes designed to keep distance from possible delay effects, may produce imprecise system update rates. The consequences could be diverse, especially in real-time systems and time-based ADSs. An increased false positive rate reduces the quality of any ADS. Therefore, where possible, all inaccuracy sources have to be removed, thus, increasing the ADS level of trust .

The presented algorithm, corrects the time reference of packets generated by the algorithm presented in Chapter 4. Those corrections are done by performing iterative variations to each system update rate. The algorithm takes as input the previously generated packet descriptions together with system update rates, variable classes and another three variables which adjust the algorithm to each ICS installation conditions. As the output, the algorithm generates a set of corrected time reference packet descriptions, which can be used as a behavioral model for a time-based ADS. As shown by experimental results, the algorithm is able to reduce the delay error almost to 25%; it can also minimize the effect of possible outliers as shown by Figure 5.18.

The main advantage of the proposed approach is that traffic models can be adjusted to each ICS installation conditions, thus, enabling their usage by time-based ADSs.

Chapter 6

Conclusions, Contributions and Future Work

In this chapter a comprehensive overview of the main thesis results is given. Section 6.1 summarizes the work performed during the completion of the research work. The achieved research has obtained several outcomes, which are listed in Section 6.2. During the investigation, different research tracks have been explored. Some of them have necessarily been discarded for the sake of completing thesis. Anyhow they deserve more attention and have been identified as future research areas in Section 6.3.

6.1 Conclusions

The research work carried out is a result of a multi-domain approach to ICS security. Thus results can be applied by different disciplines within the security community. Research fields such as anomaly detection, automatic signature generation, automatic traffic representation and bounding of delay and jitter have been included.

As part of the thesis work, a comprehensive review of different Anomaly Detection System (ADS) techniques and research projects focusing on Industrial Control Systems (ICSs) was performed, which was published as a conference paper [Gar11]. This survey shows the different researched techniques used to detect anomalies in ICSs as well as defining why behavior model-based Intrusion Detection Systems (IDSs) are preferred for ICSs over rule-based ones. This survey showed network-based ICS ADSs as an emerging research topic, which directly influenced the research fields followed during the thesis work.

We captured and analyzed several real ICS network protocol traffic packets. The ana-

lyzed protocol is proprietary and binary, which does not allow the interpretation of the payload content of the packets, and thus, the purpose of each one. Hence, a semantic analysis of packets' Protocol Data Unit (PDU) is not feasible. Traditional Information Technology (IT) IDSs such as Snort, include very few rules that correspond to ICS protocols. Additionally, the included rules belong to very few ICS protocols, e.g. Modbus and DNP3. Therefore, by default, traditional IDSs are able to detect a very low number of ICS attacks. Our approach uses a modified version of the Snort IDS to learn from common traffic traces, flagging previously unseen packets as anomalous. To do so, a system, described in Section 3.2 has been implemented. This system extracts useful information from packet payloads and automatically generates a behavioral model of the ICS traffic described as IDS rules. Rather than describing attack patterns, the generated set of rules synthesize the usual behavior. The system is able to detect anomalous behavior. Conversely, it is not able to describe which kind of attack causes it.

The behavioral patterns created from optimal conditions' usual traffic must guarantee its quality. To do so, it must be ensured that the traffic is free of any type of attack trace, which is a difficult task. In order to avoid the necessity of attack-free training data, a method is proposed to build real traffic synthetically. As described in Section 4.2, ICS applications can be characterized by the number of variables, variable classes and variable update rates. Several ICS applications (Section 4.3) were created in order to generate real ICS network traffic. Then, resulting traffic was analyzed based on several terms (Section 4.3), so changes on traffic originated by different ICS applications could be defined. As the outcome of the analysis, an algorithm, described in Section 4.4 was created. Having ICS application characteristics as input, the resulting method is designed in such a way that the algorithm gives an ordered set of packet descriptions, formed by expected packet timestamp and packet size. The output can be post-processed to calculate parameters like the network throughput. The design as well as its validation was published in two conference papers [Gar12a; Gar12b]. The output and its variations have diverse possible applicabilities, such as network capacity planning or behavioral modeling for an ADS. In this sense, we have focused in ADSs, leaving the other applicabilities as possible future work, discussed later in Section 6.3.

With an ADS as the final objective, the ICS network traffic generated by several ICS applications was compared, in terms of delay, to the predicted traffic created by the algorithm presented in Section 4.4. As we already described, the algorithm introduced

in Chapter 4 generates a set of packet descriptions, defined by packet timestamp and packet size. The packet timestamp establishes the expected delivery moment, hence, this information can be compared to real packet timestamps. The output of the comparison gave a repetitive delay pattern, as shown in Section 5.2.1. This pattern exhibits an accumulative negative delay which could result in an increase of ADS false positive rate, thus, decreasing the overall quality of the ADS. In order to compensate the accumulative delay, in Section 5.3 we present a second algorithm. This algorithm uses the outcome of the previous algorithm as input, and based on each ICS installation characteristics, it updates the timestamp reference of each packet. As output, it generates a new set of installation customized packet descriptions. The results covered in Section 5.4 show how the overall delay of packets has been decreased.

The three components presented in this dissertation can be further developed and improved, an aspect that is discussed in Section 6.3.

Regarding the security of ICSs, the ADSs are not the unique and neither the best solution. Security is a continuous process which has to evolve every day together with the complexity and capacity of threats progress day by day. The security of ICSs can be improved in multiple ways, ADSs are a promising area due to the repetitiveness and static patterns of the ICSs. Even though, as a whole and unique security solution, they are a small drop of water in the ocean. In the same way, the ADSs are very heterogeneous, there are of multiple colors and shapes and still the possibilities are enormous. This brings the opportunity to further research opportunities, as the conducted research in this area shows no sign of reaching an end.

6.2 Contributions

The following are the main contributions of this thesis work:

- This thesis provides a comprehensive review of ICS ADSs research works (Section 2.4).
- We provide a system that automatically analyzes, characterizes and creates behavioral models of ICS network traffic (see Chapter 3). In the same way, the system synthesizes each behavioral model into a set of IDS rules. The resulting set of

rules does not describe intrusion patterns, instead, it defines which is the usual traffic.

- Although the system has been designed to work with binary and private ICS protocols, we do not limit the scope of the system to a few ICS protocols. The experiments are run with real ICS traffic and we consider that the system is able to work with any type of public and binary ICS protocols as well.

- The generated set of rules describes what the network traffic should be, thus, in order to validate it, an IDS that works as an ADS has been necessary. Therefore, an open source IDS, Snort, has been modified in order to use the generated set of rules as a behavioral pattern.

- We present a method, together with an algorithm (see Chapter 4), that having ICS application characteristics, generates a set of packet descriptions, each one composed of a packet timestamp and packet size. Although the applications of the generated set of packets are diverse, we have focused on its use as a behavioral model for ADSs.

- The ICS application characteristics have never been used for ICS network data profiling (see Section 2.4). We demonstrate that this is a valid method for ICS network traffic characterization which generates useful behavioral patterns.

- We introduce an algorithm to diminish the effects generated by system clock imprecise oscillators to the delay of ICS network packets (see Chapter 5). We show that imprecise system clock oscillators could cause cumulative delays that could decrease the quality of an ADS by increasing its false positive rate. This algorithm changes the set of packet descriptions generated through the previous method by customizing to each ICS installation characteristics.

6.3 Future Work

For obvious reasons, this thesis work had to focus on limited research objectives. Anyhow, during this time, other interesting areas have been considered and analyzed. Even if they have been discarded for the completion of the thesis, they still deserve the atten-

tion of the scientific community. Here we present some of those topics and we depict how the work carried out during the thesis can further be followed.

- The automatic signature generation system for ICS private binary protocol, presented on Chapter 3, classifies the traffic based on characteristics such as source IP address among others. But, traffic flows can be classified taking into account multiple characteristics, including the packet size. Packets of a specific size in the same network protocols usually have a common purpose, even if their content changes over the time, e.g. several packets, generated to update the value of a temperature sensor, with the same source and destination, would have the same PDU size. Based on this assumption, we believe that a more extensive packet classification would provide a more explicit and reduced set of rules. Additionally, we believe that the new output would increase the quality of the ADS.

- The generated system was validated against a determinate ICS private binary protocol such as Siemens S7. Although we used a specific protocol, the method does not try to understand the content of packet PDUs, thus, we believe that the system should work with other kind of protocols, including public and text based ones.

- By using reverse engineering, it would be interesting to extract the traffic semantic by associating groups of bytes of unknown payloads, with the obtained values in the SCADA server.

- The method presented in Chapter 4, which automatically generates traffic models based on ICS application characteristics, is focused on the network traffic between a SCADA server and a PLC. In order to cover the whole communication schema, returning packets from the PLC should be analyzed and the presented algorithm should be updated thus generating a whole traffic model. This would allow not only the detection of anomalies out of traffic between the SCADA server and the communication network, but also the anomalies generated in the PLC, giving the opportunity to perform an early detection of possible attacks.

- The generation of the algorithm was based on a specific ICS public protocol. Although, the algorithm has been generalized so the output would be the same for other ICS protocols. It would be interesting to analyze its performance for all possible protocols.

- The main advantage of the proposed approach is that traffic models can be re-generated easily if needed. Furthermore, the approach can be applied in several domains, starting from ADSs, to the recreation of network traffic in simulated/-experimental environments. As future work, it would be interesting to apply the approach in the previously mentioned fields. Additionally, further work is needed to enhance it with application-specific analysis in order to further reduce errors in all measured cases below 1%.

- The algorithm presented in Chapter 5 corrects the timestamp reference of the packets generated by the method presented in Chapter 4. The algorithm takes as input a set of packet descriptions among other variables that describe the characteristics of an ICS installation, such as window size w, number of samples s and the correction factor f. Although several simulations have been done in order to determine which are the best values of those variables that give the minimum error, the simulation of all the possibilities takes months of work. Thus, further work is needed in order to obtain all the possibilities and try to find a relationship between variables corresponding to different system update rates.

- In the same way that the algorithm presented in Chapter 4 generates a traffic model between a SCADA server and a PLC, the algorithm described in Chapter 5 corrects the same traffic model. In the case of the traffic generated by a PLC, the delays would be different. Thus, as well as the update of the algorithm introduced in Chapter 4, further work is necessary to update the algorithm and include the timing corrections for the traffic originated by the PLC.

Bibliography

[Abb13] ABB Group. Abb, 2013. Online. Accessed on December 10, 2013. Available at: http://www.abb.com.

[Adv08] Advanced Micro Devices, Inc. AMD Dual-Core Optimizer, July 31, 2008. Online. Accessed on December 10, 2013. Available at: http://support.amd.com/en-us/search/utilities#k=AMD%20Dual-Core%20Optimizer.

[Bay12] Colin R. Bayliss and Brian J. Hardy. *Transmission and distribution electrical engineering*. Newnes, 4 edn., 2012.

[Big03] John Bigham, David Gamez, and Ning Lu. Safeguarding SCADA Systems with Anomaly Detection. In *Computer Network Security*, vol. 2776, pp. 171–182. Springer Berlin Heidelberg, 2003.

[Bio11] Philippe Biondi. Scapy, 2011. Online. Accessed on December 10, 2013. Available at: http://www.secdev.org/projects/scapy/.

[Bis02] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, December 12, 2002.

[Byr04] Eric Byres, Matthew Franz, and Darrin Miller. The Use of Attack Trees in Assessing Vulnerabilities in SCADA System. In *Proceedings of the International Infrastructure Survivability Workshop*. 2004.

[Byr12] Eric Byres. Understanding Deep Packet Inspection for SCADA Security. Tech. rep., Tofino Security, December 20, 2012. Online. Accessed on December 10, 2013. Available at: http://www.tofinosecurity.com/downloads/647.

[Byr13] Eric Byres. Project SHINE: 1,000,000 Internet-Connected SCADA and ICS Systems and Counting, September 19, 2013. Online. Accessed on December

10, 2013. Available at: http://www.tofinosecurity.com/blog/project-shine-1000000-internet-connected-scada-and-ics-systems-and-counting.

[Car09] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, and Alberto Trombetta. Scada Malware, a Proof of Concept. In *Critical Information Infrastructure Security*, pp. 211–222. Springer Verlag, Rome, Italy, 2009.

[Car10] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, and Alberto Trombetta. State-Based Network Intrusion Detection Systems for SCADA Protocols: A Proof of Concept In *Critical Information Infrastructures Security*, vol. 6027, pp. 138–150. Springer Berlin Heidelberg, 2010.

[Car11] Andrea Carcano, Alessio Coletta, Michele Guglielmi, Marcelo Masera, Igor Nai Fovino, and Alberto Trombetta. A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems. *Industrial Informatics, IEEE Transactions on*, vol. 7(2):pp. 179–186, May, 2011.

[Cen10] Centre for the Protection of National Infrastructure and PA Consulting Group. Good Practice Guide – Process Control and SCADA Security. Tech. rep., Centre for the Protection of National Infrastructure, 2010. Online. Accessed on December 10, 2013. Available at: http://www.cpni.gov.uk/documents/publications/2008/2008031-gpg_scada_security_good_practice.pdf?epslanguage=en-gb.

[Che06] Steven Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner, and Alfonso Valdes. Using Model-based Intrusion Detection for SCADA Networks. *Proceedings of the SCADA Security Scientific Symposium*, 2006.

[Chr07] Henrik Christiansson and Eric Luiijf. Creating a European SCADA Security Testbed. *Critical Infrastructure Protection*, pp. 237–247, 2007.

[Com08] Douglas E Comer. *Computer Networks and Internets*. Prentice Hall Press, New Jersey, fifth edn., 2008.

[Cou07] Maurilio Pereira Coutinho, Germano Lambert-Torres, Luiz Eduardo Borges da Silva, Jonas Guedes Borges da Silva, Jose Cabral Neto, E. da Costa Bortoni, and Horst Lazarek. Attack and fault identification in electric power control

systems: An approach to improve the security. In *Proceedings of Power Tech 2007*, pp. 103–107. IEEE Press, 2007.

[Cou08] Maurilio Pereira Coutinho, Germano Lambert-Torres, Luiz Eduardo Borges da Silva, Jonas Guedes Borges da Silva, Jose Cabral Neto, and Horst Lazarek. Improving a methodology to extract rules to identify attacks in power system critical infrastructure: New results. In *Transmission and Distribution Conference and Exposition, 2008.*, pp. 1–6. IEEE, 2008.

[Cou09] M. P. Coutinho, G. Lambert-Torres, L. E. B. da Silva, H. G. Martins, H. Lazarek, and J. C. Neto. Anomaly detection in power system control center critical infrastructures using rough classification algorithm. In *3rd IEEE International Conference on Digital Ecosystems and Technologies, 2009. (DEST '09)*, pp. 733–738. IEEE, jun 2009.

[Cun10] Robert Cunningham, Steven Cheung, Martin Fong, Ulf Lindqvist, David Nicol, Ronald Pawlowski, Eric Robinson, William Sanders, Sankalp Singh, Alfonso Valdes, Bradley Woodworth, and Michael Zhivich. Securing current and future process control systems. *International Federation for Information Processing Digital Library*, vol. 253(1), 2010.

[Dam95] Marc Damashek et al. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, vol. 267(5199):pp. 843–848, 1995.

[DAn06] Salvatore D'Antonio, Francesco Oliviero, and Roberto Setola. High-speed intrusion detection in support of critical infrastructure protection. In *Critical Information Infrastructures Security. First International Workshop, CRITIS 2006*, pp. 222–234. Springer, August 31, 2006.

[Deb00] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion-detection systems. In *Annales des télécommunications*, vol. 55, pp. 361–378. Springer-Verlag, 2000.

[Dem02] C. Demichelis and P. Chimento. IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). Tech. rep., Internet Engineering Task Force, 2002. Online. Accessed on December 10, 2013. Available at: http://tools.ietf.org/rfc/rfc3393.txt.

[Dig13] Digital Bond, Inc. Basecamp, 2013. Online. Accessed on December 10, 2013. Available at: http://www.digitalbond.com/tools/basecamp/.

[Düs10] Patrick Düssel, Christian Gehl, Pavel Laskov, Jens-Uwe Bußer, Christof Störmann, and Jan Kästner. Cyber-Critical Infrastructure Protection Using Real-time Payload-based Anomaly Detection. *Critical Information Infrastructures Security*, vol. 6027:pp. 85–97, 2010.

[Eur04] European Commission. Communication from the Commission to the Council and the European Parliament - Critical Infrastructure Protection in the fight against terrorism. Tech. rep., European Commission, 2004. Online. Accessed on December 10, 2013. Available at: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2004:0702:FIN:EN:PDF.

[Fal11] Nicolas Falliere, Liam O. Murchu, and Eric Chien. W32.Stuxnet Dossier, February 2011 2011. Online. Accessed on December 10, 2013. Available at: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[Fid12] David P Fidler. Tinker, tailor, soldier, duqu: Why cyberespionage is more dangerous than you think. *International Journal of Critical Infrastructure Protection*, vol. 5(1):pp. 28–29, 2012.

[Fla89] Cristian Flaviu. Probabilistic clock synchronization. *Distributed computing*, vol. 3(3):pp. 146–158, 1989.

[Flo10] Alejandro Flores, James Fields, Axton Graham, Jon Hart, Kevin Johnson, Doug Mackie, Sean Muller, Tim Rupp, Christian Svensson, and Max Valdez. Basic Analysis and Security Engine, 2010. Online. Accessed on December 10, 2013. Available at: http://base.secureideas.net.

[Fov09] Igor Nai Fovino, Andrea Carcano, Marcelo Masera and Alberto Trombetta. Design and Implementation of a Secure Modbus Protocol. *Critical Infrastructure Protection III*, pp. 83–96, 2009.

[Fov10] Igor Nai Fovino, Andrea Carcano, Thibault De Lacheze Murel Murel, Alberto Trombetta, and Marcelo Masera. Modbus/dnp3 state-based intrusion detection system. In *Advanced Information Networking and Applications*

(AINA), 2010 24th IEEE International Conference on, pp. 729–736. IEEE, 20-23/04/2010 2010.

[Gal13] Brendan Galloway and Gerhard P. Hancke. Introduction to Industrial Control Networks. *Communications Surveys & Tutorials*, vol. 15(2):pp. 860–880, July 25, 2013.

[Gao10] Wei Gao, T. Morris, B. Reaves, and D. Richey. On SCADA control system command and response injection and intrusion detection. In *eCrime Researchers Summit (eCrime), 2010*, pp. 1–9. IEEE, October 18, 2010.

[Gar11] Iñaki Garitano, Roberto Uribeetxeberria, and Urko Zurutuza. A Review of SCADA Anomaly Detection Systems. In *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO 2011*, vol. 87, pp. 357–366. Springer Berlin Heidelberg, 2011.

[Gar12a] Iñaki Garitano, Christos Siaterlis, Béla Genge, Roberto Uribeetxeberria, and Urko Zurutuza. A method to construct network traffic models for process control systems. In *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*, pp. 1–8. IEEE, September 17 2012.

[Gar12b] Iñaki Garitano, Roberto Uribeetxeberria, and Urko Zurutuza. Método para la construcción de modelos de tráfico de red para la evaluación de la seguridad de sistemas de control de procesos. In *XII Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2012)*. Mondragon Unibertsitatea, September 4, 2012.

[Gas05] Luciano Paschoal Gaspary, Ricardo Nabinger Sanchez, Diego Wentz Antunes, and Edgar Meneghetti. A snmp-based platform for distributed stateful intrusion detection in enterprise networks. *Selected Areas in Communications, IEEE Journal on*, vol. 23(10):pp. 1973–1982, 2005.

[Gen12] Béla Genge, Christos Siaterlis, Igor Nai Fovino, and Marcelo Masera. A Cyber-Physical Experimentation Environment for the Security Analysis of Networked Industrial Control Systems. *Computers & Electrical Engineering*, vol. 38(5):pp. 1146–1161, September, 2012.

[Gon07] Jesus Gonzalez and Mauricio Papa. Passive scanning in modbus networks. In *Critical Infrastructure Protection*, vol. 253, pp. 175–187. Springer, 2007.

[Gus89] Riccardo Gusella and Stefano Zatti. The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD. *Software Engineering, IEEE Transactions on*, vol. 15(7):pp. 847–853, 1989.

[Had10] Dina Hadžiosmanović, Damiano Bolzoni, and Pieter Hartel. MEDUSA: Mining Events to Detect Undesirable uSer Actions in SCADA. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, pp. 500–501. Springer, 2010.

[Hen08] Mariana Hentea. Improving Security for SCADA Control Systems. *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 3(12):pp. 73–86, 2008.

[ICS12] ICS-CERT. ICS-CERT Incident Response Summary (2009-2011). Tech. rep., Industrial Control Systems Cyber Emergency Response Team, 2012. Online. Accessed on December 10, 2013. Available at: http://ics-cert.us-cert.gov/ICS-CERT-Incident-Response-Summary-2009-2011.

[ICS13] ICS-CERT. ICS-CERT Monitor April-June 2013. Tech. rep., Industrial Control Systems Cyber Emergency Response Team, June 27, 2013. Online. Accessed on December 10, 2013. Available at: http://ics-cert.us-cert.gov/monitors/ICS-MM201306.

[IEC04] IEC. 61588-2004 - IEC/IEEE Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (Adoption of IEEE Std 1588-2008). Tech. rep., TC9-Technical Committee on Sensor Technology of the IEEE I & M Society, 2004.

[IEEE02] IEEE. 1588-2002 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. Tech. rep., IEEE Instrumentation and Measurement Society, December 10, 2002.

[Igu06] Vinay M. Igure, Sean A. Laughter, and Ronald D. Williams. Security issues in SCADA networks. *Computers and Security*, vol. 25(7):pp. 498–506, 2006.

[Int11] International Telecommunication Union. Internet protocol data communi-

cation service – IP packet transfer and availability performance parameters. Tech. rep., International Telecommunication Union, 2011. Online. Accessed on December 10, 2013. Available at: https://www.itu.int/rec/dologin_pub.asp?lang=s&id=T-REC-Y.1540-201103-I!!PDF-E&type=items.

[ISO11] ISO and IEC. Iso/iec 27005:2011 information technology - security techniques - information security risk management. Tech. rep., ISO and IEC, May 23, 2011.

[Kal09] Anas Abou El Kalam and Yves Deswarte. Critical Infrastructures Security Modeling, Enforcement and Runtime Checking. In *3rd International Workshop on Critical Information Infrastructures Security (CRITIS 2008)*, vol. 5508 LNCS, pp. 95–108. Springer Verlag, Rome, Italy, 2009.

[Kru05] Ronald L. Krutz. *Securing SCADA Systems*. Hungry Minds Inc, Indianapolis, Indiana, 2005.

[Law08] Kenneth D. Lawrence, Ronald K. Klimberg, and Sheila M. Lawrence. *Fundamentals of Forecasting Using Excel*. Industrial Press Inc., 2008.

[Mah10] Abdun Naser Mahmood, Christopher Leckie, Jiankun Hu, Zahir Tari, and Mohammed Atiquzzaman. Network Traffic Analysis and SCADA Security. *Handbook of Information and Communication Security*, pp. 383–405, 2010.

[Mat09] John Matherly. SHODAN - Computer Search Engine, 2009. Online. Accessed on December 10, 2013. Available at: http://www.shodanhq.com.

[McA11] McAfee Foundstone Professional Services and McAfee Labs. Global Energy Cyberattacks: "Night Dragon". Tech. rep., McAfee, Inc., February 10, 2011. Online. Accessed on December 10, 2013. Available at: http://www.mcafee.com/in/resources/white-papers/wp-global-energy-cyberattacks-night-dragon.pdf.

[McC91] Steven McCanne. The BPF manual page. *Lawrence Berkeley Laboratory, Berkeley, CA*, 1991.

[McC93] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference*, pp. 2–2. USENIX Association, 1993.

[McE10] Thomas McEvoy and Stephen Wolthusen. Trouble brewing: Using observations of invariant behavior to detect malicious agency in distributed control systems. In *Critical Information Infrastructures Security*, vol. 6027, pp. 62–72. Springer Berlin Heidelberg, 2010.

[McG09] Robert Wesley McGrew and Rayford B Vaughn. Discovering vulnerabilities in control system human–machine interface software. *Journal of Systems and Software*, vol. 82(4):pp. 583–589, 2009.

[McM10] Robert McMillan. Siemens: Stuxnet worm hit industrial systems, 14/09/2010 2010. Online. Accessed on December 10, 2013. Available at: http://www.computerworld.com/s/article/print/9185419/Siemens_Stuxnet_worm_hit_industrial_systems?taxonomyName=Network+Security&taxonomyId=142.

[Mil83] David L Mills. Internet Delay Experiments. Tech. rep., Internet Engineering Task Force, 1983. Online. Accessed on December 10, 2013. Available at: http://www.ietf.org/rfc/rfc889.txt.

[Mil85] David L. Mills. Network Time Protocol (NTP). Tech. rep., Internet Engineering Task Force, 1985. Online. Accessed on December 10, 2013. Available at: http://www.ietf.org/rfc/rfc958.txt.

[Mil88] David L. Mills. Network Time Protocol (Version 1) Specification and Implementation. Tech. rep., Internet Engineering Task Force, 1988. Online. Accessed on December 10, 2013. Available at: http://www.ietf.org/rfc/rfc1059.txt.

[Mil89] David L. Mills. Network Time Protocol (Version 2) Specification and Implementation. Tech. rep., Internet Engineering Task Force, 1989. Online. Accessed on December 10, 2013. Available at: http://www.ietf.org/rfc/rfc1119.ps.

[Mil92] David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Tech. rep., Internet Engineering Task Force, 1992. Online. Accessed on December 10, 2013. Available at: http://www.ietf.org/rfc/rfc1305.txt.

[Mil10] David L. Mills, Jim Martin, Jack Burbank, and William Kasch. Network

Time Protocol Version 4: Protocol and Algorithms Specification. Tech. rep., Internet Engineering Task Force, 2010. Online. Accessed on December 10, 2013. Available at: http://www.ietf.org/rfc/rfc5905.txt.

[Mil12] Bill Miller and Dale Rowe. A Survey of SCADA and Critical Infrastructure Incidents. In *Proceedings of the 1st Annual conference on Research in information technology*, pp. 51–56. ACM, 2012.

[Mit13] Robert Mitchell and Ing-Ray Chen. Behavior-Rule Based Intrusion Detection Systems for Safety Critical Smart Grid Applications. *Smart Grid, IEEE Transactions on*, vol. 4(3):pp. 1254–1263, 2013.

[Mit13] Robert Mitchell and Ing-Ray Chen. A Survey of Intrusion Detection Techniques for Cyber Physical Systems. *ACM Computing Surveys (CSUR)*, 2013.

[MP06] PS Motta Pires and Luiz Affonso HG Oliveira. Security Aspects of SCADA and Corporate Network Interconnection: An Overview. In *Dependability of Computer Systems, 2006. DepCos-RELCOMEX'06. International Conference on*, pp. 127–134. IEEE Computer Society, 2006.

[Nat06] National Institute of Standards and Technology. Minimum Security Requirements for Federal Information and Information Systems. Tech. rep., National Institute of Standards and Technology, 2006. Online. Accessed on December 10, 2013. Available at: http://csrc.nist.gov/publications/fips/fips200/FIPS-200-final-march.pdf.

[Nic12] Andrew Nicholson, S Webber, S Dyer, T Patel, and Helge Janicke. SCADA security in the light of Cyber-Warfare. *Computers & Security*, vol. 31(4):pp. 418–436, 2012.

[Oma07] Paul Oman and Matthew Phillips. Intrusion Detection and Event Monitoring in SCADA Networks. In *Critical Infrastructure Protection*, pp. 161–173. Springer, 2007.

[Pér10] Andrés Pérez Garcia and Christos Siaterlis and Marcelo Masera. Studying characteristics of an Emulab testbed for scientifically rigorous experiments. In *Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on*, pp. 215–218. IEEE, 2010.

[Pre10] Upeka Premarathne, Uthpala Premarathnet, and Kithsiri Samarasinghe. Intrusion detection for IEEE 802.11 based industrial automation using possibilistic anomaly detection. In *Wireless And Optical Communications Networks (WOCN), 2010 Seventh International Conference On*, vol. 1, pp. 1–5. IEEE, September 6, 2010.

[Pyt13] Python Software Foundation. Python, 2013. Online. Accessed on December 10, 2013. Available at: http://www.python.org.

[Rab10] Julien Rabatel, Sandra Bringay, and Pascal Poncelet. Fuzzy anomaly detection in monitoring sensor data. *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pp. 1–8, July 18, 2010.

[Rap13] Rapid7. Metasploit, 2013. Online. Accessed on December 10, 2013. Available at: http://www.metasploit.com.

[Ree12] Jason Reeves, Ashwin Ramaswamy, Michael Locasto, Sergey Bratus, and Sean Smith. Intrusion detection for resource-constrained embedded control systems in the power grid. *International Journal of Critical Infrastructure Protection*, vol. 5(2):pp. 74–83, July, 2012.

[Roe13] Martin Roesch and Chris Green. *SNORT Users Manual 2.9.5*. The Snort Project, 2013. Online. Accessed on December 10, 2013. Available at: http://s3.amazonaws.com/snort-org/www/assets/166/snort_manual.pdf.

[Roo08] Tanya Roosta, Dennis K. Nilsson, Ulf Lindqvist, and Alfonso Valdes. An Intrusion Detection System for Wireless Process Control Systems. In *5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008*, pp. 866–872. IEEE, 2008.

[Rru09] Julian Rrushi and Roy Campbell. Detecting cyber attacks on nuclear power plants. In *Critical Infrastructure Protection II*, vol. 290, pp. 41–54. Springer, 2009.

[San12] David E. Sanger. Obama order sped up wave of cyberattacks against Iran. *The New York Times*, June 1, 2012. Online. Accessed on December 10, 2013. Available at: http://www.nytimes.com/2012/06/01/

world/middleeast/obama-ordered-wave-of-cyberattacks-against-iran.html?pagewanted=all&_r=0.

[Sav12] Kevin Savage and Branko Spasojevic. W32.Flamer, June 5, 2012 2012. Online. Accessed on December 10, 2013. Available at: http://www.symantec.com/security_response/writeup.jsp?docid=2012-052811-0308-99.

[Sho11] Ahmed F Shosha, Pavel Gladyshev, Shinn-Shyan Wu, and Chen-Ching Liu. Detecting cyber intrusions in SCADA networks using multi-agent collaboration. In *Intelligent System Application to Power Systems (ISAP), 2011 16th International Conference on*, pp. 1–7. IEEE, September 25, 2011.

[Sia12] Christos Siaterlis, Andres Perez Garcia, and Bela Genge. On the Use of Emulab Testbeds for Scientifically Rigorous Experiments. *IEEE Communications Surveys and Tutorials*, 2012. Accepted.

[Sou13a] Sourcefire, Inc. Protecting control networks. Tech. rep., Sourcefire, Inc., 2013. Online. Accessed on December 10, 2013. Available at: http://www.sourcefire.com/protecting-control-networks-whitepaper.

[Sou13b] Sourcefire, Inc. Snort, 2013. Online. Accessed on December 10, 2013. Available at: http://www.snort.org.

[Sto13] Keith A. Stouffer, Joseph A. Falco, and Karen A. Scarfone. Guide to Industrial Control Systems (ICS) Security. Tech. rep., National Institute of Standards and Technology, May 14, 2013. Online. Accessed on December 10, 2013. Available at: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r1.pdf.

[Sve09a] Nils Svendsen and Stephen Wolthusen. Modeling And Detecting Anomalies In Scada Systems. *Critical Infrastructure Protection II*, pp. 101–113, 2009.

[Sve09b] Nils Svendsen and Stephen Wolthusen. Using physical models for anomaly detection in control systems. In *Critical Infrastructure Protection III*, vol. 311, pp. 139–149. Springer Berlin Heidelberg, 2009.

[Sym11] Symantec Corporation. W32.Duqu The precursor to the next Stuxnet, November 23, 2011 2011. Online. Accessed on December 10, 2013. Available at: http://www.symantec.com/content/en/us/enterprise/media/security_

response/whitepapers/w32_duqu_the_precursor_to_the_next_stuxnet.pdf.

[Tcp13a] Tcpdump/Libpcap. libpcap, 2013. Online. Accessed on December 10, 2013. Available at: http://www.tcpdump.org.

[Tcp13b] Tcpdump/Libpcap. tcpdump, 2013. Online. Accessed on December 10, 2013. Available at: http://www.tcpdump.org.

[Ten13] Tenable Network Security. Nessus vulnerability scanner, 2013. Online. Accessed on December 10, 2013. Available at: http://www.tenable.com/products/nessus.

[The12] The University of Utah. Emulab bibliography, February 28, 2012. Online. Accessed on December 10, 2013. Available at: http://www.emulab.net/expubs.php.

[The13] The MathWorks, Inc. MATLAB - The Language of Technical Computing, 2013. Online. Accessed on December 10, 2013. Available at: http://www.mathworks.com/products/matlab/.

[Tsa05] Chi-Ho Tsang and Sam Kwong. Multi-agent intrusion detection system in industrial network using ant colony clustering approach and unsupervised feature extraction. In *Industrial Technology, 2005. ICIT 2005. IEEE International Conference on*, pp. 51–56. IEEE, December 14, 2005.

[Tur05] Robert J Turk. *Cyber incidents involving control systems.* Idaho National Engineering and Environmental Laboratory, 2005.

[Tur13] Aaron Turner. Tcpreplay, 2013. Online. Accessed on December 10, 2013. Available at: http://tcpreplay.synfin.net.

[Val03] Alfonso Valdes. Detecting novel scans through pattern anomaly detection. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, vol. 1, pp. 140–151. IEEE, 2003. Online. Accessed on December 10, 2013. Available at: http://www.csl.sri.com/papers/valdes-discex03/valdes-discex03.pdf.

[Val09a] Alfonso Valdes and Steven Cheung. Communication Pattern Anomaly De-

tection in Process Control Systems. In *IEEE Conference on Technologies for Homeland Security, 2009. HST'09.*, pp. 22–29. IEEE, April 11, 2009. Online. Accessed on December 10, 2013. Available at: http://www.csl.sri.com/papers/HST09ValdesCheung/commPatternPCS-Valdes-Cheung-HST09.pdf.

[Val09b] Alfonso Valdes and Steven Cheung. Intrusion monitoring in process control systems. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pp. 1–7. IEEE, January 5, 2009.

[Ver08] Jared Verba and Michael Milvich. Idaho National Laboratory Supervisory Control and Data Acquisition Intrusion Detection System (SCADA IDS) In *Technologies for Homeland Security, 2008 IEEE Conference on*, pp. 469–473. Idaho Natl Lab, Idaho Falls, ID 83415 USA., IEEE, 2008.

[Wan11] Wenye Wang, Yi Xu, and Mohit Khanna. A survey on the communication architectures in smart grid. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 55(15):pp. 3604–3629, Octover, 2011.

[Wan13] Wenye Wang and Zhuo Lu. Cyber security in the smart grid: Survey and challenges. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 57(5):pp. 1344–1371, April, 2013.

[Whi02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *ACM SIGOPS Operating Systems Review*, vol. 36(SI):pp. 255–270, 2002.

[Xue07] Feng Xue and Weizhong Yan. Parametric model-based anomaly detection for locomotive subsystems. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pp. 3074–3079. IEEE, August 12, 2007.

[Yan06] Dayu Yang, Alexander Usynin, and J. Wesley Hines. Anomaly-Based Intrusion Detection for SCADA Systems. In *5th Intl. Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies*, pp. 12–16. Citeseer, 12-16 of November, 2006.

[Yan12] Ye Yan, Yi Qian, Hamid Sharif, and David Tipper. A survey on cyber se-

curity for smart grid communications. *Communications Surveys & Tutorials*, vol. 14(4):pp. 998–1010, January 30, 2012.

[Zha11] Difan Zhang, Wei Yu, and Rommie Hardy. A distributed network-sensor based intrusion detection framework in enterprise networks. In *Military Communications Conference, 2011 (MILCOM 2011)*, pp. 1195–1200. IEEE, 2011.