Design a Secure Cyber-Physical Systems

How Models and Security Adaptation Techniques can address CPS's challenges

Ermanno Battista

Department of Electrical Engineering and Information Technology

University of Naples Federico II

Tutor: Prof. Nicola Mazzocca **Coordinator:** Prof. Franco Garofalo
Co-Tutor: Prof. Valentina Casola

Submitted in partial fulfillment of the XXVII Course requirements for the degree of

Doctor of Philosophy

March 2015

Engineering or Technology is the making of things that did not previously exist, whereas science is the discovering of things that have long existed.

– David Billington

MY GREATEST

Dedicated to my Mentors, my Family and my Friends

INSPIRATION

Abstract

Given the pervasiveness and extremely critical nature of activities performed by cyber-physical systems (CPSs), a better integration of security into the core architecture of the system is required in order to design and deploy survivable and secure infrastructures. The major challenges of modeling, designing and securing CPSs arise from the intrinsic heterogeneity, the geographical scale and the uncertainty regarding sensors readings, status and trustworthiness. These fundamental system properties of CPSs require to address security in ways more closely tied to the physical application and through a design that supports the inclusion of security within the application architecture.

This thesis addresses challenges in both design and security of cyber-physical systems and its contribution is threefold. First, it proposes compositional and multiformalism modeling approaches for the **design and validation of large scale monitoring infrastructures**. The proposed approach is intended to integrate and model the interaction between different embedded nodes and between the environment and embedded nodes. The main limitation of state of art solution to modeling and simulating monitoring infrastructure, arise from their specificity: they are able to give a number of details on a particular aspect. Most of them are specifically intended to model a system part: such as network simulators, software simulators and hardware platform simulators. On the other hand, compositional and multiformalism modeling approaches allow the adoption of the best suiting modeling formalism at different levels of system abstraction and provide a concrete scalable mean to perform early performance evaluation and what-if analysis to ease the design process. They have been adopted in this thesis with regard to CPS's monitoring infrastructure modeling and simulation.

Second, it proposes two ways to **advance the software and hardware security for CPSs**

monitoring infrastructures. Todays approaches to secure the monitoring infrastructure are based on lightweight hardening techniques, in order not to stress the limited resources of wireless sensor nodes and this may limit their strength. Moreover, security seems to be approached as a feature that is not considered from the early stages of design and consequently is not intrinsically integrated within the embedded nodes architecture. On the other hand, this thesis proposes mechanisms for software and hardware security that are closely tied to the embedded architecture. As regards the *software security*, in this thesis efforts have been done to make security Adaptation Techniques (AT) applicable for embedded sensor nodes. Mechanisms based on AT consider security in a proactive way; the system is modified over time in order to reduce vulnerability exposure and disrupt attackers reconnaissance efforts. To make it possible to exploit the advantages of AT at the monitor infrastructure level, a novel node's software reconfiguration framework is presented. The feasibility of the proposed mechanism is shown for battery-supplied wireless sensor nodes through the demonstration of how to alter the network topology, exploit software diversity and react to threats. As regards the advances to the *hardware security* of the monitoring infrastructure, the thesis presents a special purpose hardware architecture specifically intended to address CPSs' physical security challenges through the adoption of TPM-based (Trusted Platform Module) architectures. An architecture based on FPGA is proposed and it is shown how it can provide secure primitives to asses the hardware and software integrity, as well as offering cryptographic operation.

Finally, the thesis focuses on **improving the processing infrastructure security**. Efforts aimed at protecting CPSs should not only focus on the monitoring infrastructure even if it seems the weakest point. The processing infrastructure is generally composed of several networked systems. In this field a number of state of art solution are applicable. This thesis investigates the adoption of novel security adaptation techniques in order to overcame the limitation of static hardening techniques that are generally applied to networked systems. The focus will be on disrupting attacks that are intended to identify the CPS processing infrastructure's network architecture. The thesis will address this challenge by looking at security from a control prospective. Again it will exploit adaptation techniques but with the final goal of disrupting reconnaissance attack targeting CPS's back-end infrastructures. The thesis proposes a graph-based algorithmic solution to manipulate attacker's view of processing systems. This formalization is then used to design a deception based defense to defeating operating system and services fingerprinting. Experimental results show that the proposed approach can efficiently and effectively deceive attackers.

Table of contents

List of figures **ix**

List of tables **xii**

Introduction **1**

1 Context and Motivation 1

2 Open Issues 4

3 Thesis Contribution 5

I Modeling Approaches for Monitoring Infrastructures Design **9**

I.1 Introduction and Background **10**

I.2 A compositional modeling framework **13**

I.2.1 A “generic” modelling framework 13

I.2.2 A real system model 16

I.2.3 Evaluation of a Strago S.p.A. Case Study 20

I.3 A multiformalism modeling approach **25**

I.3.1 A multiformalism process 26

I.3.2 Models of WSN nodes and network 27

I.3.3 Experimental Results 31

I.4 Summary **36**

II Design a Secure Monitoring Infrastructure 38

II.1 Introduction and Background 39

II.2 Adaptation Techniques to Secure Embedded Node 43

II.2.1 Adaptation Techniques and Moving Target Defense in embedded sensor nodes . . . 44

II.2.2 Modeling what is reconfigurable in WSN's architecture . . . 46

II.2.3 SIREN Architecture . . . 49

II.2.4 Evaluation of the approach . . . 54

II.3 Advancing WSN Physical Security adopting TPM 58

II.3.1 Trusted Platform Module and its adoption for CPS's monitoring infrastructures . . . 59

II.3.2 Implementing a TPM-enabled node using FPGA . . . 61

II.3.3 Security considerations about the Chain of Trust . . . 64

II.3.4 Performance . . . 66

II.4 Summary 68

III Securing the Processing Infrastructure 70

III.1 Introduction and Background 71

III.2 Manipulating the Attacker's View 75

III.2.1 Attacker Reconnaissance of a target system . . . 76

III.2.2 View Model and Problem Statements . . . 79

III.2.3 An Algorithmic solution to Identify Views . . . 83

III.2.4 Experimental Evaluation . . . 88

III.3 Steering Attackers using Fingerprint Deception 95

III.3.1 Reconnaissance and Fingerprinting . . . 96

III.3.2 Technical Background . . . 97

III.3.3 Fingerprintg Tools and Mechanisms . . . 98

III.3.4 An approach to defeat fingerprinting efforts . . . 104

III.3.5 Solution Implementation . . . 106

III.3.6 Evaluation . . . 110

III.4 Summary 115

Conclusion **117**

References **120**

List of Acronyms **126**

List of figures

Introduction

i.1 Cyber-physical system infrastructures 2

Chapter I.2 A compositional modeling framework

I.2.1 Modelling framework 14
I.2.2 Application fields Subregions 15
I.2.3 Model Architecture 16
I.2.4 SINK_MODEL implementation 18
I.2.5 NODE_MODEL implementation 19
I.2.6 NETWORK_MODEL implementation 20
I.2.7 Percentage of Loss for a tree with depth of 1 21
I.2.8 Percentage of Loss for a tree with depth of 2 22
I.2.9 Percentage of Loss for a tree with depth of 3 22
I.2.10 Unbalanced tree 23
I.2.11 Unbalanced trees results 23

Chapter I.3 A multiformalism modeling approach

I.3.1 The overall multiformalism model 26
I.3.2 TelosB model structure 27
I.3.3 *Temperature & Humidity* sensing model 28
I.3.4 Markovian Agent Class of the Node 29

I.3.5 Parameter exchange between the models. 31
I.3.6 Validation of Sensors 32
I.3.7 Sensors Timing 33
I.3.8 Topology with fixed PDR 35
I.3.9 Validation of Temperature Sensor 35

Chapter II.2 Adaptation Techniques to Secure Embedded Node

II.2.1 Abstraction of Network and Node Configuration 48
II.2.2 Examples of Configuration Switching Functions 49
II.2.3 Node (Security) Configurations 50
II.2.4 SIREN Control-side 52
II.2.5 Mote-side Components 52
II.2.6 Reconfiguration mote's data flow 54
II.2.7 Deluge and SIREN dissemination and reprogramming time 55
II.2.8 SIREN energy consumption 57

Chapter II.3 Advancing WSN Physical Security adopting TPM

II.3.1 Trusted Platform Module 59
II.3.2 Overview of Zynq internal architecture 62
II.3.3 TPM and Chain Of Trust 64
II.3.4 Trusted environment and secure boot 65
II.3.5 Performance Results 66

Chapter III.2 Manipulating the Attacker's View

III.2.1 Control Prospective 76
III.2.2 Attacker's strategy flow chart 77
III.2.3 Running Example and System Architecture 78
III.2.4 Mutated System Architecture 78
III.2.5 Example of Manipulation Primitive 80
III.2.6 Example of View Manipulation Graph[1] 81
III.2.7 TopK-Distance – Processing time vs. Number of hosts 88
III.2.8 TopK-Distance – Processing time vs. Graph size 89
III.2.9 TopK-Distance – Average processing time vs. k 89
III.2.10 TopK-Distance – Approximation Ratio vs. k 90
III.2.11 TopK-Distance – Number of Solutions vs. k 91

III.2.12 TopK-Budget – Processing time vs. Number of hosts 91
III.2.13 TopK-Budget – Processing time vs. Graph size 92
III.2.14 TopK-Budget – Average processing time vs. k 92
III.2.15 TopK-Budget – Approximation Ratio vs. k 93
III.2.16 TopK-Budget – Number of solutions vs. k 93

Chapter III.3 Steering Attackers using Fingerprint Deception

III.3.1 Taxonomy of OS fingerprinting tools 99
III.3.2 SinFP report against a Windows 7 host 101
III.3.3 Packets exchanged during a SinFP scan 101
III.3.4 Nessus OS identification report 104
III.3.5 Manipulation of outgoing traffic to reflect deceptive signatures 105
III.3.6 Netfilter packet handling and hooks 106
III.3.7 Apache Benchmark 110
III.3.8 FTP Transfer Rate (500 MB file) 111
III.3.9 HP Officejet 7200 Printer 113
III.3.10 SinFP Printer Deception 113

List of tables

Chapter I.3 A multiformalism modeling approach

I.3.1 SAN parameters for Temperature and Humidity Sensor 31
I.3.2 Simulation parameters 34

Chapter II.2 Adaptation Techniques to Secure Embedded Node

II.2.1 Memory Footprints 56

Chapter II.3 Advancing WSN Physical Security adopting TPM

II.3.1 Threat coverage matrix using TPM protection mechanisms 60

Chapter III.3 Steering Attackers using Fingerprint Deception

III.3.1 OS-dependent IP and TCP parameters 98
III.3.2 Results of Nessus scans 112
III.3.3 OS Fingerprinting Tools Deception 113

Introduction

1 Context and Motivation

The proliferation of low-powered wireless networks of embedded sensors has promoted the development of new applications at the interface between the real world and its digital manifestation [6]. This advance in technology enabled for pervasive wireless sensor and actuators and modified the way in which monitoring and control systems are deployed. Examples of such systems can be found in areas as diverse as aerospace, automotive, chemical processes, civil infrastructure, power plants, health-care, manufacturing and transportation. **Cyber–physical systems (CPSs)** are integrations of computation and physical processes. Networks of embedded computers monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. Figure i.1 depicts the main decomposition of a CPS architecture. The *physical infrastructure* is monitored and controlled through a wireless/wired sensor and actuators network. The embedded nodes and the network constitute the *monitoring infrastructure*. In their design and deployment, several architectural parameters can be tuned in order to satisfy functional and non functional requirements, such as embedded nodes' hardware architecture and resources, the kind of operating system (real-time or event-driven), the tasks being performed by each node and the way in which they exchange data. The collected information are usually aggregated and sent to the processing infrastructure where networks of general-purpose computers are used to evaluate the information and if necessary take corrective actions.
The design of such systems, therefore, requires understanding the joint dynamics of computers, software, networks, and physical processes. It is this study of joint dynamics that sets this discipline apart. When studying CPS, certain key problems emerge that are rare in

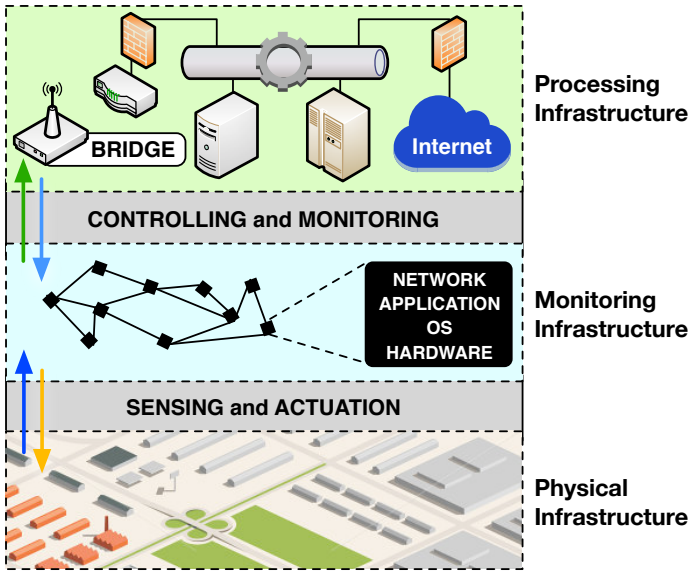


Figure i.1 Cyber-physical system infrastructures

general-purpose computing [29, 59]. For example, real world physical processes are compositions of many parallel (concurrent) processes. Measuring and controlling the dynamics of these processes by orchestrating actions that influence the processes are the main tasks of embedded systems. Those systems sense the world through a number of dedicated sensors and, enforce the mission logic by means of different actuators. Consequently, concurrency is intrinsic in CPSs. Many of the technical challenges in designing and analyzing embedded software stem from the need to bridge an inherently sequential semantics (in computation and software) with an intrinsically concurrent physical world [29].

Despite significant inroads into CPS technology in recent years, we do not yet have a mature science to support systems engineering of high-confidence CPS (in terms of reliability, availability and security), and the consequences are profound. Traditional analysis tools are unable to cope with the full complexity of CPS or adequately predict system behavior. For example, minor events that trip the current electric power grid – an ad hoc system – can escalate with surprising speed into widespread power failures. This scenario exemplifies the lack of appropriate science and technology to conceptualize and design for the deep interdependencies among engineered systems and the natural world. The challenges and opportunities for CPS are thus significant and far-reaching. New relationships between the cyber and physical components require new architectural models that redefine form and function. They integrate the continuous and discrete, compounded by the uncertainty of open environments. Traditional real-time performance guarantees are insufficient for CPS when systems are large and spatially, temporally, or hierarchically distributed in configura-

tions that may rapidly change. With the greater autonomy and cooperation possible with CPS, greater assurances of safety, security, scalability, and reliability are demanded, placing a high premium on open interfaces, modularity, interoperability, and verification [58].

Furthermore, the geographic distribution of the devices opens to both design and security challenges. From a **design point of view, it is required to model and validate heterogeneous large scale systems** in a reasonable time and with the necessary amount of details. From a **security point of view, new kind of threats are present**: the unattended nature allows an attacker to physically capture nodes and learn secret key material, or to intercept or inject messages; the hierarchical nature of sensor networks and their route maintenance protocols permit the attacker to determine where the most critical nodes are placed [6].

Given the pervasiveness and extremely critical nature of activities performed by cyber-physical systems, being able to design and deploy survivable and secure infrastructures requires a better integration of security into the core design of the system. The characteristics of CPSs force us to look at **security in ways more closely tied to the physical application**. Security for such systems need to be considered architecturally, not as a separate "security architecture", but as a secure architecture for the deployment of such applications [59]. The tight coupling between monitoring infrastructure and the physical environment, opens to the existence of communication channels not typically considered which are required to be secured. An attacker does not need to break into control infrastructure to affect such a system; she could cause a coordinated series of physical actions that are sensed and which cause the system to respond in an unexpected manner. Defenders protecting CPSs from this kind of attack, are required to have a deep knowledge of both the system and its physical context, which is not the typical case for general-purpose computer security.

This security-context challenge is even more critical if one considers that cyber-physical systems are often unattended and geographically dispersed. Such distribution increases the difficulty of ensuring *physical security* and physically reset, or reload the software on a compromised device. As stated in [59], security requires a fundamental integration in a way that will permeate the basic design and structure of the application itself. Security solutions must be tied in part to resilience of the application in spite of such compromise, rather than focus solely on preventing compromise of component in the first place. Bearing this in mind, security mechanisms should cope with compromise of some infrastructure entities in order to not escalate to a total loss of security. Rather than providing all-or-nothing guarantees about privacy and security, there is a need for providing **probabilistic guarantees** with respect to compromise [6].

The goal of this thesis is to addresses challenges in both the design and the security of cyber-physical systems. To this aim, the effort made to advance the cyber-physical system state of art will be presented regarding how novel modeling techniques can ease and support

the design process and how security adaptation techniques can be included in both the monitoring and processing CPS's infrastructure.

2 Open Issues

Several attempts have been made towards the definition of proper methodologies and tools supporting the *design of CPSs* in order to avoid resource waste and optimize performance. These large monitoring infrastructures require novel techniques due to their intrinsic heterogeneity and complexity which stresses all existing modeling language and frameworks [29]. A model of a CPS comprises models of physical processes as well as models of the software, computation platforms, and networks. The *feedback loop* between physical processes and computations encompasses sensors, actuators, physical dynamics, computation, software scheduling, and networks with contention and communication delays. **Modeling such systems with reasonable fidelity is challenging**, requiring inclusion of control engineering, software engineering, sensor networks, etc. Additionally, the models typically involve a large number of heterogeneous components, the composition semantics becomes central [29]. Since the design space can easily become very huge, the designer must be provided with methodologies and supporting tools in the choice of best configurations and thus a quick evaluation of different configurations is necessary. Being able to conduct what-if analysis and early measurements is fundamental in order to have confidence in the physical realization of the design and explore solution space. If one only focuses on modelling and simulating the monitoring infrastructure, a number of tools and models are available. Most of them have the goal to simulate with high details a specific system part or algorithm (such as network simulators for medium contention or micro-controllers power consumption models). A state of art CPS model or simulator capable of integrating and composing all the interdependent system parts is still not prominent. One of this thesis contribution will be the proposal of compositional and multiformalism models to design monitoring infrastructures.

Focusing on the security design other open issues arise. Existing literature has proposed the use of computationally inexpensive cryptographic techniques to handle message confidentiality and authenticity in monitoring infrastructures [6] based on wireless embedded nodes. The difficulty of ensuring confidentiality and authenticity is not, however, due solely to the energy constraints imposed on sensors. Todays approaches to security are based on *reactive mechanisms*: after a threat detection a corrective action is performed. Detection systems are useful in many cases but false alarms and missed detections are impossible to avoid because identifying malicious logic is an undecidable problem [16]. Indeed, security is one of the main open challenges to face; available security solutions are not able to cope with new attack scenarios and ***proactive* measures to protect embedded nodes are needed**.

Techniques aimed at continuously changing a system configuration, recently referred to as Moving Target Defense (MTD) or Adaptation Techniques (AT), are emerging to improve the security level provided by the system but their feasibility in resource constrained environment should be studied and evaluated. By applying these techniques to embedded nodes they will be able to show a different and non predictable facade to attackers in terms of different OS, different software modules, different communication protocols and security mechanisms. Reconfiguration mechanisms for wireless sensor networks were primary conceived to reconfigure a network and restart it after a failure or a design change, an open challenge is to analyze their suitability for security purposes.

Still focusing on monitoring infrastructure security, **assessing the hardware and software integrity of its elements is a challenging issue**. The sensor network is comprised of many small computing devices, each of which is subject to physical capture. Any cryptosystem must therefore tolerate the compromise of sensors and their keys. However, the compromise of some nodes need not result in a total loss of security. Rather than providing all-or-nothing guarantees about privacy or security, there is a need for providing probabilistic guarantees with respect to compromise [6]. Physical security should be considered at the beginning of embedded devices development in order to harden the architecture of embedded nodes. New hardware and software solutions are needed to establish distributed trust and perform cryptographic operations in a trustworthy manner.

Efforts aimed at improving security of cyber-physical systems should not focus only on the monitoring infrastructure. Even though the monitoring infrastructure seems the weakest point due to not so mature technological solutions, **advances are still to do in the protection of the processing infrastructure**. The system used in the control and processing of CPSs are similar, in their form, to general-purpose computing architectures. When system configurations are static, attackers will always be able, given enough time, to acquire accurate knowledge about the target system, which enables them to engineer effective attacks. To address this problem adaptive techniques should be enforced to dynamically change some aspects of a system's configuration in order to **introduce uncertainty for the attacker**. Efficient and effective ways to steer attackers away from critical resources and disrupt attackers reconnaissance efforts are still an open challenge.

3 Thesis Contribution

This dissertation works on three directions to address design and security challenges in the context of cyber-physical systems. More specifically, this thesis contributes to the state-of-the-art on CPS by

- **Defining modelling approaches to designing monitoring infrastructures:**
  A solution to the challenge of modelling and simulating large scale monitoring in-

frastructures is proposed through the adoption of compositional and multiformal models. Their ability to scale and to provide the required amount of details is shown and the advantage for designer is discussed in terms of how them allow to conduct what-if analysis and early measurements in order to have confidence in the physical realization of the design and explore solution space.

- **Improving hardware and software security of monitoring infrastructures:** Most of the efforts for security of the sensing networks have been focused on hardening the communication infrastructure, this is not enough. The thesis proposes the adoption of novel security adaptation techniques to gain advantage of *proactive* security mechanisms. These techniques have been proved effective in other context of information technology but their adoption in a constrained resource domain, such as embedded sensor nodes, have not been exhaustively investigated. The proposal and implementation of a reconfiguration framework, specifically designed for embedded nodes, is here presented as well as results that proves its applicability and efficacy. Moreover, the thesis advances the CPS's physical security state of art the by presenting a special purpose hardware architecture through the adoption of TPM-based (Trusted Platform Module) architectures. An architecture based on FPGA is proposed and it is shown how it can provide secure primitives to asses the hardware and software integrity, as well as offering cryptographic operation allowing the implementation of trusted infrastructures.
- **Proposing novel Adaptation Techniques to securing the processing infrastructure:** Even though the monitoring infrastructure seems the weakest point due to not so mature technological solutions, advances are still to do in the protection of the processing infrastructure. The thesis addresses this challenge by analyzing how adaptive techniques can be enforced to dynamically change aspects of a networked-system's configuration in order to introduce uncertainty for the attacker. Adaptation Techniques can offer defenses to different kind of attacks. The thesis focuses on attacks aimed at discovering the processing infrastructure architecture and disrupting reconnaissance attack targeting CPS' back-end infrastructures. The thesis proposes a graph-based algorithmic solution to manipulate attacker's view of processing systems. This formalization is then used to design and implement a deception based defense to defeating operating system and services fingerprinting.

This thesis includes materials from the following research papers, already published in peer-reviewed conferences and journals or submitted for review:

- Valentina Casola, Ermanno Battista, Stefano Marrone, Roberto Nardone, Nicola Mazzocca **"A Compositional Modelling Approach for Large Sensor Networks Design"** *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2013)* - Year: 2013

- Ermanno Battista, Valentina Casola, Stefano Marrone, Nicola Mazzocca, Roberto Nardone, Valeria Vittorini **"An integrated lifetime and network quality model of large WSNs"** *IEEE 2013 International Workshop on Measurements and Networking (IEEE M&N 2013)* - Year: 2013
- Ermanno Battista, Valentina Casola, Antonino Mazzeo, Nicola Mazzocca **"SIREN: a feasible moving target defence framework for securing resource-constrained embedded nodes"** *IJCCBS - International Journal of Critical Computer-Based Systems, Vol. 4, No. 4, pp.374–392* - Year: 2013
- Massimiliano Albanese, Ermanno Battista, Sushil Jajodia, Valentina Casola **"A Graph-Based Approach to Deceive Attackers by Virtualizing a System's Attack Surface"** *Communications and Network Security (CNS), 2014 International Conference on* - Year 2014
- Ermanno Battista, Mario Barbareschi, Antonino Mazzeo, Sridhar Venkatesan **"Advancing WSN Physical Security Adopting TPM-based Architectures"** *IICPS 2014, colocated with the 15th IEEE International Conference on Information Reuse and Integration* - Year: 2014
- Massimiliano Albanese, Ermanno Battista, Sushil Jajodia **"A Deception Based Approach for Defeating OS and Service Fingerprinting"** *Under Review: Journal of Computer Networks, ComNet (Elsevier)* - Year: 2014

The thesis is organized as follows. Part I address challenges relative to the design of monitoring infrastructures. After introducing the limitation of current modeling and simulation solutions, a compositional modelling approach is presented in Chapter I.2, advantages and achievable results are presented. The compositional model is then refined and extended, in Chapter I.3, through the definition of a more sophisticated multiformalism composition to overcome limitations, proper of the CPS domain, in the modeling of interaction between sensing nodes and network.

Part II focuses on the protection of the monitoring infrastructure. After introducing current approaches to securing monitoring infrastructures, Chapter II.2 shows the applicability of Moving Target Defense and Adaptation Techniques to improve the software security of embedded nodes deployed within the sensing infrastructure. On the other hand, Chapter II.3 focuses on how to improve the hardware and physical node security through the adoption of novel Trusted Platform Module (TPM). An FPGA TPM-based architecture is here discussed.

The thesis moves to the protection of processing infrastructures in Part III. After introducing reconnaissance attacks targeting CPS back-end infrastructure a novel control prospective is used to propose a graph-based approach to manipulate attacker view and their perception of a system's attack surface. An attack model is presented and will be used as reference through the following chapters. In Chapter III.2 the proposed formalization

is discussed through the use of a motivating example. In Chapter III.3 the formalization is then used to propose and approach to defeat attackers fingerprinting effort through deception. To defeat operating systems and services fingerprinting the system outgoing traffic is manipulated in order to resemble traffic generated by a host with a different configuration. Experimental results of the implementation will show its efficiency and effectiveness.
The dissertation concludes with final remarks, the indication of the lesson learned and future research directions.

PART I

Modeling Approaches for Monitoring Infrastructures Design

CHAPTER I.1

Introduction and Background

The availability of small low-cost devices has allowed the widespread diffusion of sensor networks as mean to control large areas in a very high number of applications and led to the definition of the foundation of a CPS infrastructure: the physical sensing network. Their adoption domain, and where the interests of both industry and academia usually meet, can space from monitoring of large structures and areas as in civil engineering, marine or agricultural monitoring and even power grids [2]. In such cases the dimension of the *monitoring infrastructure* can be huge in both number of the nodes and distance between nodes themselves. Such feature opens for several problems, one of the most industry-sensitive is the definition of proper design methodologies able to define physical and technological parameters in order to fulfill both functional and non-functional requirements. To have a deep characterization of a CPS design, it is required to move beyond the classical fundamental models of computation and physics. CPSs require new models and theories that unify perspectives, capable of expressing the interacting dynamics of the computational and physical components of a system in a dynamic environment. A unified science would support composition, bridge the computational versus physical notions of time and space, cope with uncertainty, and enable cyber-physical systems to inter-operate and evolve [58].

The CPS design space can easily become hard to explore due to the number of variable involved and their dependencies. Only limiting to the possible network topologies, a slight difference in the network configuration can determine a great variation of non-functional requirements such as performance, security and dependability [2, 18, 19, 67].
Moreover, designers ask for a way to evaluate different node distributions as they influence

network connectivity and sensing coverage. They are interested in evaluating network lifetime and in general operational reliability. Once they defined the nodes dislocation, the network needs to be tuned in order to load balance and efficiently allocate energy resources. For this purpose, it is useful to simulate and analyse network communication, identify critical paths and measure performances in terms of latencies and packet loss rates. If one of these parameters does not satisfy a specification, the designer can explore a different solution for example adding nodes/base-stations, changing the network topology or varying the battery allocation.

To overcome this intrinsic complexity, the designer must be provided with methodologies and supporting tools in the choice of best configurations and thus a quick evaluation of different configurations is necessary. Traditional techniques rely on network simulators: in recent years surveys about sensor network simulators have been published (see [3, 48] in the field of Wireless Sensor Networks - WSNs). Several simulators have been compared detecting three main categories: *architecture specific* , *architecture independent* and *problem specific* solutions. The first approach can be very effective if the designer knows a-priori the specific sensor node to use since such simulators are customized to a specific architecture (examples are TOSSIM [72] and JProwler [76]). The second kind of simulators is architecture generic since they aim at being easily customizable according to the specific needs (Castalia [60] and COOJA [24]) while such simulators seem to suffer from the scalability and performance point of view. In the third category, they belong all the simulators that can be customized to a specific application or to a specific environmental condition.

On the other hand, model-based approaches have been proven to be very effective in several contexts: the most important benefit in the introduction of these approaches is the **capability to have early measures in order to tune the design and explore solutions space**. This capability derives from the higher level abstraction of the real system that model-based approaches have. Petri Nets have been used in energy consumption evaluation in [70] and in [26] with a Fluid Stochastic Petri Nets model; authors in [15] focus on performance evaluation of protocols while the problem of finding an optimal solution in the design of a WSN is addressed in [14] where both a Markov Decision Well-Formed Petri Net and a Stochastic Activity Network (SAN) models are used. Hybrid approaches, based both on traditional simulators and on formal modeling, are also present in the literature [30]. Another formalism that has been successfully used in WSN modeling is: Markovian Agents Models (MAMs) as in [37].

Notwithstanding, in order to make formal modelling more industry appealing, these approaches should not only have comfortable high level modelling languages but they should also allow the customization of formal models according to the specific architecture and the composition of several submodels into a larger one. This last characteristic enables (sub)model reuse allowing the construction of a library of sub-models that can be easily

composed to evaluate design variants and also enable the evaluation of heterogeneous networks [20].

In this thesis, and in publications [11, 13], it is shown **how compositional and multi-formalism modelling approaches are suitable for easing the design of CPS's monitoring infrastructures**. Multiformalism is a modeling techniques dealing with the integration/-composition between sub-models expressed by different formalisms, also based on different modeling paradigms ([28, 50, 74, 77]). The *divide-and-conquer* approach is applied at analysis level as it will be better explained in the next Chapters. First, the model architecture and the composition of relevant sub-models is shown. After, showing the decomposition and validating the sub-model interfaces, it will be shown how to exploit the advantages of using a multiformalism approach, that is choosing the best suiting formalism at different levels of abstraction: (i) one can use a low-level formalism able to capture and model architectural and behavioral features when modeling at node level and (ii) use a formalism that may express complex topology and relations among nodes when focusing on large networks. The proposed approach exploits multiformalism by means of SAN as a "low-level" formalism and MAMs to combine simplified models of single nodes at network level. These formalisms also best suit the necessity of having efficient analysis process since MAMs allow fast analysis of large model by means of numerical solution of ordinary differential equations [37].

CHAPTER I.2

A compositional modeling framework

This chapter presents a modelling methodology for the construction of formal models in the design of sensor networks. The approach has been published in [13] and is here presented by introducing a **"generic" reference model** of a monitoring infrastructure (sensor network): *such model is necessary in order to define model components and their interfaces.* The generic model is then implemented by defining "concrete" SAN sub-models conforming to the defined sub-model interfaces. In particular the model is intended to support design phase by providing a valid tool to define both topology and sensor nodes features and parameter configuration in order to have an efficient, reliable and cost effective monitoring network. In fact, the main requirements whose fulfillment is aimed to be supported, are related to some configuration and deployment choices that should be taken during the design phase; i.e:

- maximum admissible sampling frequency;
- minimum number of nodes needed to cover a large area;
- nodes configuration and network topology;
- node mission time (according to energy capabilities and maintenance policies);
- communication quality indexes (such as Packet Error Rate).

I.2.1 A "generic" modelling framework

This Section describes the modelling approach to support the sensor network design, defining a "generic" modelling framework for the analysis of sensor networks. The approach

exploits the usage of formal models, allowing to create the overall model of the system under analysis in a well-structured way. A component-based view of the system is applied through the *divide-et-impera* technique: the overall system model is decomposed recursively into sub-models until reaching an atomic sub-model. The effectiveness of this approach has been showed in [55] where a "generic" modelling framework has been defined, and then implemented into a "concrete" model exploiting the flexibility of the SAN formalism.

The analysis of a monitoring infrastructure requires to consider the entire application field, modelling four different aspects: *Nodes*, *Network*, *Faults*, *Phenomenon*. This decomposition has allowed the design of a four-module framework, sufficiently general to model any sensor network (both wired and wireless). The modelling framework has been developed following the approach used in [64] and is represented in Figure I.2.1.

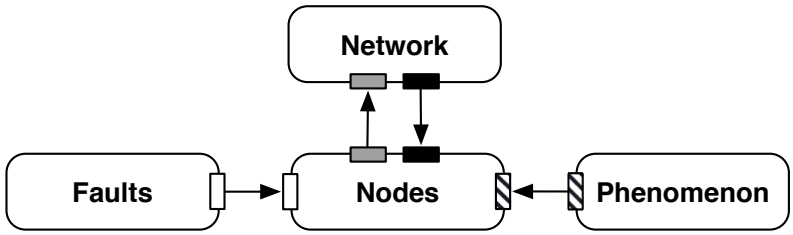


Figure I.2.1 Modelling framework

Each module shares some information about nodes (such as node position, communications ranges, etc.) and communicates with others through defined interfaces, that are described in next Subsections. As already said, the proposed framework has been intended to help in the design of real world sensor networks, for this reason one need to model the application and environmental field where nodes are deployed and different phenomena are observed. The field is structured in a 3D-grid (Figure I.2.2) and the application field is structured in a set of *subregion*s - obtained starting from the axis origin (*O*) with sufficiently small steps of Δ_x, Δ_y and Δ_z. Each subregion is indexed through its coordinates on the three axis (*x*, *y*, *z*), hence the subregion $SR_{i,j,k}$ is the small portion of the application field which is located at coordinates i, j, k respectively on the $x, y, z-axis$. The subregions can be considered independent from the application field, in the sense that a particular subregion is a spatial area which can include terrain, underground, water, etc. The introduced modeling approximation is proportional to the step lengths Δ_x, Δ_y and Δ_z. In next paragraphs details about the four modules are given, a description of the framework then follows.

Phenomenon: This module is dedicated to represent the evolution of all physical phenomena of interest, within the application field. Wide fields could be modeled with different stochastic processes; an *area* is defined as a set of subregions characterized by the same stochastic process. For this reason, each *area* and each phenomenon will require the devel-

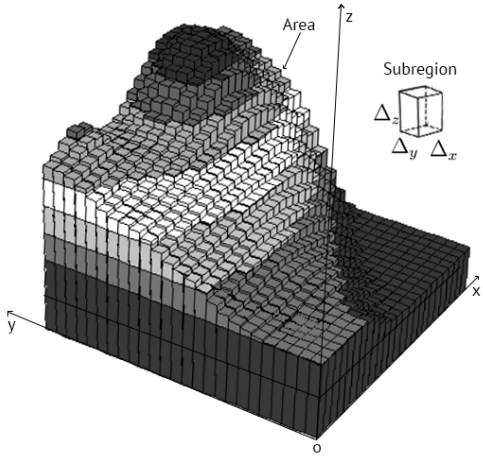


Figure I.2.2 Application fields Subregions

opment of a specific formal model representing the phenomenon evolution in that spatial area. The *Phenomenon* module produces, as output, the evolution of physical phenomena over time so that the *Nodes* module (representing sensing network nodes) can take it as input: hence sensor nodes deployed in the same area will read the same input data value.

Faults: This module deals with the modeling of faults affecting nodes. In particular, dependability models can be here integrated. In a sensor network, nodes are failure prone and can work in degraded modes: this module represents the evolution between nominal and degraded modes according to component failure rate. The *Faults* module produces in output information about working modes of all nodes; this output is read by the *Nodes* module, which has to consider received information for node operations modeling.

Network: This module is responsible for the implementation of network communications. Starting from information about nodes, this module realizes message transfers from sender to recipient nodes. It models message collisions and acknowledgments, if needed, as well as channel noises. The features here modeled are not included in the node module for a simpler modeling and understanding. Furthermore, this module is required to cope with broadcast and dynamical routing strategies implemented by nodes. The *Network* module takes in input, from *Nodes* module, messages from sender nodes and produces, in output, messages for recipient nodes.

Nodes: This module allows to represent physical nodes of the network (i.e. sensing nodes, forwarders and sinks). Each node receives messages from the network and, according to header and body, reacts with proper actions: it can forward the message to another node, it can start/stop sampling operations, it can send stored samples to the sink node, etc. Obviously this module interacts with all others. In particular, node performance is affected by working mode information received by the *Faults* module (e.g. a node in nominal operation

mode, i.e. not affected by faults, requires about 1 ms to send a message while, in case of a fault, it can require even 2.5 ms); node performance can be affected by the value sampled, too[1].

I.2.2 A real system model

This Section analyses performances of a real sensor network scenario used by an Italian company, the STRAGO SpA, to monitor mass-movement. Specification and parameters gathered from one of their operative installations with about 50 nodes have been used.The network is built with: (i) a control entity called sink, (ii) forwarder nodes capable of forwarding messages and (iii) sampling nodes which actually measure physical phenomena. This type of WSNs has a simplified network management and centralized control. For instance, routing is managed by storing the full path into each message header. Moreover, the access to the channel is not bound and eventual "retry and retransmissions" are handled at application level and absent by default. Usually, a sampling node computes sampling operations at a given frequency, properly obtained by the sink within a *start sampling* message. Then, the node collects twenty-four samples and sends them back to the sink into a single reply message (this parameter, named *aggregation level*, can be changed and it is usually chosen according to the monitoring application).

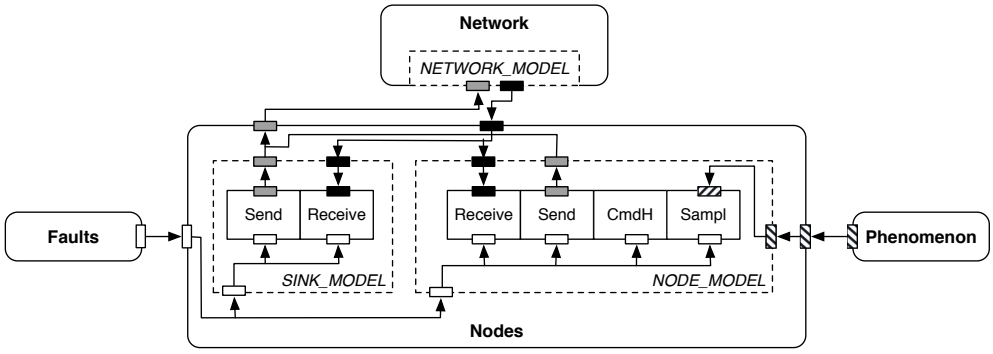


Figure I.2.3 Model Architecture

This system is modelled using the framework defined in the previous Section; at this aim different sub-models have been implemented, each of them is encapsulated in one framework module and has a set of well-defined interfaces that properly match module interfaces. In detail, Figure I.2.3 depicts the sub-models which need to be developed for

[1] Some power-saving mechanisms can change the node's sampling frequency according to the variance of the phenomenon under monitoring.

this system, it gives sub-models location inside the modelling framework and interface connections (with different colors).
The *Nodes* module requires the realization of two models, one contemplating sink nodes and the other sampling and forwarder nodes: the *sink_model* and the *node_model*. The *sink_model* has been structured in two parts, one dealing with sending and the other with receiving operations. The *node_model* has been structured into four portions: besides the send and receive sub-models, sampling operations and command handling shall be modelled. For simplicity of discussion, in next experiments no models are used for the *Faults* and *Phenomenon* modules. The *Nodes* module, even if it implements the interfaces towards these two modules, reads nominal and constant values. At last, the *Network* module requires the development of a *network_model*, which implement message passing between nodes. The implementation of the system has been realized using the Stochastic Activity Network (SAN) formalism [69]. This choice is due to the great flexibility and modeling power given by SAN models; furthermore, its adoption has allowed to **directly map the designed component-based architecture into "atomic" models**.

I.2.2.1 A SAN-based implementation

The implementation of the system has been realized using the Stochastic Activity Network (SAN) formalism [69]. This choice is due to the great flexibility and modeling power given by SAN models; furthermore, its adoption has allowed to directly map the designed component-based architecture into "atomic" models. In the following, each atomic model is described in details while all component interfaces have been realized sharing extended places; these places contain complex data structures where atomic models read and write shared information.

SINK_MODEL: This atomic model represents a base-station in the WSN infrastructure (Figure I.2.4). It is responsible for initiating all nodes sampling activities and for receiving measured samples.These two activities (sending and receiving messages) have been modeled with two SAN sub-models, respectively *Send* and *Receive*. Basically, the sink node creates a *start sampling* message for each node in the network which includes information about: (i) the sampling frequency (f_S) and (ii) a routing path according to the underlying network topology. Details about the generation of start messages are omitted for the sake of brevity. Once the start message is built, it is ready for being delivered. The *Send* sub-model is reported in Figure I.2.4-A. The input gate *SendMessage* is activated on new message generation. The *SendDelay* timed activity fires after the actual transfer time, it is computed by dividing the real message size by the transmission rate. The output gate *OG* updates the number of pending outgoing messages (incrementing the tokens in *numOutMess* place) and updates the information in *outMess* to pass the new message's identifier to the network

model. Then the sink is immediately ready to send a new message. The couple *outMess* and *numOutMess* places implement the outgoing interface between the *sink_model* and *net_model*. In Figure I.2.4, interfaces are represented according to the colors adopted in the general model architecture. In Figure I.2.4-B the *Receive* sub-model is reported. The

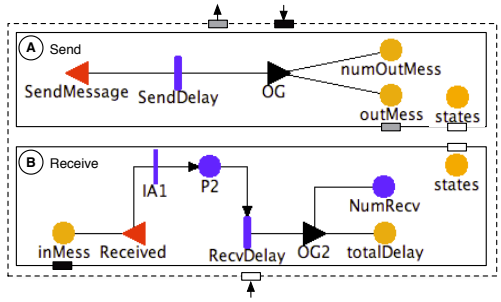


Figure I.2.4 SINK_MODEL implementation

Received input gate is active if a valid message is present in the node's input interface (*inMess* extended place). The *inMess* extended place implements the ingoing interface between *sink_model* and *network_model*. The *Received* input function updates the delay time of the *RecvDelay* timed activity and for this reason the "*IA1-P2*" sequence is inserted before *RecvDelay*. The output gate *OG2* reads the message body, computes the message transmission delay and updates the number of received messages by incrementing *NumRecv*'s tokens.

NODE_MODEL: This atomic model represents a sampling or a forwarder node in the WSN infrastructure. When a node receives a *start sampling* message, it enables the sampling process and, according to the aggregation level, samples are sent back using the inverse path. Moreover, it is able to forward messages in order to participate to the multi-hop routing protocol. The *Receive* sub-model structure is equal to the Sink's one, it is in Figure I.2.5-A. The only difference is in the *OG2* output gate where the message header is checked to distinguish between messages to forward and messages to process. If the message needs to be processed, it is checked if it is a start message and, in this case, the *isReceivedSTART* place is properly set to enable the sampling process. On the other hand, if the message has to be forwarded, its identifier is moved in the *msgToForward* extended place. If the node is already forwarding a message and another one arrives, the latter is discarded. In this sub-model, in case of power consumption analysis, on each message reception, the battery level can be properly decremented according to the consumption rate.
In Figure I.2.5-B the command handler (CmdH) sub-model is represented, it handles start messages. The *setS1* activity is enabled by *ReceivedSTART* when *isReceivedSTART* contains a token (see Figure I.2.5-A). Once enabled, the *set_S1* output gate retrieves information from

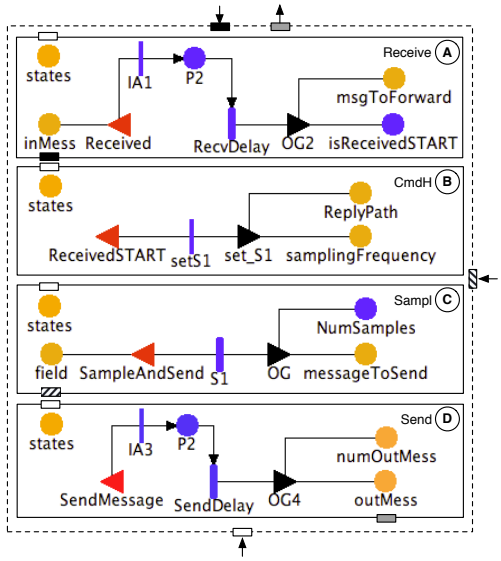


Figure I.2.5 NODE_MODEL implementation

the start message about the sampling frequency (*samplingFrequency* place is updated) and the reply path is stored in the *ReplyPath* extended place.

In Figure I.2.5-C the sampling sub-model is reported. Once a start message is handled the *SampleAndSend* input gate is activated and the timed activity *S1* can fire according to the sampling frequency. The output gate *OG* counts the number of samples (*NumSamples* place) and, when sending threshold is reached, it generates a message for sink. This message contains an array of samples and its size is set to the actual number of bits including message header, body and tail. The identifier of the message generated is stored in *messageToSend* extended place in order to exchange data with the sending sub-model.

In Figure I.2.5-D is reported the SAN model to send/forward a message. The *SendMessage* input gate is active when a message has to be forwarded (*msgToForward* extended place of Figure I.2.5-A) or a new message is being generated by the node itself (*messageToSend* extended place of Figure I.2.5-C). According to the actual operation, the transmission time is set by modifying the firing rate of *SendDelay* timed activity. Then the *OG4* output gate decrement the battery level, increments the number of pending outgoing messages (*numOutMess*) and updates the identifier in *outMess* to signal the network model of which message this node is trying to send. Even in these figures, the same convention on interfaces have been adopted.

NETWORK_MODEL: This atomic model implements the simplified network model proposed in the system. Its basic feature is to fetch outgoing messages and to deliver them to the input interface of the right node, as illustrated in Figure I.2.6. It uses global variables,

shared among different atomic model, to count the number of pending outgoing messages. It implements the transmission by means of the *inMess* and *outMess* interfaces. In details,

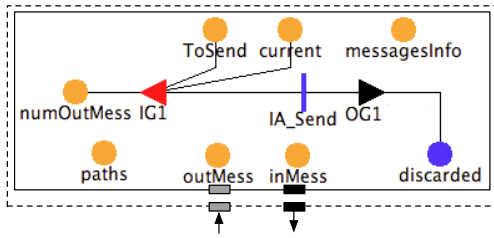


Figure I.2.6 NETWORK_MODEL implementation

inMess and *outMess* contain pointers to the current message in the input/output interface of each node in the network. In the same way *paths* and *messagesInfo* are arrays used to respectively store the route of each message and the message's body itself. The input gate *IG1* fires if the mark of *numOutMess* differs from zero. The *IG1* input function polls the outgoing interfaces to find the message to be processed and prepares for the operations that are performed by *OG1*. So, it saves the current hop in *current* extended place and the identifier of the message currently processed in the *ToSend* extended place. The output gate *OG1*, knowing each message's path and the current routing hop, finds the next hop. At this point if the ingoing interface of the addressee is vacant, the message is delivered by updating pointers in *inMess*. Otherwise the message is marked as collided and the counter of lost messages (*discarded* place) is incremented.

I.2.3 Evaluation of a Strago S.p.A. Case Study

In this Section experimental results for the analysis and design of a sensor network are reported. The performance of different network deployments are evaluated in terms of number of nodes, connection topologies and sampling frequencies (f_S). Different scenarios and network topologies are illustrated and design principles that can derive from the analysis done are discussed.

I.2.3.1 Balanced tree with depth of 1

In the first experiment, a simple topology where each node is directly connected to the sink, is considered. To perform the experimentation, different configurations have been evaluated by changing the number of total connected nodes and the sampling frequency. The model is initialized with five nodes and a sink, then the number of nodes is increased with a step of five nodes per time. Figure I.2.7 outlines the loss trends against number of nodes with

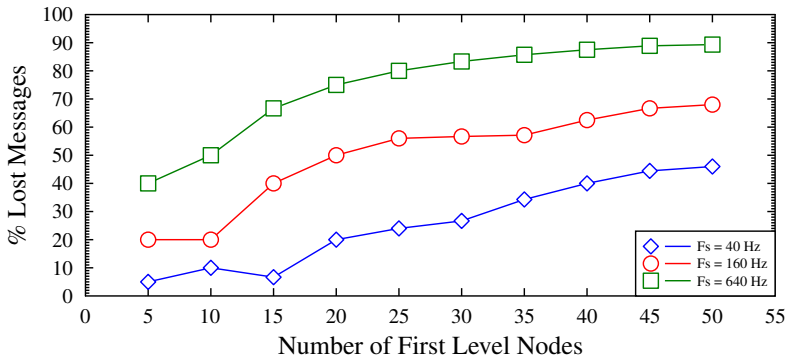


Figure I.2.7 Percentage of Loss for a tree with depth of 1

different f_S. As one can notice the absence of buffering and collision avoidance to access the channel leads to high loss rate both with the increase of nodes and with the increase in the frequency of sending. We evaluated the number of messages received by the sink and discovered that it saturates around 4100 messages (98400 samples); as a consequence with f_S equals to 640Hz the number of received messages leveled off with more than ten nodes, explaining the high loss rate measured for this curve.

I.2.3.2 Balanced tree with depth of 2

A two hops balanced tree topology is here considered. Each leaf is a node and every five nodes are connected to a forwarder. Each forwarder is connected to the sink. Figure I.2.8 reports loss rates with the same computational load of the first experiment (maximum of 50 leafs - second level - nodes). Initially, the network consists of five nodes, one forwarder and the sink. Then five nodes per time as long as their forwarder are added. In this way, the last x-axis point (50 nodes) refers to a topology of ten forwarders and fifty leaf nodes. Also in this case a saturation effect affects the network starting from ten nodes sampling at 640Hz.

Comparing the loss trends of Figures I.2.7 and I.2.8, one can discovered that the 1-hop and 2-hop topologies have almost the same performance in many conditions. In particular, with a 2-hop topology and for low-frequencies, some improvements are visible in networks with more than 30 nodes and the lost message percentage decrements of about 13,5%. With medium and higher frequencies there is a even less significant improvement: with medium frequencies and 25 nodes, the maximum gain (in terms of difference in lost message percentage) is about 10%, while with high frequencies and with 40 nodes the maximum gain obtained is only 5%.

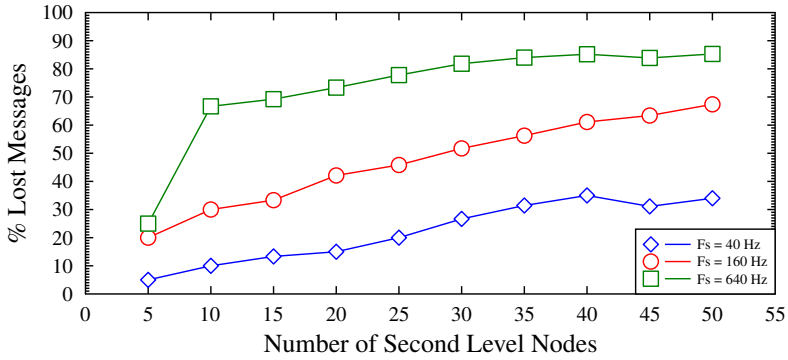


Figure I.2.8 Percentage of Loss for a tree with depth of 2

I.2.3.3 Balanced tree with depth of 3

In this experiment, the depth of the topology tree is increased. This topology has two forwarding levels organized as follows. Each 5 leaf nodes are connected to a 2nd level forwarder. Two 2nd level forwarders are connected to a 1st level forwarder and every one of them is connected to the sink. As before, the start is a topology of 5 leaf nodes and then arrived to 50 leaf nodes by adding 5 leaf nodes per time with corresponding forwarders. This topology works generally better than the others as even if the delay of receiving a message is higher, the sink node starts to saturate after 35 nodes with the maximum sampling frequency.

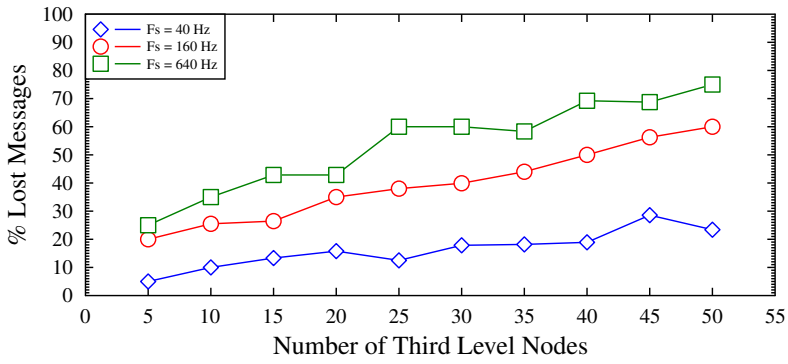


Figure I.2.9 Percentage of Loss for a tree with depth of 3

By comparing the loss trends of Figures I.2.7, I.2.8 and I.2.9, one can see that the performance of a 3-hop topology are better of both previous cases, for every f_S. In particular, at low and medium frequencies there is a lost message percentage gain of respectively 22% (with 50 nodes) and 18% (with 25 nodes). The results are even better for higher performances:

the 3-hop is 25% better than the 2-hop topology with 35 nodes, and the 3-hop is 32% better than the 1-hop topology with 20 nodes.

I.2.3.4 Unbalanced tree

For a better assessment of the effects of the depth of tree, in the last experiment the fifty node topology is gradually unbalanced. The aim of this evaluation is to measure the variations of the loss rate and the delay increment.

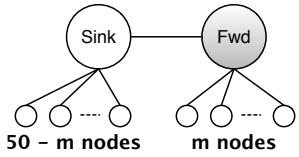
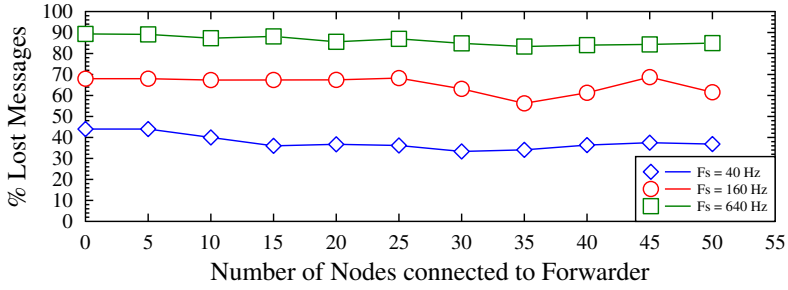
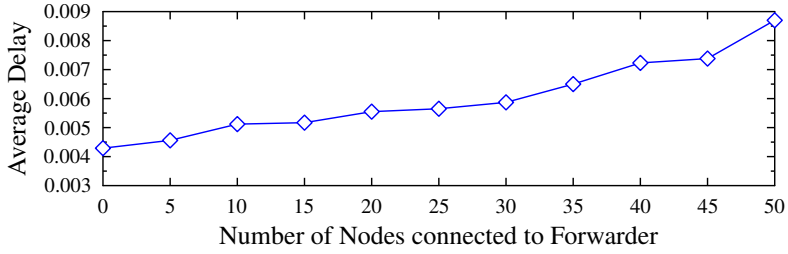


Figure I.2.10 Unbalanced tree

Referring to Figure I.2.10, for the initial condition all the nodes are directly connected to the sink ($m = 0$, this is the case of one hop communication). Then five nodes per time ($m_{i+1} = m_i + 5$) move from 1-hop, to a 2-hop connection through the forwarder.



(a) Loss for increasingly unbalanced trees



(b) Delay for increasingly unbalanced trees

Figure I.2.11 Unbalanced trees results

Packet loss trends are reported in Figure I.2.11a, note that on the x-axis there are the nodes directly connected to the forwarder and not the leaf nodes as in the previous plots so, for N=0 this is the case of the 1-hop topology and for N=50 the case of a 2-hop topology but with 50 leafs connected to the only forwarder. The important result is that for every f_S the lost message trends have a slight negative slope with values in agreement with those for 1-hop and 2-hop, putting in evidence that a second level improves the general quality of the network but degrades the response time.
Finally, the plot in Figure I.2.11b traces the average delay to send a message to the sink. It went up whit an almost constant rate as long as the number of 2-hop connected nodes becomes predominant. The last point on the graph (N=50, 2-hop) has a value slightly more than twice the initial value (N=0, 1-hop) because the size of a packet in 2-hop is greater.

CHAPTER I.3

A multiformalism modeling approach

Many different parameters should be taken in consideration while designing and deploying a wireless sensor networks and monitoring infrastructures. In the previous Chapter a "general" modelling framework to support the design of large sensor networks has been presented and a real system has been modelled through the adoption of the SAN model. The goal of this Chapter is to advance this modelling framework by showing how network and node models can interact to better represent the nature of cyber-physical systems.

The main idea behind this work is that the distributed nature of the application does not allow to separate a system model into a hierarchy of "completely independent" sub-models. The real world hypotheses at the base of this considerations are that **the network's quality and lifetime of an entire WSN depends on single node energy consumption as well as the rate at which messages are exchanged in the network**. On the other hand, the energy consumption of a single node depends on the sampling period, the architecture of the node and the quantity of messages the node must forward: *this last variable recursively depends on the topology of the network*. In order to cope with the different aspects of both nodes and network, a multiformal model is proposed and its effectiveness is discussed. Multiformalism approaches allows choosing the best suiting formalism at different levels of abstraction: (i) one can use a low-level formalism able to capture and model architectural and behavioral features when modeling at node level and (ii) use a formalism that may express complex topology and relations among nodes when focusing on large networks.

I.3.1 A multiformalism process

In order to cope with the different abstraction requirements of both nodes and network, a multiformal model is proposed, its structure is depicted in Figure I.3.1. It is composed by two sub-models: the *Node Model* and the *Network Model*. A Node Model is characterized by three kinds of input parameters: *architectural*, *application* and *radio*. The architectural parameters are in charge to model the hardware-related inner behavour of the node (e.g. the latency of the sampling activity); such parameters are almost constant since they are hard to be customized by the user and they change only between different revisons of the node family (e.g. Telos revA and revB [57]). The application parameters can be changed by means of software configurations: one of the most important is the sampling period by which the sensor node acquires data from the envinroment. The third class of parameters is constituted by the radio messages that are received from the other nodes: the frequency at which such messages are received is important since these messages should be forwarded to other nodes consuming node resources (energy and time). Since the dependency of this parameter from the network, it is calculated as the analysis results of the Network Model (*radio rx profile*).

The Network Model captures the aspects of spatial-aware interactions between nodes in order to combine simplified models of single nodes. The Network Model allows the modeler to specify the topology of the network by defining the spatial distribution of the nodes inside the monitored area. Moreover the behaviour of the single nodes inside the network can be specified by describing the probability of a node to run out of battery (*power-consumption profile*) and the rates at which each single node transmitts messages to other nodes (*radio tx profile*). These values are the analysis outcomes of the Node Model analysis.

Two formalisms are used: Stochastic Activity Networks (SAN) for the Node Model and Markovian Agents Model (MAM) for the Network Model. A brief introduction is required for MAMs [37]. MAM is an emerging formalisms where each Markovian Agent (MA) can be

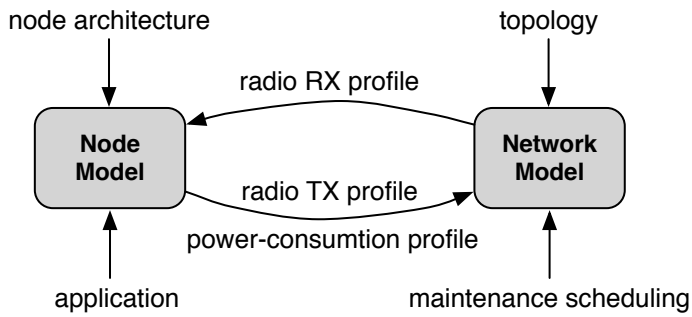


Figure I.3.1 The overall multiformalism model

modeled by means of discrete Markov chain. The MA communicates with the environment and with the other MA's by sending messages that can be received (perceived) by the other agents. The MA's are distributed in a finite geographical area according to a given spatial density and their interaction is defined by a perception function that depends on the spatial distribution.

The editing and analysis of both sub-models can be assisted by proper tools according to the SAN and the MA formalism: notwithstanding the analysis of the global model is in charge of the WSN designer. The designer should manually explore the DSE by trying different topology and application configuration in order to maximize network quality and lifetime: future works would automate such analysis.

I.3.2 Models of WSN nodes and network

This Section shows how the overall model described so far has been instantiated on the specific architecture of the TelosB node [57].

TelosB model using SAN: The Node Model has been realized according to a component-based view of the node architecture: this approach consists in composing different models, each one concerning with a specific aspect of WSN's node behavior (e.g. sampling, processing, sending, etc.). In the previous Chapter, a not publicly available architecture was used as reference, here instead the model is based on the architecture of TelosB[57] commercial nodes. Figure I.3.2 depicts the structure of the sub-models (rounded rectangles): which exchanges information through *interfaces* (small rectangles). The sub-models reflect

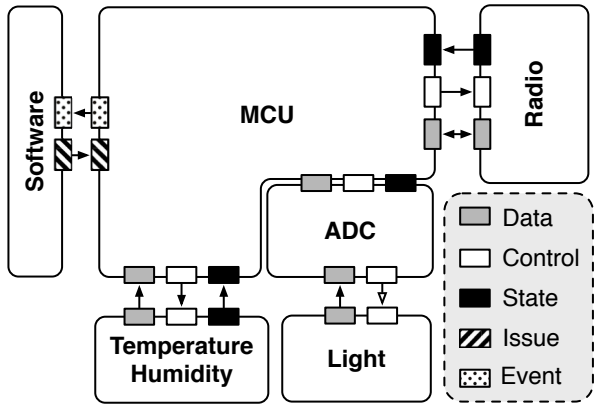


Figure I.3.2 TelosB model structure

the hardware internal organization of the TelosB node except for the *software* model: the MicroController Unit (*MCU*) model, the *ADC* model, the *temperature* & *humidity* sensing

model, the *light* sensing model and the *radio* model.

The *software* model is able to generate sampling requests (to on-board sensors) and to process samples; furthermore it is able to generate/manage messages working in cooperation with the *radio* model. All these actions are conducted producing *issue*s and consuming *event*s, which are exchanged with the *MCU* model through the homonymous interfaces.

The *radio* model implement the message exchanging mechanism: this model, receiving control signals from the *MCU* model, is able to represent the sending and receiving operations conducted by each sensor node.

Since the *Radio*, the *temperature & humidity* and the *ADC* represent hardware components, the corresponding models implement the data/control/state communication abstraction: each sub-model shares three interfaces with the *MCU* which represent the communication points; the *data* interface (from the component to the *MCU* model) is mono-directional with the exception of the *radio* model which needs a bidirectional connection since messages can be sent and received by the sensor node. The *Light* model does not have the *state* interface since it reproduces an analog component. In addition, the *control* interface should not be implemented but it have been introduced to increase the solving efficiency (avoiding the generation of tokens when no light sampling operations are needed).

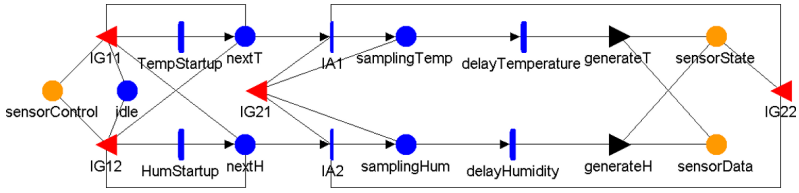


Figure I.3.3 *Temperature & Humidity* sensing model

As a reference, the *temperature & humidity* SAN model is described (Figure I.3.3). The *temperature & humidity* model represents the on-board sensors in charge of the sampling of these phenomena (Sensirion SHT11). Samples produced by this model are directly processed by the *MCU* model since this component is equipped with an integrated Analog/Digital Converter; the same behavior is represented by the corresponding model.

The three interfaces (*data*, *control* and *state*) have been implemented using extended places, which encapsulate complex structures able to store all the necessary informations. More in detail, when the "start sampling" command is received by the component, *IG11* and *IG12* check if a temperature or a humidity sample has been requested respectively. The *idle* place is necessary to store information about the operativeness of the component: if it is in an idle state, *idle* place is marked and the component needs an extra startup delay (*TempStartup* and *HumStartup* transitions) before the sampling action, this delay

is 0 otherwise. As in the real component, the model is able to store one next sampling operation: if a temperature sample request is occurred then a token is stored in the *nextT* place, otherwise a token is generated in the *nextH* place. The sampling is enabled by the firing of the *IA1* or *IA2* transitions, which are enabled by the input gates (*IG21* and *IG22)*: which check if another sampling operation is active and if a data is ready but not yet processed by the MCU. At the end, after a sampling action, data are stored in the *sensorData* place, while signals to the MCU are sent using the *sensorState* place.

The *ADC* model represents the Analog/Digital Converter, internal to the MCU. This model takes as input analog signals and produces digital samples which can be processed from the MCU. This component is connected with the *light* model which is able to reproduce the sampling of the light phenomenon.

The *MCU* model reproduce the behavior of the microcontroller: it processes issues coming from the *software* giving corresponding control signals to the hardware component, and manages packets received by the network and samples generated by sensors generating events for the *software* model.

Network model using Markovian Agents: The MAM has been described with the same formalism adopted in [37]. Two agent classes (*node* and *sink*) and one message type (*m*) have been identified. Figure I.3.4 depicts the MAM for the *node* class. The states are drawn as circles. A local transition is represented by a solid line and labeled with its transition rate. An induced transition is shown by a dashed line, the label is the message type which forces it to occur. The MAM has four states: (0) *No Radio* node is ready to perform a radio operation, (1) *Down* node is no longer operative due to failure or battery depletion, (2) *Receive* node is receiving a message and (3) *Send* node is sending a message.

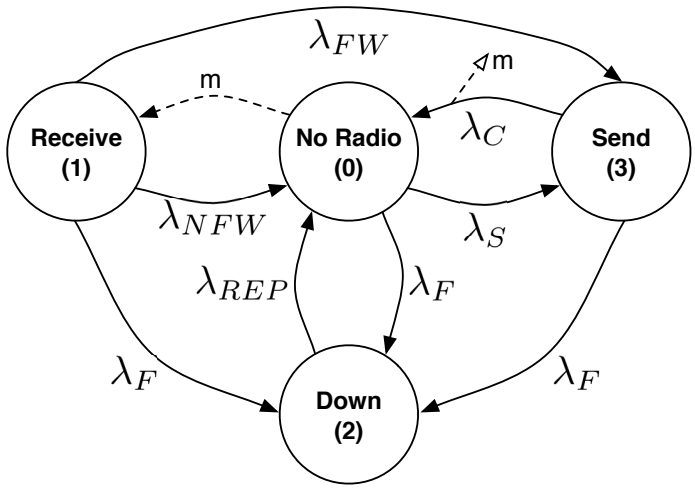


Figure I.3.4 Markovian Agent Class of the Node

The initial steady state is *No Radio(0)*. At a constant rate λ_S an agent can start a sending operation arriving in *Send(3)*. A send is completed with rate λ_C and a message (of class $\underline{m}$) is generated. This message can be received or ignored by other MAs according to the *perception function* $u_m(\cdot)$. A a simple $u_m(v, \bar{v})$ is adopted which is function of agents position in the space.

$$u_m(v, \bar{v}) = \begin{cases} 0 & dist(v, \bar{v}) > T_H \\ 1 & dist(v, \bar{v}) \leq T_H \end{cases}$$

Upon message arrival ($u_m(v, \bar{v}) = 1$) the agent moves form *No Radio(0)* to *Receive(1)* state. The agent, with rate λ_{NFW}, does not forward the massage and returns in *No Radio(0)*, on the other hand it forwards the message with rate λ_{FW} and arrives in *Send(3)*. In each of the states described above the agent can fail with rate λ_F moving to *Down(2)*. The agent can return in *No Radio(0)* with the repair rate λ_{REP}.

The sink agent class Markovian model is similar to the one presented above except for the absence of the *Send (3)* state. For the sake of simplicity an hypothesis is introduced: it is supposed that the most relevant part of the traffic is flowing from nodes to the sink (monitoring application) and for this reason it is possible to not consider the traffic directed in the opposite direction.

Models Interaction Figure I.3.5 details the parameters exchanged by the two models. With reference to Figure I.3.1, some of the SAN model results describing both the battery discharge and the radio transmission profiles are needed by the MA network model in order to be analyzed. Such parameters and their meanings are described above. On the other hand, MA model produces for each agent a radio receiving profile that is represented by the state sojourn times at steady state π: in particular π_0 is the mean sojourn time in *No Radio* state, π_1 is the mean sojourn time in *Receive* state. These two values are used by the SAN model to compute the rate of incoming messages. In general each node of the network is characterized by a tuple of the $v = (\lambda_F, \lambda_S, \lambda_M, \lambda_{NFW}, \lambda_{FW}, \pi_0, \pi_1)$ parameters. Thus a network on n sensors is characterized by a vector of $V = (v^0, ..., v^n)$ arrays of parameters.

The interaction between these two models is clearly recursive: some considerations must be done on the solution process of the global model. The propose is a two phases iterative process that, starting from an initial solution, aims at finding a fixed point of V. In such case, fixed point iteration methods could be applied: first, an initial value $V_0 = (v_0^0, ..., v_0^n)$ is set; then SAN models and MA model are analyzed to calculate $V_1 = (v_1^0, ..., v_1^n)$. The iterations stop when a fixed point is reached: initially all the SAN models of the n nodes are fed with the same values thus a single execution of the SAN model analysis is enough. As the fixed point iteration run (say at the generic $k-th$ iteration), v_k^i starts to differentiate from v_k^j where $i, j \in \{1, ..., n\}$ and $i \neq j$ since, under the effect of the network topology, the MA contributions

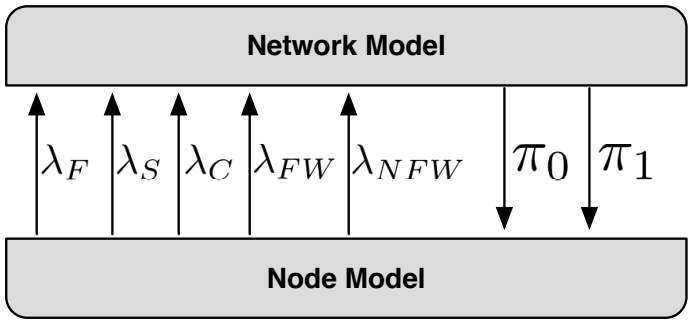


Figure I.3.5 Parameter exchange between the models.

of the different agents inside the model change in different ways the behaviors of the SAN model of each node. Thus more than one SAN model simulation is needed.

The formal assurance of the existence and the uniqueness of such a fixed point solution and the assessment of the proposed solution process are currently ongoing research works. Moreover other approaches can be exploited as Genetic Algorithms or Particle Swarm Optimization.

I.3.3 Experimental Results

In this Section experimental results are reported in order to validate the node model and analyze the network quality.

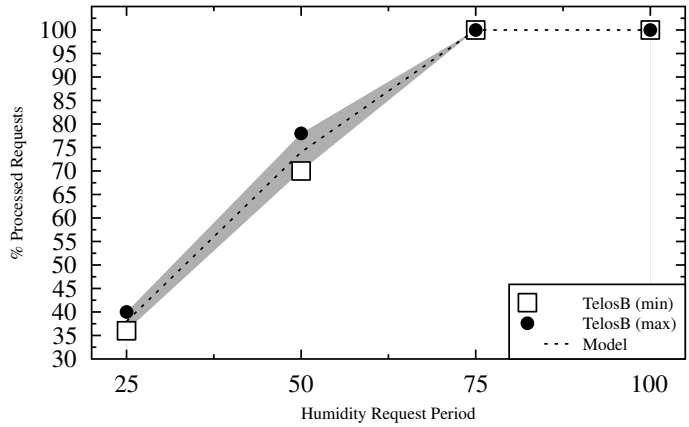
Node validation The testbed is composed of real TelosB sensor nodes in order to tune the node model parameters. In particular, the startup, sampling and processing delays have been modeled as Gaussian random variables whose parameters are reported in Table I.3.1. The behavior of different nodes is analyzed while varying the temperature and humidity sampling frequencies.

Parameter	Average	Variance
Temp&Hum sensor startup delay	10.5 ms	0.25 ms
Temperature sampling delay	234 ms	12.25 ms
Humidity sampling delay	68.5 ms	12.25 ms

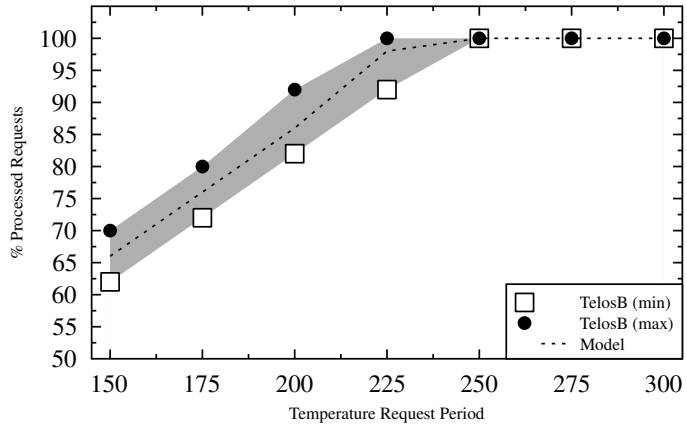
Table I.3.1 SAN parameters for Temperature and Humidity Sensor

Repeated measures have been conducted on different TelosB nodes in order to gather information on the percentage of processed requests while varying the sensor sampling period. gThe nodes were configured with a TinyOS operating system and no radio interaction. The performed measures go from the minimum sampling period (higher frequencies [57]) to a

sufficient long one when all requests are served. Each time the on-board application issues a read request, the timestamp is marked and stored in memory. In case the corresponding sample is generated, the arrival time is saved, too. After a predefined number of requests the collected timestamps are sent to a computer through a serial connection.
The following plots show the accuracy of the proposed SAN model against real nodes behaviour. As one can see in Figure I.3.6a and in Figure I.3.6b the model well fits the mean value of the percentage of processed requests for both the humidity and temperature sensors, actually all values are inside the gray area which is limited by the min and max values obtained from real TelosB.



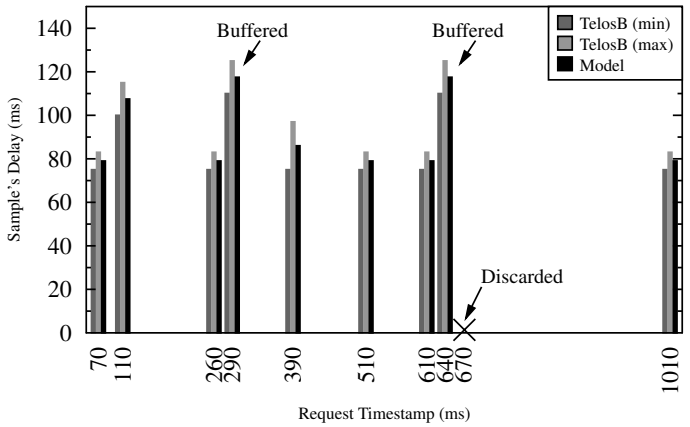
(a) Validation of Humidity Sensor



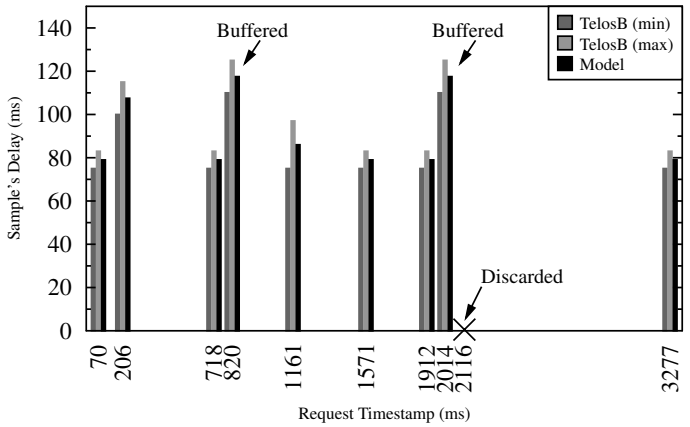
(b) Validation of Temperature Sensor

Figure I.3.6 Validation of Sensors

In Figures I.3.7a and I.3.7b sensor delays are reported (including sensor startup and sensing), these values are measured at different instant of times (randomly chosen) and performed on different TelosB nodes. Consider the three consecutive requests at 610, 640 and 670 ms in Figure I.3.7a[1]. The plot report for each request the min and max values obtained on TelosB and the node results. The 1^{st} and 2^{nd} requests are accomplished by both real node and Node Model. The sample delay is longer for the second request due to buffering. Indeed both real and model nodes serve one request at time and can buffer only one more. This is the reason why the 3^{rd} request (670 ms) is discarded. It is important to notice that the



(a) Humidity Sensor: Request/Sampling time (ms)



(b) Temperature Sensor: Request/Sampling time (ms)

Figure I.3.7 Sensors Timing

delay of consecutive request is not constant. The former is served immediately but lasts

[1] Same considerations hold for Figure I.3.7b at 1912, 2014 and 2116 ms

longer due to startup operations, on the other hand the latter waits to be served (buffered) but his actual service time is shorter. Even in this case, the results are interesting as they show how well the model fits the real node behavior.

Network analysis: The network behavior is evaluated focusing on the effective packet delivery ratio (PDR) and average lifetime. In order to do so the network characteristics have been simplified in terms of fixed link and fixed PDR. Figure I.3.8 depicts the network configuration. In this scenario each node generates a message as soon as the sampling operation completes (with 30s sampling period), furthermore nodes *2*, *4*, *7*, *10* forward all the traffic they receive.
The parameters used during simulations are reported in Table I.3.2. Some nodes have been simulated with diverse configurations in terms of different *Battery* and different *MTTR*. For each case the average mean time to battery depletion among all nodes is computed and the network throughput evaluated as the number of packet the sink can receive (network PDR) under the assumption that the sink is not prone to failure.

Parameter	**Description**	**Value**
Duty Cycle	Percentage of time the node is active	30%
Active	Current consumption while the node is active	3 mA
Idle	Current consumption while the node is sleeping	0.015 mA
Radio_TX	Current consumption to send	21.7 mA
Radio_RX	Current consumption to receive	22.8 mA
Packet_Time	Time to send/receive a packet	0.008 s
Battery	Battery capacity	1500mAh
MTTR	Mean time to repair a node	1 day

Table I.3.2 Simulation parameters

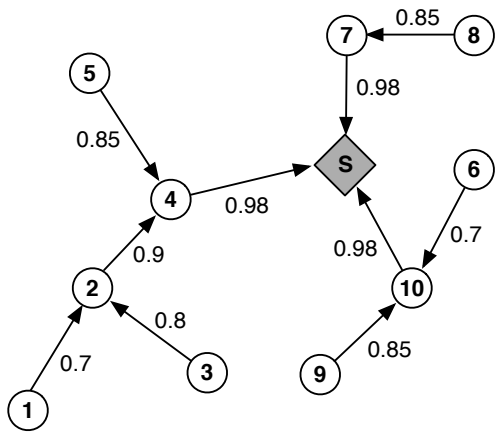


Figure I.3.8 Topology with fixed PDR

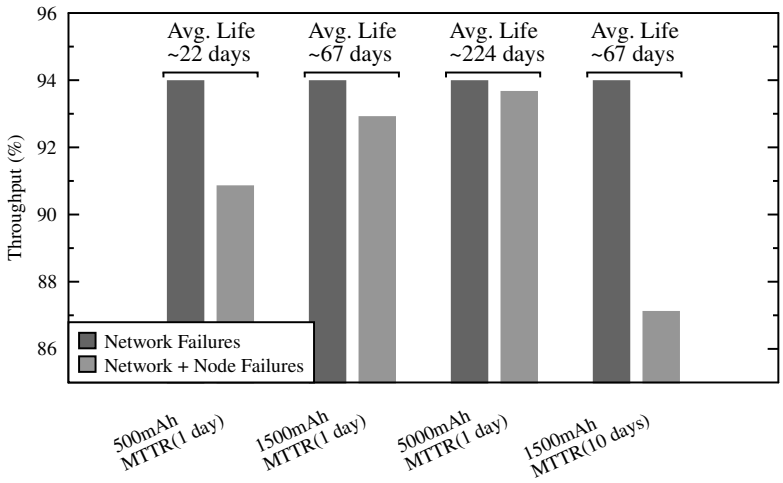


Figure I.3.9 Validation of Temperature Sensor

Figure I.3.9 reports the simulation results. The PDR values are weighted on the maximum network delivery capacity calculated in the absence of failures. For each configuration the first bar represents the effect on PDR due to faulty links; as one can notice this value is independent of the configuration because the parameters which change are not related to the network. The second bar, in each group, takes into account the effect of node's availability. As expected the PDR is affected by both battery capacity and mean time to repair: a large battery capacity increases the nodes availability and consequently the number of packets the network delivers. On the other hand a long mean time to repair causes a loss of throughput (lower PDR).

CHAPTER I.4

Summary

In this part of the thesis, it has been presented how a model of a CPS comprises models of physical processes as well as models of the software, computation platforms, and networks. Moreover, it has been shown how to address the feedback loop between physical processes and communication infrastructure.

Modeling such systems with reasonable fidelity is challenging due to the large number of heterogeneous components; the composition semantics becomes central. Since the design space can easily become very huge, the designer must be provided with methodologies and supporting tools in the choice of best configurations and thus a quick evaluation of different configurations is necessary. The modelling approach hereby presented is an advance to the relative state of art in the direction of easing the design process. The contribution of the compositional and multiformal approach is twofold. A comprehensive and detailed model of sensing infrastructure embedded node has been presented: even if the architecture of the model is quite generic and can be adapted to several kinds of nodes, it has been proven to be consistent to real world architectures by comparing simulation results with real world measurements for both the cases of Strago SpA architecture and commercial TelosB. Moreover, it has been show that Markovian Agent Model can be very useful in the detection of quality/reliability bottleneck of the network and on the definition of proper and cost effective maintenance policies. The formal assurance of the existence and uniqueness of a fixed point solution for the recursive dependencies is still an ongoing work. For the tests conducted it was possible to converge with a reasonable confidence in three to four recursions. Even though, the number of sensor nodes has been limited to the tens so as to

validate the model easily, the results collected so far confirm that this approach supports designers in the choice of different parameters by means of what-if analysis and early measurements to fine tune the monitoring infrastructures variables (such as sampling period, number of nodes, battery allocation, node dislocation). A future research plan is to extend the simplified phenomenon model in order to be spatially aware and represent in a better way real world physical processes. Moreover, the Markovian Agent model will be extended so as to threat different message types in order to improve the network model and reducing the gap with real applications.

PART II

Design a Secure Monitoring Infrastructure

CHAPTER II.1

Introduction and Background

Nowadays, networks of embedded nodes can accomplish a wide number of tasks. In spite of their resource constraints (namely elaboration capabilities, memory space, power-supply) many applications and new services have been effectively deployed in different domains. Due to their capabilities, it seems very promising to use such distributed and heterogeneous devices to design CPS's monitoring infrastructures. Unfortunately, the unattended deployment, lightweight protocols and data sharing lead to new risk scenarios that cannot be faced with traditional security mechanisms. This is the case of monitoring infrastructures, where different types of embedded systems are connected to the physical infrastructure and their data are sent to and elaborated by the processing infrastructure. The interaction among these layers is so tight that attacking the physical layer can produce the same negative effects of attacking the processing one. The geographic distribution of the devices allows an attacker to physically capture nodes and learn secret key material, to intercept or inject messages; the hierarchical nature of sensor networks and their route maintenance protocols permit the attacker to determine where the critical nodes are placed[6].

Security threats range from the authentication of the device, to the confidentiality, trustworthiness and integrity of sensed data. Existing literature has proposed the use of computationally inexpensive cryptographic techniques to handle message confidentiality and authenticity in wireless connected monitoring infrastructures [6]. Due to the possibility of node's physical capture, any cryptosystem must therefore tolerate the compromise of sensors and their keys. However, the compromise of some nodes need not result in a total loss of security. Rather than providing all-or-nothing guarantees about privacy or

security, there is a need for providing probabilistic guarantees with respect to compromise [6, 16]. Techniques for detecting tampering, and validating the inputs provided by sensors is important to prevent these control inputs to the cyber-physical system from being recruited by adversaries (e.g. bot-nets)[59]. Moreover, for a sensor value to have meaning, the context is needed for interpretation, such as where, when and by who it has been sensed. Conversely, if these information about a reading are known, it may be possible for an adversary to infer a great deal about other readings nearby. Therefore, designers must be aware of these metadata and the role they play in security. Furthermore, due to data aggregation and hierarchical topologies, the distribution of information among the network is not uniform. Therefore, attacking a leaf node in a tree-structured network gains little influence (for disruption or denial of service) or information (for eavesdropping); attacking a node near the root gains significant influence and information about the aggregate value[6, 16, 59]. Security mechanisms should be able to hide and obfuscate the network topology.
The difficulty of ensuring confidentiality and authenticity is not, however, due solely to the energy constraints imposed on sensors. Todays approaches to security are based on *reactive mechanisms*: after a threat detection a corrective action is performed. Detection systems are useful in many cases but false alarms and missed detections are impossible to avoid because identifying malicious logic is an undecidable problem [16].

To overcame limitation of currently available solution and to address others security challenges it has been decided to investigate solutions based on *reconfiguration approaches* and *special hardware architectures* for sensor nodes.

Adaptation Techniqies and Reconfiguration approaches The theme of security through reconfiguration, although intuitive, was only recently formally addressed in the Moving Target Defence (MTD) and Adaptation Techniques (AT) concepts [34, 43, 44]. The main idea of those solutions is to dynamically modify the attack surface to increment attackers' uncertainty. As stated in [35]: "Moving Target Defense enables us to create, analyze, evaluate, and deploy mechanisms and strategies that are diverse and that continually shift and change over time to increase complexity and cost for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency". By applying AT and MTD techniques to embedded nodes they will be able to show a different and non predictable facade to attackers in terms of different OS, different software modules, different communication protocols and security mechanisms. This will help to engineer solution aimed at offering: (i) context obfuscation, (ii) secure data aggregation, (iii) resiliency and survivability, (iv) topology obfuscation, (v) trust and (vi) key management. As for security evaluation, few quantitative approaches exist in the literature to evaluate the augmented security provided by MTDs as it depends on many security attributes and reconfiguration frequency. Authors in [17] carried out a number of experiments by simulating attack scenarios where an at-

tacker is able to gather partial information on the adopted cryptosystem and attempts a brute force attack. They evaluated the effectiveness of a moving target defense approach by measuring the probability of successfully completing an attack, furthermore they evaluated how reconfiguration dramatically decreases such probability when varying reconfiguration frequency for different security mechanisms. Different reconfiguration mechanisms can be enforced to change embedded systems surface but they are highly coupled with the specific target technology. For this reason it has been decided to focus the attention on Wireless Sensor Networks (WSN) technologies for cyber-physical systems monitoring infrastructures. Following discussions will refer to this specific case.

Reconfiguration mechanisms for WSN were primary conceived to reconfigure a network and restart it after a failure or a design change, but not for security purposes. In the literature, some security and reconfiguration mechanisms are available [18, 19, 56, 80]. Unfortunately they tend not to be efficient in: time, security and energy consumption.
In order to engineer proper reconfiguration schemes and configurable properties of a network, in the following Chapters, the reconfiguration grain for generic networks of embedded nodes is first identified. Moreover, different reconfiguration mechanisms and design issues that one need to address to effectively implement AT for WSN are discussed. To this extent a framework, called SIREN, is proposed to provide a secure and efficient reconfiguration mechanism. Its suitability for AT will be shown: it is able to quickly restart any monitoring application with new security features in order to shift and alter its attack surface. Indeed, with SIREN one is able to prevent and mitigate a large set of attacks based on eavesdropping and sniffing of communication channels but the evaluation that will be presented is intend to prove the feasibility of the proposed framework to enable the adoption of adaptation techniques and moving target defenses.

Special Hardware Architectures for sensor nodes Cyber Physical Systems typically operate unattended in hostile outdoor environments. A lot of effort has has been made to protect the communication between sensing nodes and the processing infrastructure. However, with regards to physical protection of a node, assessing the integrity and trustworthiness of its hardware/software is still an open and challenging issue. Systems that lack of physical protection for either the software or the hardware, may put sensitive application, data and the integrity of the overall system at risk.

Physical security should be considered at the beginning of embedded devices development in order to efficiently and effectively secure the architecture of embedded nodes. So as to propose a solution to physical threats, in the following Chapters, a new node architecture is discussed and evaluated. The architecture makes use of Trusted Platform Module (TPM) to perform cryptographic operations in a trustworthy manner. The Trusted Platform Module (TPM) is a specification for a secure cryptoprocessor defined by in the ISO/IEC 11889 of

the Trusted Computing Group (TCG) [36]. This specification defines the components and operations which a cryptoprocessor has to implement in order to build a chain of trust. It builds a chain of trust which enforces a trustworthiness relationship among the node's components. Given this capability, the node will function only if all the hardware and software configurations have been verified by means of cryptographic operations. Moreover, using tamper resistant hardware it will ensure that the cryptographic keys equipped in the node do not leave a secure perimeter. A prototype is presented making use of FPGA technology.

CHAPTER II.2

Adaptation Techniques to Secure Resource Constrained Embedded Nodes

Many physical infrastructures are monitored through a wide range of embedded systems such as special purpose sensors and actuators. They can be easily and dynamically connected to gather different information about the status of the system [73]. Modern technologies made easy to connect and disconnect more nodes into the network, to configure and deploy new applications or to monitor different phenomena but the capabilities of a sensor device are limited. Moreover, they could be targeted by a plenty of attacks and their protection is non trivial. One of the main open issues in the deployment of such distributed embedded systems is related to the security and performance of the cyber-physical layer [22, 56, 83]. Traditional computer security approaches tend to be resource greedy, the literature focused on how to re-engineer them and on proposing ad hoc solutions. These solutions try to balance the trade-off between security and performance but, as a general rule, none of them can be considered definitive and "truly secure", especially if it is enforced for a long period of time.

Reducing the "exposed time" of a security mechanism to attacks, reduces the probability of performing a successful attack [44]. Furthermore a system reconfiguration takes advantages of software diversity, different cryptosystem and security key renewal process. In this thesis and in publication [12] a framework for SecurIty Reconfiguration of Embedded Nodes (SIREN) is discussed. It will be shown how SIREN makes possible to adopt Moving Target and Adaptation Techniques security strategies to the case of embedded nodes and

how to exploit their intrinsic advantages in terms of security level provided. SIREN is made of a centralized component, deployed in the processing infrastructure to control reconfiguration strategies, and a node-level (monitoring infrastructure) component to enforce the reconfiguration. Some modules of TinyOS (one of the most widely used OS for sensor nodes) have been extended in order to enable a selective and efficient application reconfiguration. In particular, it is able to selectively reconfigure each single node according to different administration policies and to react to triggers that can initiate the reconfiguration process. The suitability of SIREN for Adaptation Techniques will be discussed and illustrated with a set of experiments to evaluate the impact of the approach on node performances and resources in terms of reconfiguration delay, memory footprint and battery consumption.

II.2.1 Adaptation Techniques and Moving Target Defense in embedded sensor nodes

Security and reconfiguration mechanisms are strictly dependent on the technology and system architecture. Different reconfiguration mechanisms can be implemented according to the available components and different reconfiguration strategies can be enforced. For this reason, following discussions will explicitly target Wireless Sensor Networks (WSN) technologies for cyber-physical systems monitoring infrastructures. At this aim two steps have been followed. The first step has been developing and adapting different security mechanisms to protect the communication among embedded nodes and cope with performance issues; these mechanisms will be part of the node configuration and each of them will expose a different "surface" to an attacker. The second step consisted in providing reconfiguration capabilities to all nodes in the network in order to switch their configuration and security features or, telling it in another way, to change their "attack surface" [44].

Before introducing the reconfiguration architecture, next subsection discuss and illustrate some solutions to secure and to reconfigure nodes that are already available in the literature but, at current time, present some limits and constraints that make them not immediately applicable in real scenarios.

II.2.1.1 Security and reconfiguration techniques in embedded sensor nodes

Security mechanisms, to protect data in embedded nodes, require proper security protocols since hardware and software constraints limit the adoption of cryptographic techniques used in other contexts.

Two main well-known mechanisms are available: in Symmetric Key Cryptography a unique secret shared key is used for both encrypting and decrypting messages, while in Public Key Cryptography each node manages a couple of keys. On one hand, symmetric

schemes are very attractive for their energy, memory and performance efficiency and many approaches are available for WSN (TinySec [47], MiniSec [52] and SNEP [63]), but they only fulfill confidentiality requirements, while not considering other security issues such as node authentication and data integrity. They are prone to a number of cryptoanalisys and brute force attacks. Furthermore, the key agreement phase seems to be critic and the adoption of a pre-loaded symmetric key is not a security desirable requirement. On the other hand, the use of asymmetric schemes in WSN has been usually considered as "nearly impossible" because of their great impact on energy and computational resources and the requirement of an infrastructure for key management. However, RSA algorithm [66] and, above all Elliptic Curve Cryptography (ECC) [21], are among the most adopted public key algorithms for the security of embedded systems.

All these protocols are subject to a number of different attacks and the longer is the exposition time, the higher is the probability of successfully completing an attack. No matter how secure a mechanism is [18], from the MTD point of view, the interesting fact is that these two mechanisms offer two different attack surfaces to an attacker and the reconfiguration approach provides a higher security level [44]. For example, to improve the overall security provided by the system, one can switch among different crypto libraries or different keys giving an attacker less time to perform the attack.

With the aim to provide reconfiguration capabilities to wireless sensor networks, some reconfiguration solutions were addressed in literature:

Image Replacement: this technique replaces the whole software stack running on a node. A framework, which uses this approach in WSN, is Deluge [41]. It replaces the binary image receiving a new application over radio channel (over-the-air reprogramming). Deluge is the most widely used reconfiguration framework for WSN. The most of the effort is given to implement an efficient and reliable protocol by which one can transfer new images of an application. For example, the differential code updates solution tries to reduce the size of data sent to a mote [46, 62].

Virtual Machine: this solution uses a code interpreter running on nodes to switch among different applications. One implementation for WSN is Maté [51] which is a bytecode interpreter that runs as a middlewares on top of TinyOS.

Modular Replacement: this technique proposes a partial reconfiguration of nodes, it aims to overcome the monolithic architecture of some reconfiguration approach but it is not available for many embedded devices.

On the other hand, to achieve certain MTD's objectives for embedded nodes, some requirements are needed. First of all it is necessary to have secure, flexible and dynamic reconfiguration mechanisms and the reconfiguration should not waste the limited resources available on nodes and should ensure the continuity of the monitoring application. Un-

fortunately, according to the above requirements, the analyzed solutions are not directly suitable for MTD as they:

- do not provide a secure dissemination scheme;
- do not provide a selective node reconfiguration;
- are very time and battery consuming.

Regarding the dissemination scheme, Deluge offers unsafe interface to reprogram a node. Indeed there are no authentication and authorization phases when receiving a command. Interfaces are public and every one can establish a communication with a node. Moreover, during the dissemination phase, the attacker can pretend to be the legitimate node and subsequently receive the complete binary image. Also the attacker could disrupt image transmission with jamming attacks, causing excessive battery consumption that could breakdown the nodes. Dutta et al., in [33], present a way to secure the Deluge's dissemination protocol but a plenty of attacks are still feasible and discussed by the authors. Furthermore, Deluge only allows to configure the entire network with a single application at a time (homogeneous network). This limitation makes hard to fully exploit advantages of moving target defense and adaptation techniques. On the other hand, through a selective node reconfiguration one can control each single node and achieve asymmetric uncertainty for attackers by showing a number of different views of the network.

The results of this analysis is that available reconfiguration mechanisms present feasibility issues to be adopted for AT purposes and subsequently it has been decided to purpose a new framework. Next section formalizes the possible reconfiguration actions (switching functions) while the following ones will illustrate details about the architecture.

II.2.2 Modeling what is reconfigurable in WSN's architecture

In this section, a set-based formalism is introduced to determine the possible reconfiguration solutions before proposing the WSN mechanism. The introduction of these general definitions is very useful, in fact many concepts and issues will remain valid even for different node technologies (e.g. WSN, FPGA and Smartphones).

The basic idea is that a network is defined by its nodes' and edges' sets. A node represents a processing unit and an edge is the communication link between a pair of nodes. Moreover, a node is described by its architectural stack (from the hardware layer up to the application interfaces). Thus, the definition of network and node as the composition of reconfigurable elements defined over finite sets. First of all, the concept of *general network* is defined: it is a set of nodes and suitable communication links between them. The composition of those links is used to define different network topologies that are called *network configurations*. As regards nodes, they are represented by hardware resources, operative systems, application

task and supported security mechanisms. Thus, the definition of the *general node* concept that represents the set of all possible kinds of node.

Definition (General Network): *a general network (GN) is a triple* $\langle \mathcal{V}, \mathcal{E}, \mathcal{P} \rangle$ *where,*

- $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ - Is the *node* set. Its cardinality is N.
- $\mathcal{E} = \{e_{ij} / i \neq j; \quad i \epsilon [1 \ldots N]; \quad j \epsilon [1 \ldots N];\}$ - Is the set of possible *links*. Hence e_{ij} is an ordered pair defined on $\mathcal{V} \times \mathcal{V}$. If the edge e_{ij} exists and belongs to $\mathcal{E}$ the v_i node can directly communicate with v_j over the e_{ij} link.
- $\mathcal{P} = \{p_1, p_2, \ldots, p_K\}$ - Is the *protocol* set which defines the admissible communication protocols. A communication protocol is a set of standards to implement a communication.

Definition (General Node): *a general node (gn) is a collection of admissible parameters which constitute the layered architecture of an embedded node* $\langle \mathcal{A}, \mathcal{S}, \mathcal{L}, \mathcal{O}, \mathcal{T} \rangle$ *where,*

- $\mathcal{A}$ - is the set of application interfaces to interconnect applications distributed across multiple nodes, such as application level APIs.
- $\mathcal{S}$ - is the set of security mechanism to protect a node.
- $\mathcal{L}$ - is the set of core application which a node can execute.
- $\mathcal{O}$ - is the set of operating systems supported by the node's hardware.
- $\mathcal{T}$ - is the set of hardware resources (hardware architecture) which defines the physical environment and capabilities, such as micro-controllers, radio chips or sensors.

The composition of different components abstracts the real behavior of a node and of the network. Starting from a *General Network* or a *General Node* it is possible to choose an element for each set; these instances are respectively called network/node configuration.

Definition (Network Configuration): *given a GN, a network configuration is an instance of GN. Hence a network configuration (NC) is an element* c_i *of the Network Configurations Set* $\mathcal{C}_{\mathcal{N}}$ *which is defined by a subset of the Cartesian product* $\mathcal{C}_{\mathcal{N}} \subseteq \{\mathcal{V} \times \mathcal{E} \times \mathcal{P}\}$.

It is important to notice that, $\mathcal{C}_{\mathcal{N}}$ is not strictly equal to the Cartesian product because not all the combination in $\mathcal{V} \times \mathcal{E} \times \mathcal{P}$ are suitable for a configuration. For instance, if $\mathcal{V} = \{v_1, v_2, \ldots, v_5\}$ and a *NC* involves only nodes $\{v_1, v_2, v_3\}$, a suitable combination must be defined using only links between these three nodes. Furthermore, it is also possible to vary the configuration by modifying the network topology over time with a *configuration switching*. It abstracts the switching to a new available network configuration.

$$f_{\mathcal{C}_{\mathcal{N}}} : c_i \epsilon \mathcal{C}_{\mathcal{N}} \rightarrow c_j \epsilon \mathcal{C}_{\mathcal{N}}$$

As an example, Figure II.2.1 shows different topologies of a WSN; as illustrated, it is possible to move from a star topology to a hierarchical topology using a $f_{\mathcal{C}_{\mathcal{N}}}$ which changes the active links.

Definition (Node Configuration): *given a generic node, a node configuration is the set of instances from each node parameters sets. A node configuration (nc) is an element* c_{n_i}

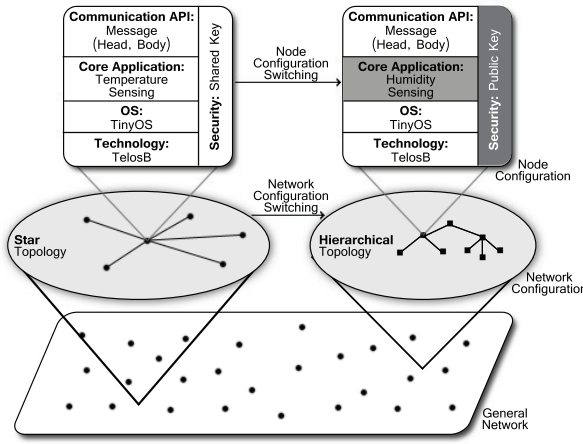


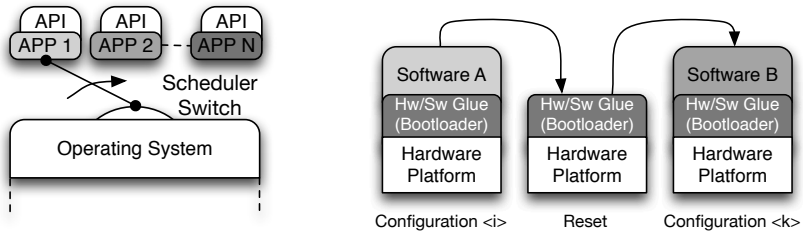
Figure II.2.1 Abstraction of Network and Node Configuration

of the Node Configurations Set $\mathscr{C}_n$ *which is defined by a subset of the Cartesian product* $\mathscr{C}_n \subseteq \{\mathscr{T} \times \mathscr{S} \times \mathscr{L} \times \mathscr{O} \times \mathscr{A}\}$.
Note that even in this case, $\mathscr{C}_n$ it not strictly equal to the Cartesian product because not all the combination in $\mathscr{T} \times \mathscr{S} \times \mathscr{L} \times \mathscr{O} \times \mathscr{A}$ are suitable configurations. In the same way as described for a *NC*, it is possible to define a function for the *node configuration switching*:

$$f_{\mathscr{C}_n} : c_i \epsilon \mathscr{C}_n \rightarrow c_j \epsilon \mathscr{C}_n$$

Figure II.2.1 also illustrates different node's architectures, in particular the node architectural stack is illustrated, where each component is defined and it is typical of a WSN *node configuration*.

It is important to underline that a configuration achieves certain performance, reliability and security requirements, so switching between different configurations enables to dynamically improve security and meet several goals but many design and implementation issues may arise. They may depend on the running application and its status, on resource availability, on the security level to provide and on available configurations. For these reason a framework to address these issues has been designed, it has a centralized approach that control the network and the node reconfiguration strategy.
Figure II.2.2 illustrates the working principle of two node configuration switching functions; in particular, Figure II.2.2a refers to *application level* reconfiguration, where the API layer ($\mathscr{A}$) or a part of the application task ($\mathscr{L}$) are substituted without varying the other stack's elements. This mechanism is easily implementable by using an *application scheduler* which executes the right application logic to apply a certain configuration. Another feasible reconfiguration is to change even the operating system ($\mathscr{O}$) or *the entire software stack*: Figure II.2.2b illustrates this kind of reconfiguration. This switching mechanism is able to



(a) Application Switching (b) Image Switching

Figure II.2.2 Examples of Configuration Switching Functions

replace all the software layers starting from the operating system up until the API interface layer ($\mathscr{A}$, $\mathscr{L}$, $\mathscr{S}$, $\mathscr{O}$). A persistent layer of low level software, like a bootloader, is able to reconfigure the application tiers performing a node reboot. Whenever one want to move from the configuration $c_{n_i} \epsilon \mathscr{C}_n$ to $c_{n_j} \epsilon \mathscr{C}_n$ he/she have to go through the reset configuration (node reboots) $c_{n_r} \epsilon \mathscr{C}_n$, following the path: $c_{n_i} \rightarrow c_{n_r} \rightarrow c_{n_j}$. In both cases, a persistent layer is needed to preserve the node state and the running application state; this can be achieved in different ways. In Application Switching scenario it is assumed that all the applications share global variables or they can rely on OS persistence mechanisms to share the state and application wide variables. In Image Switching, the node state is saved on the external memory before the reset and restored through bootloader initialization actions.
Next sections will illustrate and detail a the proposed reconfiguration framework for commercial embedded nodes (such as micaz [25] and telosb [57] motes) and discuss the feasibility of the approach by evaluating the impact of reconfiguration on:

- the interval of time the node will be not responding because under reconfiguration;
- the resource consumption (memory and battery).

II.2.3 SIREN Architecture

In this section the *SecurIty Reconfiguration for Embedded Nodes* architecture is described. It is designed considering the WSN constraints and it is intended to offer a fully automated mechanism to dynamically reconfigure wireless nodes and limiting the exposure of vulnerabilities. It is able to reach a fine grain control enabling to selectively configure each mote in the network ($f_{\mathscr{C}_n}$). SIREN relies on centralized component in the network to control reconfiguration strategies and a node-level component to enforce the reconfiguration. In particular, the node-level components have been designed for TinyOS operating system enabling to switch between configurations (c_{n_i}).

Being the primary goal to shift and alter the attack surface by moving from a cryptosystem to another and changing the format and protocol of exchanged messages, two

different configurations with two security libraries have been considered (Figure II.2.3): TinyPairing[80] and WM-ECC[56] to provide different security levels [18]. The TinyPairing libraries use node identities to evaluate a common key to be adopted in an Elliptic Curve Cryptography (ECC) based encryption scheme [80]. WM-ECC libraries enable Public Key Cryptosystems in WSN so two communicating nodes can achieve the same secret key without physically exchanging it across the network. For simplicity, these configurations only differ for the communication API (message format) and the security library.

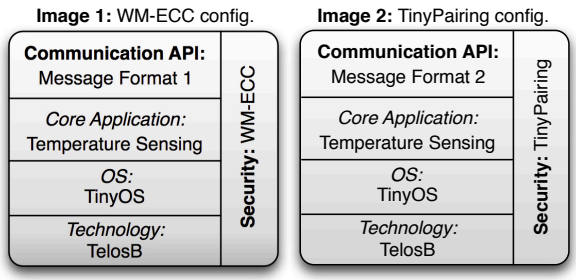


Figure II.2.3 Node (Security) Configurations

TinyOS, the most used operating system in WSN domain, lacks of reconfiguration support. It is compiled as a monolithic program, so that the application logic and the operating system are melted into a unique entity. In this way they are statically compiled and optimized, but there are no replaceable modules. Both Application and Image Switching can be implemented in TinyOS, nevertheless, trying to mitigate or prevent the widest number of attacks, it has been decided to implement and evaluate the Image switching strategy that is more complex to realize but it provides a higher level of diversity.

The basic idea is that each node can be equipped with a set of different binary images (configurations c_{n_i}), loaded into external flash memory, which are called *built-in software images*. A built-in image contains all the components to configure and run a node application apart from the bootloader. Each image implements a different cryptosystem and provides a different communication interface; the attacker's system view is changed by switching among the available binary images. In this way, not only different security protocols and communication interfaces are presented to attackers during the network's lifetime, but it also becomes harder for them to exploit software vulnerabilities.

The SIREN architecture involves both node-side components and control-side components. As names suggest, a node-side component is deployed within an embedded node, on the other hand control-side component is used to deploy and manage the WSN and reside on a workstation. The node-side component is designed to offer an high level API interface to enable selective reconfiguration in TinyOS operating system. The control-side enforces a

reconfiguration by sending a command to select which of the built-in binary images have to be executed. It has been implemented in Java and offers a user-friendly interface to deal with event detection, network management and reconfiguration. In the next subsections, details of these components are illustrated.

II.2.3.1 SIREN Architecture: control-side

The base station, usually provided with a consistent source of energy and able to communicate with all the network nodes, is used to control the network. It triggers a reconfiguration command to all nodes of the network or to a selected one, the command is sent in unicast as the framework is able to perform a fine grain control of all nodes in the network. In this way it is possible to vary the topology of the network, to modify the application of a single node and to allow multiple heterogeneous communication to coexist in the network. The control-side architecture has two macro components:

- *Deployment component*: it is the set of tools used to initialize a mote and to configure it with built-in application images.
- *Reconfiguration component*: is the software capable of monitoring the WSN and enforcing a reconfiguration. It enables an administrator to have a high level view of the WSN and to control each node.

The *Deployment* component is made of a set of scripts by which configuring each node belonging to the WSN. The core component is the *mote-manager*. Basically, it allows to manage the binary images loaded into mote's external flash memory. It is used during the deployment phase and it interacts with motes only through serial interface and deals with mote's external flash memory management. The mote-manager is used to store the built-in application binary images into the mote's external flash. When a mote needs to be deployed, an operator connects a dormant mote to the workstation and flashes, just once, the operating system plus a management commands handler into the mote. Now the node can be configured with a set of built-in applications. At the configuration completion, the monitoring application is executed and the node is ready.
After a deployment, one can reconfigure each single node. A centralized *Reconfiguration* component has been implemented. It is capable of maintaining information about the network history and the current state of the network. It alerts when some security border condition is reached. For instance, when an application is running for a long time and consequently there is a high probability that the cryptosystem could be broken by brute force attacks, the framework triggers an alert.
Figure II.2.4 describes the interaction among reconfiguration framework's components. The *Event Handler* is designed to deal with a wide range of events, as an example (i) reaching the maximum execution time for an application, (ii) increased rate of battery discharge,

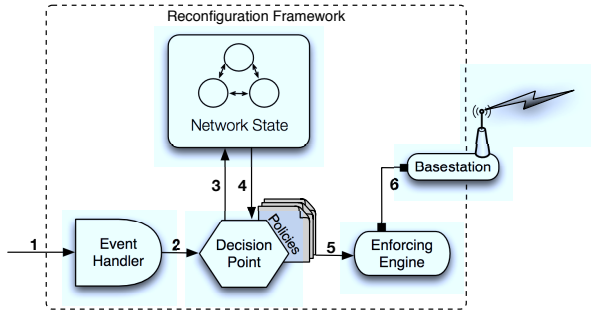


Figure II.2.4 SIREN Control-side

(iii) non-reachability of a node. When an event is detected (phase 1) it is forwarded to the *Decision Point* (phase 2). It collects information about the network state (phase 3 and 4) and according to an administrator policy coordinates the *Enforcing Engine* (phase 5). The *Enforcing Engine* is the component designed to initiate a node reconfiguration. It communicates with each nodes (phase 6) that have to be reconfigured by sending an encrypted reconfiguration command and uses a base station to interface the WSN.

II.2.3.2 SIREN Architecture: node-level implementation details

As regards the internal node structure, available reconfiguration techniques have been improved so as to address the security and selection issues discussed in the previous sections. The Image Switching approach required a considerable implementation effort due to the lack of built in solution in TinyOS. SIREN implements this capability basing on three main components:

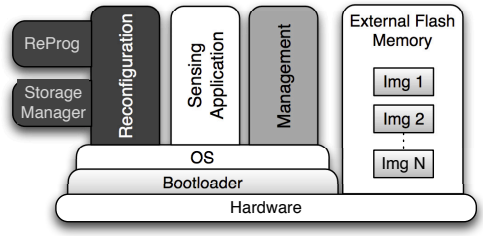


Figure II.2.5 Mote-side Components

- *Bootloader component*: it provides the set of mechanisms to program a node with a built-in software image. Images contain OS, Application Logic and communication API.
- *Reconfiguration components*: to prepare the mote to be reconfigured and save information to restore the node and application execution state after a reconfiguration.

- *Management mote-side component*: it enables the mote to communicate with and receive commands from a control-side workstation.

The **Bootloader** component is *TOSBoot* [40] and it is derived from the Deluge T2 Framework. It is able to read a software image form external flash memory and reconfigure the mote's microcontroller software. The **Reconfiguration** components uses the *ReProg* interface to accept inputs from application dependent logic, such as TinyOS modules, and starts a reconfiguration by interacting with the *Bootloader*. It offers a dedicated API to configure a reconfiguration process. The configuration includes the starting address of the binary image which resides in external memory and the node's network identifier. In this way the selected binary image defines new software configuration (elements of $\mathscr{A}$, $\mathscr{S}$, $\mathscr{L}$, $\mathscr{O}$ sets) and the node identifier enables network reconfiguration. *ReProg* implements indirect information passing through EEPROM to interact with the Bootloader and save some *state* information.

Using this interface, a TinyOS application can simply include the *ReProg* component and implement any possible reconfiguration policy. For instance, to implement the centralized management approach, an handler logic is implemented and deals with incoming control messages from the base station and added into the handler's implementation a call to the *ReProg* interface specifying the binary image to reprogram the mote with. The *ReProg* component is also able to modify the node identifier (in TinyOS it corresponds to the node network address), in this way it is possible to easily vary the network topology (and mitigate different types of attacks). There also is a component dealing with logic names resolution:the *Storage Manager*. It provides a storage map by which one can obtain the physical address of a program image starting from its logical name (identifier).

Figure II.2.6 illustrates the interaction among components during reconfiguration. Assume that a reconfiguration command has been sent by the base station and a node receives it. The command packet is delivered to the application layer (1), it is decrypted and verified (2). The configuration information is extracted (3) and the storage manager component resolves the software image starting address in external memory (4). The resolved address is used to fire the reconfiguration process: this is done by invoking the ReProg component and passing the external memory address of the application with which one want to reprogram the mote with (5). Now the reconfiguration feasibility is checked by measuring the battery energy. If a sufficient amount of energy is present the reconfiguration can take place. All the information needed by the bootloader to reconfigure the mote and/or to preserve the node and application execution state is written into a static location in EEPROM (6). In this way the bootloader knows where to fetch the new application starting address and reads the node's network identifier. Moreover, all the variables to be restored after the reconfiguration are saved in another EEPROM location. On completion, the function to reboot the mote is called. Subsequently, the microcontroller boots up and, according to the information stored in EEPROM, reconfigures the node. According to the enforced ECC libraries, the

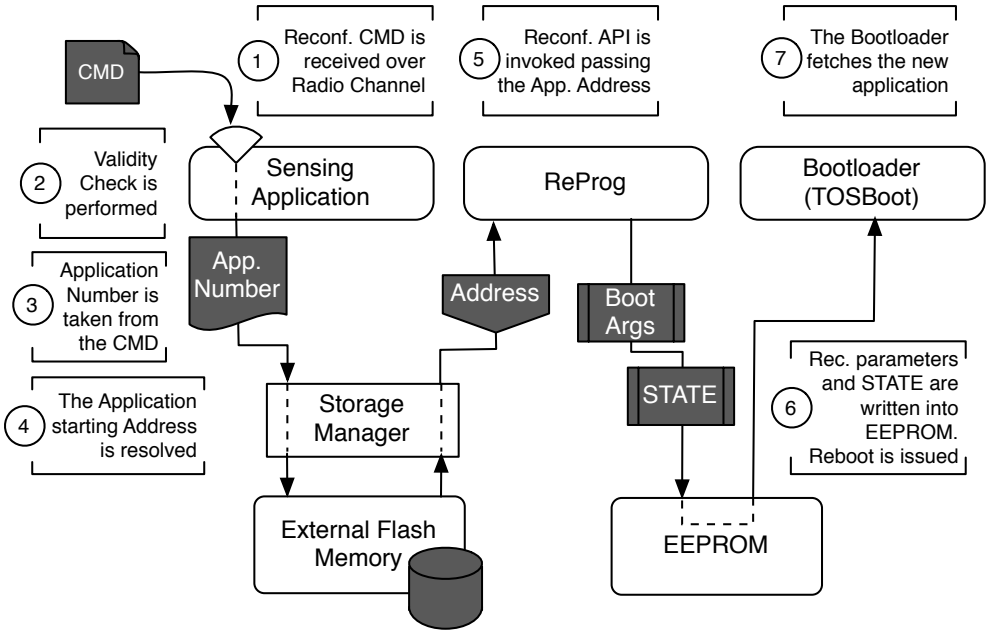


Figure II.2.6 Reconfiguration mote's data flow

application tasks restart and a new set of cryptographic keys are negotiated among the involved neighbours.

II.2.4 Evaluation of the approach

In this section, experiments to demonstrate the feasibility of the approach and evaluate the overall performance of the proposed moving surface strategy are reported. The goal is to evaluate the architecture performance on a specific target technology. The reference architecture is *Telos Rev.B* platform which is built on the TI MSP430 microcontroller with 10KB RAM, 48KB ROM and 1MB external flash, an integrated on board antenna, a 16-bit data bus and integrated temperature, light and humidity sensors. A simple testbed has been considered where a basestation can communicate with a single node in a single hop; the basestation and the remote mote are placed at a fixed distance and in a controlled environment in order to limit network interferences. The monitoring application used in the experiments is able to access to different sensors and to provide message signing and encryption/decryption with two different secure configurations based on TinyPairing and WM-ECC libraries.

To illustrate the feasibility of the approach, different experiments have been carried out to directly compare SIREN with Deluge's reconfiguration approach. In particular, the evalua-

tion regards the impact of reconfiguration on the interval of time the node will be down and the resource consumption:

- the dissemination and reprogramming time;
- static memory footprint;
- energy consumption;

The results of the first experiment confirm what expected, and regards comparing the performance between Deluge dissemination and reprogramming times against SIREN reprogramming time. As said, SIREN supports pre-loading of different images on a mote so the heavy time spent by Deluge to disseminate the images is not wasted in the SIREN framework. A basestation forwards messages between the workstation and the WSN's mote. Indeed, Deluge dissemination time is dependent on number of hops to cross before delivering the image to all nodes [41] so, the chosen testbed with only one hop, represents the best configuration for Deluge (higher performance). Furthermore, Deluge's time-to-completion depends on image size that involves the number of pages to disseminate (in its basic implementation each page is 1104 byte). So, the comparison has been carried out on Deluge's best case against SIREN. Our testbed is opportune for Deluge characteristics standing to the charts presented in [41]. Figure II.2.7 reports numerical results. The time needed

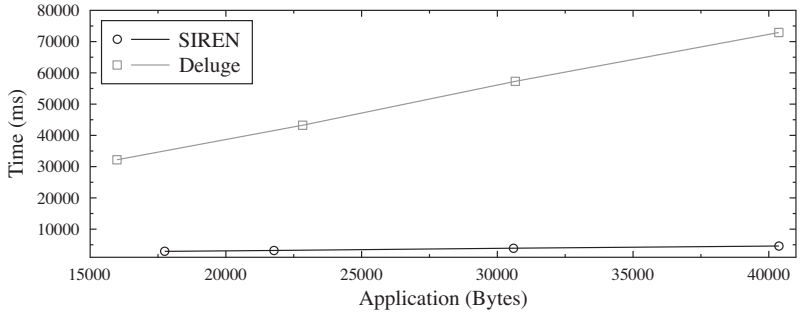


Figure II.2.7 Deluge and SIREN dissemination and reprogramming time

to reconfigure a node with applications of different sizes has been measured. To provide significant data an automatic reconfiguration of each application has been done fifteen times. The plot reports average values (the variance is less than five percent) and one can easily relate the reconfiguration time to the number of bytes that must be programmed. As expected, by avoiding the image dissemination, SIREN performance are slightly affected by the image size. If the analysis had not considered Deluge's best case, as in the Deluge secure dissemination approach [33], his performance are even worse due to page size increment and the need to complete integrity checks on each page before forwarding them to other nodes. As for the reprogramming time itself, there are no significant differences between the two frameworks as they use the same TinyOS bootloader component (TOSBoot).

Moreover a measure of the required time to reconfigure a node with both the WM-ECC application and TinyPairing is here reported. The node has been configures so as to equip both applications into mote's external flash memory. An application enabled with SIREN reconfiguration capabilities was also installed on the nodes to perform the reconfiguration task. Each application have been reconfigured fifteen times. The reconfiguration time is almost constant; the SIREN average time for the WM-ECC application takes 4982 ms and with TinyPairing it takes 5111 ms, applications size are about 45KB. This result is incomparable with the Deluge approach as it takes about 72 seconds to disseminate and reprogram a generic application of about 40KB (see Figure II.2.7).

On the other hand, to enable SIREN, one need to pre-load all configurations on board of each node and the total amount of memory available may represent a limitation on the approach. The second experiment evaluated the memory footprint of the two frameworks. The memory footprint is here reported for: each application with a security library on its own, each framework on its own, and each framework with the secure applications (as WM-ECC footprint plus SIREN footprint); results are reported in Table II.2.1. It is important to notice that the space needed by the framework with the secure applications is not the rough sum of single application's footprints due to operating system module aliasing: aliased modules are included only once.

Application	ROM bytes	RAM bytes
WM-ECC	22302	1813
TinyPairing	25294	933
SIREN	27198	1144
Deluge	38882	1644
WM-ECC + SIREN	42022	2402
TinyPairing + SIREN	45214	1524
WM-ECC + Deluge	gt. 48000 (!)	
TinyPairing + Deluge	gt. 48000 (!)	

Table II.2.1 Memory Footprints

The Deluge's footprint is evaluated by measuring the *GoldenImage* application, which is delivered with Deluge library. This application includes the network module to implement reception and dissemination of configuration (over the air reprogramming). Due to Deluge footprint, when combining it with a security library (both WM-ECC and TinyPairing), the resulting application is greater than maximum ROM memory space available in Telos B mote (48KB), making Deluge not suitable for complex security applications.

Battery consumption is another key factor to demonstrate the feasibility of the proposed framework. Indeed, it is known from the literature and WSN nodes data-sheets that the most power is spent during message transmission and reception. As a direct consequence,

the Deluge dissemination approach is not suitable for the MTD especially if one thinks to high reconfiguration frequencies. The energy consumption estimation has been carried out by measuring the battery depletion using the on board analog-to-digital converter. In the last experiment, it has been evaluated how many consecutive reconfigurations are possible before node unavailability (complete battery consumption).

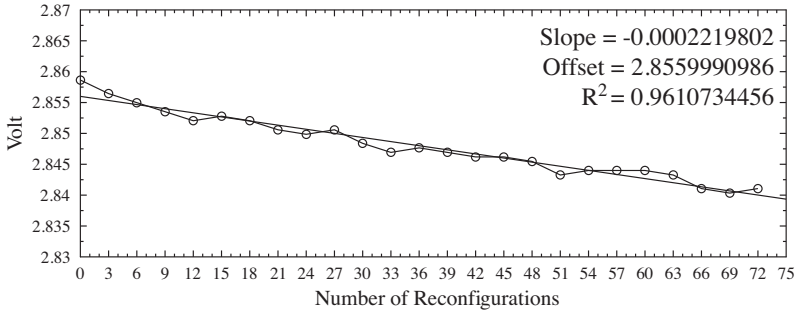


Figure II.2.8 SIREN energy consumption

In Figure II.2.8, the battery level is plotted; at the beginning, with a new battery pack, the voltage is around 2.8V then, for increasing number of reconfiguration accomplished. The fitting curve shows that the consumption trend is similar to a linear trend. Evaluating the coefficient of determination R^2, one can predict future outcomes on the basis of measured data; R^2 is about 0.96 and and so the real data fits well the linear trend in the range of 2.8V. This information can be used by a system designer to define the reconfiguration strategies and get how many reconfiguration tasks could be issued during a mission (WSN mission-time).

To conclude, the experimental results showed how SIREN offers good performance from all point of views. Indeed, the possibility to preload different applications on board lets the reprogramming approach be more efficient and secure than others. Thanks to its architecture, it is possible to really enforce Security Adaptation and Moving Target Defense at monitoring infrastructure level of a cyber-physical system.

CHAPTER II.3

Advancing WSN Physical Security Adopting a TPM-based Architecture

Cyber-physical systems are usually distributed system deployed in broad areas and remotely managed as shown so far. This Chapter focuses on node physical protection and proposes and hardware solution that can be coupled with the software framework presented in the previous Chapter. Being able to provide physical security is a major concern because due to the nature of CPSs, the sensing and actuating infrastructure typically operate unattended or physical surveillance may be impractical as a consequence of geographic distribution. Embedded nodes are subject to a hostile outdoor environment and are highly susceptible to physical attacks [61]. Being exposed to this kind of attack is extremely dangerous as a savvy attacker may be able to extract cryptographic keys from the memory, alter the software running in the sensor node and subsequently, inject malicious traffic into the network. Systems that lack of physical protection for the software and the hardware, may put sensitive application, data and the integrity of the overall system at risk. Physical security should be considered at the beginning of embedded devices development in order to harden the architecture of embedded nodes. A lot of effort has been made to secure communication among nodes, [12, 19]. On the other hand, this Chapter focuses on ways to mitigate physical attacks by re-designing embedded node's architecture to ensure software and hardware integrity. In order to discuss and present a feasible architecture further discussion will refer to an FPGA node implementation as nowadays technology, thanks to power efficiency and low cost, allows to adopt FPGA as hardware core for nodes architectures [9]. The architecture

being presented is based on Xilixn Zynq 7000 FPGA/SoC family and will provide secure primitives to asses the hardware and software integrity as well as offering cryptographic operation accordingly to the Trusted Platform Module (TPM) standard that is discussed in the next section.

II.3.1 Trusted Platform Module and its adoption for CPS's monitoring infrastructures

The Trusted Platform Module (TPM) is a specification for a secure cryptoprocessor defined by in the ISO/IEC 11889 of the Trusted Computing Group (TCG) [36]. This specification defines the components and operations which a cryptoprocessor has to implement in order to build a chain of trust. A chain of trust is a trust relationship between entities that is established by validating each component of hardware and software from the bottom up to the top. It is intended to ensure that only trusted software and hardware can be used. According to the specifications, the root of the chain of trust has to be designed as secure and *tamper resistant* hardware platform. Figure II.3.1 abstracts the functions and modules embedded into a TPM.

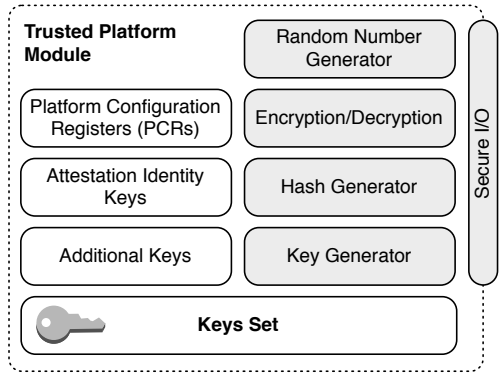


Figure II.3.1 Trusted Platform Module

The TPM offers cryptographic facilities based on an endorsement key (referred in the future as master key) or on another trusted key derived from it (obtained through Key Derivation Function). The master key is embedded into a OTP memory[1] at the moment of its manufacture and cannot be read outside the TPM. Using TPM, one can trust all the cryptographic operations that are performed within the platform, such as Random Number Generation,

[1] OTP: One Time Programmable Memory

Hashing, Key Generation and Encryption/Decryption. Some of the functions of a TPM include:

- creating a hash summary of a system hardware/software configuration. This offers a mechanism to assess if the hardware or software has been changed after deployment.
- encrypting data using the endorsement key or other temporary keys generated using the TPM facilities.
- storing a limited amount of data into a tamper resistant memory.

In the context of WSN domain, TPM can be useful to mitigate some threats to a node's physical security.

Node Theft A node can be physically compromised. However, it cannot be re-programmed or manipulated in a way that it is undetectable by a remote verifier.

Software Theft It is not possible to extract the node's software image; it is encrypted and signed with a key directly derived from the master key.

Privacy of the data Data can be easily encrypted or decrypted using the AES/RSA engine and the keys stored into the TPM anti-tamper memory. The TPM allows the generation of multiple session keys, enabling to easily change the cryptographic keys during the node life-cycle.

Cloning Node cloning is not possible because it is not feasible to extract the keys. Moreover, even in the extreme case of an attacker knowing a key, it is not possible to inject it into the TPM.

Alteration Adopting a hash summary of the node configuration, it is possible to recognize changes made to the node hardware and software. These security checks are generally made before the node start up. If a component or the software is altered, the hash verification will fail and the system will stop its execution.

	Integrity Check	**Data Encryption**	**Key Protection**	**Secure Storage**
Node Theft			×	
Software Theft		×		
Privacy Violation		×	×	
Node Cloning	×		×	
Alteration	×	×	×	×

Table II.3.1 Threat coverage matrix using TPM protection mechanisms

Table II.3.1 reports how TPM features (columns) cover the threats (rows) discussed above. As for the integrity check, the TPM can assure the platform integrity: together with the boot phase, the TPM forms a root of trust and it can check, by using its internal

Platform Configuration Registers (PCRs), if platform components have been altered. The application running on the node can exploit the TPM crypto-facilities to provide integrity, confidentiality and authentication in a trusted chain. The crypto-keys are only used inside a secure perimeter and generated using the TPM key generator: the generated keys are stored securely and they never leave the TMP chip. As for the secure storage, some platform information can be securely stored within the secure perimeter of the TMP. For instance the memory pages hashes can be stored and checked periodically in order to avoid memory alteration.

Most of the literature discusses how TPM facilities can be used to ensure trustworthy relationship between nodes, how to implement remote attestation and study key establishment and management between nodes, the focus is on how to exploit TPM advantages but not on how to design an architecture enabled with its facilities. Authors in [81] provide an algorithm for addressing privacy issues in WSN exploiting the features of Trusted Computing. They focus on the case of a heterogeneous and hierarchical network. The nodes are grouped and each group has a cluster head equipped with trusted computing technology. Their goal is to achieve user query privacy by means of remote attestation and trust relationship between cluster heads. They neither present a practical architecture nor discuss the benefits of adopting TPM for node security. Authors in [49] show a lightweight attestation mechanism to check the status of cluster heads. The basic idea is that a cluster node can verify the integrity of cluster heads using the TPM as a trust anchor.

The design and implementation of a trusted sensor node with enhanced security facilities is discussed in [39]. Authors present *trustedFleck*, a proprietary node implementation which uses a TPM chip to extend the capabilities of a standard wireless sensor node. They focus on describing the advantages introduced by the architecture for enforcing messages and system integrity, confidentiality and authenticity. Moreover, they compare their approach with a software implementation of cryptographic operation such as TinyECC. They also describe examples of using trustedFleck for symmetric key management, secure software updates and remote attestation.

The thesis goal, as well as publication [10], differs from [39] in the sense that the discussion regards the feasibility of implementing a wireless sensor node enabled with an FPGA implementation of TPM, not a TPM chip.

II.3.2 Implementing a TPM-enabled node using FPGA

Since the TPM specification requires a secure hardware perimeter to securely store/generate the keys and to compute the crypto functions. The Xilixn Zynq 7000 FPGA/SoC family can be a perfect starting point to meet these requirements. Zynq architecture is a System on Programmable Chip, based on the ARM family. The processing system (PS) is static

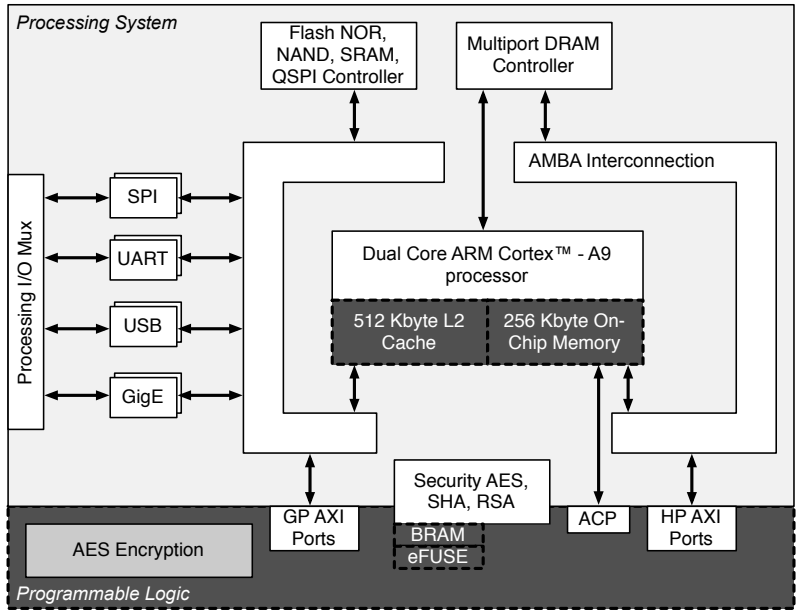


Figure II.3.2 Overview of Zynq internal architecture

(not programmed on FPGA) and can operate independently from the FPGA programmable logic (PL). The core of the PS is an ARM dual-core Cortex A9 with 32/32 KB I/D caches, 512 KB L2 cache and 256 KByte On-Chip Memory (OCM). Some peripherals are available to the processor, including standard serial buses (such as USB device OTG, SPI, UART, etc.), memory controllers (DDR3, NAND, NOR, etc.), cryptographic accelerators (AES and SHA), high throughput interfaces to the PL and the Device Configuration (DevC) . The DevC peripheral allows the processor to load full/partial bitstreams to the FPGA programmable logic.

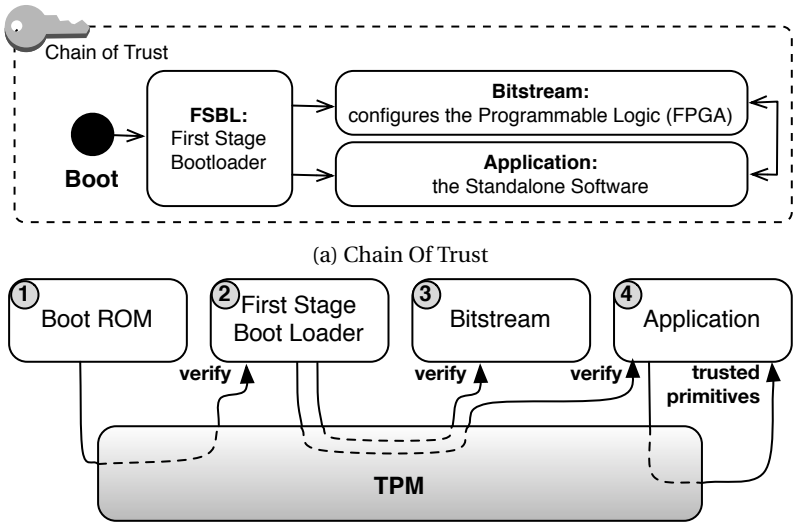
The Zynq hardware natively supports the deployment of secure systems. Embedded hardware blocks offer decryption engines, authentication, keys storage and unique device identity. These blocks reside in a secure hardware perimeter in order to guarantee the anti-tampering property, trustworthiness and integrity to protect the system from the power-up and throughout its life cycle. The chip provides two kinds of secure storage which can be used to burn the master key: BBRAM and eFUSE. Both register arrays can host a 256 bits AES key; however the former is a volatile memory and needs a backup battery to keep the key when the power supply is disconnected. In both the solutions, there is no way to read the keys once they are written. In fact all the on chip memories (OCM, L1 and L2 caches, PL configuration memory, BBRAM and eFUSE) reside within the security perimeter of the FPGA [68].

Figure II.3.2 illustrates the Zynq internal organization along with the secure perimeters (in dotted rectangles). As Zynq does not provide any encryption module (but only a decryption one), an AES 256 bits accelerator with only encryption feature has been added into the programmable logic.

Next subsection discuss the boot process on the Zynq with regards on how it implements the trusted chain.

II.3.2.1 Building the chain of trust on Zynq architecture

As described in [68], secure booting commences by configuring the processing system (Figure II.3.3b). The ARM processor starts executing the code residing in the on-chip ROM. This code is provided by Xilinx and can not be modified. It performs a simple initialization of the L1 cache and the bus system; also it contains low-level drivers to be able to retrieve the user defined software from one of the external memories. While secure booting, if the key source specified in the eFUSE and the key source specified in the boot header do not match or if the there is an illegal boot mode, the device will transit into a *secure lock down state*. This ensures that a successful boot implies that the chip is in a secure state. Then, the First Stage Boot Loader (FSBL) partition header is parsed in order to retrieve the location of the boot code and the actual boot configuration. These information can be stored in two modes: (i) encryption disabled, (ii) encryption enabled. In the first case, the code can be directly executed without the needing of a decryption operation (insecure boot).
In the latter case, the FSBL is passed to the AES/SHA engine before being loaded into the on-chip SRAM. The data is then decrypted, authenticated, and stored in the on-chip RAM. The processor then will continue with the boot operations. The FSBL has to load: (i) the bitstream, which will configure the PL with the user peripherals and custom modules, and (ii) the user application. In the secure mode, bitstream and each software partition is encrypted with AES and signed by means of RSA. Enciphered bitstreams can be loaded into the PL by using the DevC (Device Configuration). The decrypted bitstream is directly pushed into the PL configuration memory, it will never be read if the eFUSE registers are configured in the write-only mode. As for the software, once the partitions are verified and ready to be loaded, the allocation process takes place. Where to load the *.text* and *.data* areas depends on the memory design specified in the linker script. In case these areas are loaded into the OCM, the security properties are automatically inherited and guaranteed trough the whole life-cycle. Otherwise, if they are loaded into memories outside the secure perimeter, such as external DDR RAM, it is necessary to adopt an application level mechanism to ensure data integrity during the application life-cycle.



(a) Chain Of Trust

(b) TPM adoption

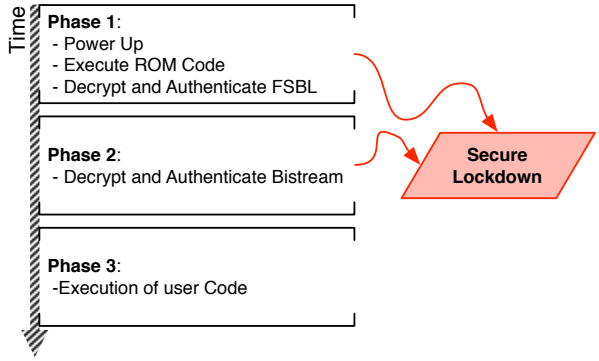
Figure II.3.3 TPM and Chain Of Trust

Figure II.3.3a reports the chain of trust: using the TPM primitives, an application running on the processing system can dynamically configure the PL with partial bitstreams, preserving the chain of trust (Figure II.3.3b).

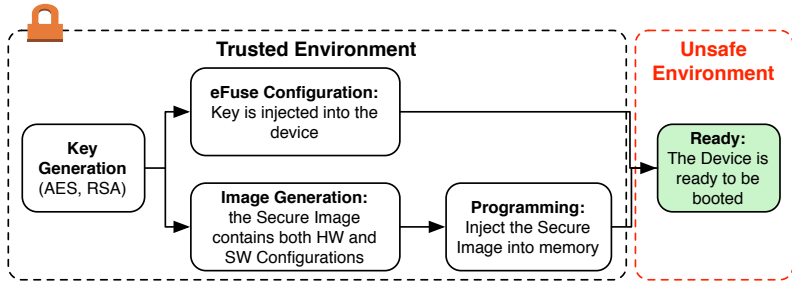
II.3.3 Security considerations about the Chain of Trust

The Zynq family has an embedded security mechanism implemented to protect the system during the boot process. In fact, if one of the boot cryptographic function fails, the system immediately reaches a stable state of *lock down*, only a reset operation can reinitialize the node and perform again its verification (Figure II.3.4a). Moreover, the Zynq allows the disabling of write access to critical system control registers and to key controller registers [79]. It is a common practice to lock the state of the system through the First Stage Boot Loader: the lock-down approach improves overall system security by prohibiting the running software from modifying critical system registers (authorization).

It is important to remember that the design and development of such architecture must be security-aware. Figure II.3.4b contains the development/deployment process of a Zynq-based system and highlights the security domains in which each operation has to take place. First, the master key is generated and injected into the eFUSE register. Then, the First Stage Boot Loader, the bitstream and the software are merged in order to create a system image. This image is then encrypted and signed (by means of RSA) using the generated key. The encrypted image is then loaded into one of the Zynq memories and the system is ready to



(a) Stages during secure booting and execution of user code.



(b) Development and deployment of a Zynq system

Figure II.3.4 Trusted environment and secure boot

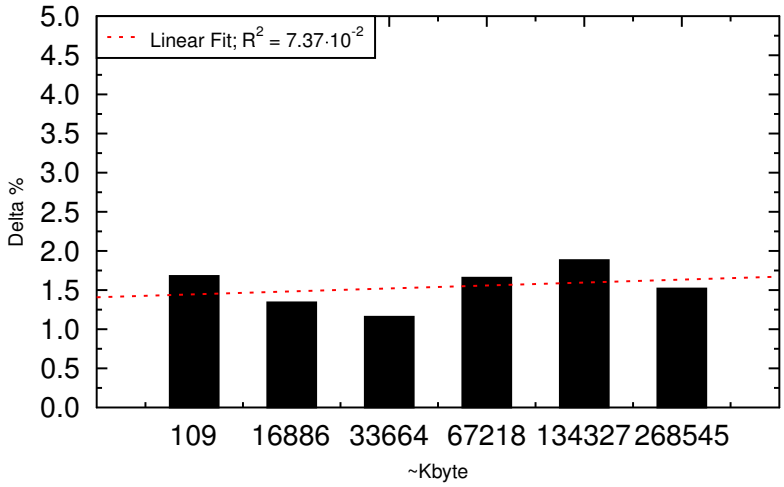
boot. The above steps must take place in a secure and trusted environment to avoid key leakage (e.g. USB and JTAG are unsecured channels). Also, all systems and components involved in this process have to be audited in order to be sure that no sniffers or other malicious components have been injected. Once the system is configured, it is ready for deployment in an unsafe environment.

With regard to temporary session keys, they can be stored in either of the two secure perimeters: the OCM or directly inside the AES block. In both cases, the key can be dynamically changed rewriting the OCM memory or through *Self Dynamic Partially Reconfiguration* to update the AES block in PL. Through this mechanisms for keys protection, it is also possible to improve the security of remote hardware and software reconfiguration. For instance, this proposal guarantees a tamper resistant storage for different node configurations. In this way, it is possible to extend mechanisms such as [12] (presented in the previous Chapter), in order to ensure physical security and making more feasible to implement effective moving target defense techniques for embedded node.

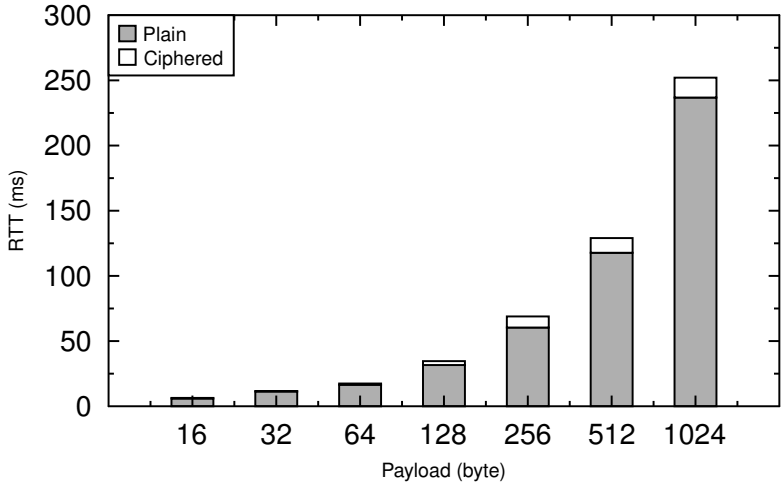
II.3.4 Performance

This section reports the results of an experimental campaign that was conducted to compare a non-secure system to a TPM-enabled one, the evaluation focuses on the latency introduced by the approach for:

- boot time against software image footprint;
- time to transmit and receive ciphered data packets against packet size[2].



(a) Difference in Secure and Normal Boot



(b) AES Communication Overhead

Figure II.3.5 Performance Results

[2] In the test environment the packet loss is about 2.3%

To measure the boot time, an external controller has been used. It automatically shuts down and restarts the Zynq board. As soon as the Zynq system starts-up, it signals the controller through a dedicated interrupt line. This allowed us to perform repeated measures of the boot time. Figure II.3.5a presents the percentage of delta difference between the time to boot a non-secure system and the time to boot its secure version as a function of increasing software image size. It can be seen that the latency introduced by the secure boot is really small and is contained between 1.5% and 1.8%. Figure II.3.5b shows the round trip time for the communication among two nodes with increasing packet size. As one can notice, the performance degradation due to the encryption/decryption process is negligible for packet sizes less than 128 byte. Also as the packet size increases, the cryptographic operations only have a small affect on the performance: for the worst case of a 1024 bytes payload, the delay introduced is of 6.44%. This shows the advantages of a special purpose hardware to implement cryptographically secured communication among wireless sensor nodes.

CHAPTER II.4

Summary

In this part of the thesis, mechanisms to effectively and efficiently improve security and resiliency of a CPS's monitoring infrastructure have been presented and discussed with regard to the challenging case of wireless sensor networks. Both the proposed solutions focused on integrating security into the core architecture design, as opposed to hardening a design after its deployment.

In Chapter II.2 a software level mechanism has been presented in order to make it possible to exploits the advantages of novel security through adaptation techniques such as moving target defense. To augment software security for wireless embedded nodes one need to cope with different security constraints. Moving Target Defense as well as security through adaptation and reconfiguration seems very promising. Chapter II.2 analyzed different security solutions and reconfiguration mechanisms for WSN and, more in general, for embedded devices. It has been shown how different reconfiguration strategies may be implemented to enhance security and address limitation of current approaches. At this aim, a reconfiguration framework has been proposed and it has been demonstrated how it overcomes the limits of available reconfiguration tools in order to make reconfiguration feasible for security purposes. The evaluation of the framework was focused on feasibility and performance points of view. Even though, the reconfiguration implies a certain interval of unavailability of a node, it has been shown that this time can be negligible for certain kind of monitoring applications. Moreover, the designer has been provided with the necessary information to implement reconfiguration plans by estimating the cost in terms of resources required. On the other hand, the quantification in terms of security level benefit deriving

from the adoption of Moving Target Defense and security reconfiguration is still an open issue. Efforts to quantitatively estimated the security improvement are a future and ongoing work.

Chapter II.3 proposed an architecture with advanced security capabilities through a special purpose hardware implemented in FPGA technology. As regards physical security for embedded nodes, Chapter II.3 described an efficient and effective approach to defeat and mitigate physical attacks to the sensing infrastructure of Cyber Physical Systems. Attacks whose goal is to alter, tamper or extract information by directly probing an embedded node, will have to deal with a new security layer which protects both the node's hardware and can offer guarantees about software genuineness. Disrupting physical attacks makes the injection of malicious traffic, as well as the information leakage, more difficult or even unfeasible. To this end, an FPGA-based hardware architecture has been presented. It is compliant with the TPM standard and offers security primitives to asses node integrity and perform efficient AES encryption through hardware support.

It is important to notice that the two proposed solutions can be merged in order to exploit advantages of both physical and adaptive security. This fusion is an ongoing work to make it possible to efficiently and securely reconfigure the entire embedded node stack with the SIREN approach.

PART III

Securing the Processing Infrastructure

CHAPTER III.1

Introduction and Background

Today's approach to cyber defense is governed by slow and deliberative processes such as security patch deployment, testing, episodic penetration exercises, and human-in-the-loop monitoring of security events. This is also the case of CPS's processing infrastructures. Adversaries can greatly benefit from this situation, and can continuously and systematically probe target networks with the confidence that those networks will change slowly if at all. Cyber attacks are typically preceded by a reconnaissance phase in which adversaries aim at collecting valuable information about the target system, including network topology, service dependencies, and unpatched vulnerabilities. As most system configurations are static – hosts, networks, software, and services do not reconfigure, adapt, or regenerate except in deterministic ways to support maintenance and uptime requirements – it is only a matter of time for attackers to acquire accurate knowledge about the target system. This will eventually enable them to engineer reliable exploits and pre-plan their attacks. Self interested attackers will spend a great effort into discovering and analyzing CPSs' processing infrastructures, due to their critical nature in controlling and monitoring environment, industrial process and power grids. To this aim they are interested into identifying which are the components of the processing infrastructure, which operating systems are being used, which services are offered by networked devices and how these systems are connected one to the other.

Specifically, operating system fingerprinting aims at determining the operating system of a remote host in either a passive way, through sniffing and traffic analysis, or an active way, through probing. Similarly, service fingerprinting aims at determining what services

are running on a remote host. To address this problem, many adaptive techniques have been devised to dynamically change some aspects of a system's configuration in order to introduce uncertainty for the attacker. Significant work has been done in the area of Adaptation Techniques (AT) and Adaptive Cyber Defense (ACD), which includes concepts such as Moving Target Defense (MTD), as well as artificial diversity and bio-inspired defenses, to the extent they involve system adaptation for security and resiliency purposes. Essentially, a number of techniques have been proposed to dynamically change a system's attack surface by periodically reconfiguring some aspects of the system. In [54], a system's attack surface has been defined as the "subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack". Intuitively, dynamically reconfiguring a system is expected to introduce uncertainty for the attacker and increase the cost of the reconnaissance effort. However, one of the major drawbacks of current approaches is that they force the defender to periodically reconfigure the system, and this may introduce a costly overhead for legitimate users, as well as the potential for Denial of Service conditions. Additionally, most of the existing techniques are proactive in nature or do not adequately consider the attacker's behavior.

Adaptation Techniques for networked systems As already said in Chapter II.2, Moving Target Defense (MTD) defines mechanisms and strategies to increase complexity and costs for attackers [35]. MTD approaches aiming at selectively altering a system's attack surface usually involve reconfiguring the system in order to make attacks' preconditions impossible or unstable.

Dunlop et al. [32] propose a mechanism to dynamically hide addresses of IPv6 packets to achieve anonymity. This is done by adding virtual network interface controllers and sharing a secret among all the hosts in the network. In [31], Duan et al. present a proactive Random Route Mutation technique to randomly change the route of network flows to defend against eavesdropping and DoS attacks. In their implementation, they use OpenFlow Switches and a centralized controller to define the route of each flow. Jafarian et al. [42] use an IP virtualization scheme based on virtual DNS entries and Software Defined Networks. Their goal is to hide network assets from scanners. In Chapter 8 of [45], an approach based on diverse virtual servers is presented. Each server is configured with a set of software stacks, and a rotational scheme is employed for substituting different software stacks for any given request. This creates a dynamic and uncertain attack surface.

These solutions tend to change the system in order to modify its external attack surface. On the other hand, the external view of the system is usually inferred by the attackers based on the results of probing and scanning tools. Starting from this observation, the approach presented in this thesis consists in modifying system *responses* to probes in order to expose an *external view* of the system that is significantly different from the actual attack surface,

without altering the system itself.
Reconnaissance tools, such as *nmap* or *Xprobe2*, are able to identify a service or an operating system by exploiting the protocol's *ambiguities* analyzing packets that can reveal implementation specific details about the host [53, 78]. Network protocol *fingerprinting* refers to the process of identifying specific *feature* of a network protocol implementation by analyzing its input/output behavior [71]. These features may reveal specific information such as protocol version, vendor information and configurations. Reconnaissance tools store known system's features and compare them against the scan responses in order to match a fingerprint. Watson et al. [78] adopted protocol scrubbers in order to remove TCP protocol ambiguities. They used a *fingerprint scrubber* to restrict an attacker's ability to determine the operating system of a protected host. Moreover, some proof-of-concept software and kernel patches have been proposed to alter a system fingerprint [27], such as IP Personality and Stealth Patch.
Honeypots have been traditionally used to try to divert attackers away from critical resources. Even though the approach here proposed and honeypots share same goal, they are significantly different: honeypot-based solutions introduce vulnerable machines in order to either capture the attacker [1] or collect information for forensic purpose [23], the proposal, instead, does not alter the system at all. The aim is to divert attackers by manipulating their view of the target system and forcing them to plan attacks based on incorrect or inaccurate knowledge. As such, these attacks will likely fail.

To the best of my knowledge, this thesis and publication [4] are the first to propose an adaptive mechanism for changing the attacker's view of a system's attack surface without reconfiguring the system itself. To this aim, Chapter III.2 advances the state of the art in adaptive defense by looking at the problem from a control perspective and proposing a graph-based approach to manipulate the attacker's view of a system's attack surface. To achieve this objective, the notion of system view and distance between views is formalized. A principled approach is then defined to manipulate responses to attacker's probes so as to induce an external view of the system that satisfies certain properties. In particular, it is proposed an efficient algorithmic solutions to different classes of problems, namely (i) inducing an external view that is at a minimum distance from the internal view while minimizing the cost for the defender; (ii) inducing an external view that maximizes the distance from the internal view, given an upper bound on the admissible cost for the defender. Experiments conducted on a prototypal implementation of the proposed algorithms confirm that the approach is efficient and effective in steering the attackers away from critical resources.

The results and formalization of Chapter III.2 are then used as a reference for Chapter III.3 where a practical approach to defeat an attacker's fingerprinting effort through deception is proposed. To defeat OS fingerprinting, the outgoing traffic is manipulated so that it resembles traffic generated by a host with a different operating system. Similarly, to

defeat service fingerprinting, services banner are modified by intercepting and manipulating certain packets before they leave the host or network. Details on how to achieve these results are presented and experimental results show that the approach can efficiently and effectively deceive an attacker.

CHAPTER III.2

Manipulating the Attacker's View to defend the Processing Infrastructure

This Chapter aims at advancing the state of the art in adaptive defense by looking at the problem from a control perspective and proposing a graph-based approach to manipulate the attacker's perception of a cyber-physical system processing infrastructure's attack surface. To achieve this objective,the notion of system view as well as the notion of distance between views is firstly formalized. The formalization refers to the attacker's view of the system as the *external view* and to the defender's view as the *internal view*. A system's attack surface can then be thought of as the subset of the internal view that would be exposed to potential attackers when no deceptive strategy is adopted. Starting from this definitions, a principled yet practical approach is defined to manipulate responses to attacker's probes so as to induce an external view of the system that satisfies certain properties. In particular, here is proposed an efficient algorithmic solutions to different classes of problems, namely (i) inducing an external view that is at a minimum distance from the internal view while minimizing the cost for the defender; (ii) inducing an external view that maximizes the distance from the internal one, given an upper bound on the admissible cost for the defender.

Figure III.2.1 shows how standard and Adaptive Cyber Defense (ACD-based) controllers can be adopted in order to alter the attacker's perception of the attack surface. State of the art solutions aim at dynamically reconfiguring the system in order to create a moving target for the attacker as it has been shown in Chapter II.2 with regards to monitoring

infrastructures. Both internal and external system variables are used to monitor the system's state and enforce a moving target objective through an ad-hoc *controller*. On the other hand, the approach here proposed aims at transparently altering the attacker's perception. The attacker's knowledge of the system is inferred from the system's observable variables (e.g., ports, packets, web-pages content, DNS records). In order to gather additional information, the attacker probes the system. The proposed *ACD controller* identifies such probes and controls the information exposed through the system's observable variables by crafting adequate probe responses to bias the attacker's view of the system.

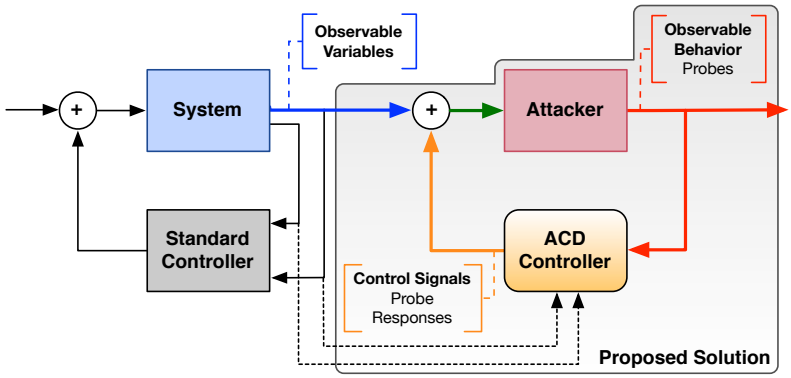


Figure III.2.1 Control Prospective

There are two key advantages comparing to traditional approaches: (i) one can introduce uncertainty for the attackers without actually changing the system's configuration, thus minimizing the overhead; (ii) one can deceive the attackers and steer them away from critical resources rather than simply introducing uncertainty and forcing them to use a random attack strategy. The Section relative to the evaluation of the approach, on a prototypal implementation of the proposed algorithms, confirms that this approach is efficient and effective in steering the attackers away from critical resources.

III.2.1 Attacker Reconnaissance of a target system

This section defines a threat and attacker reference model in order to set the scope of further discussions. The adversary is assumed external and attempting to infer a detailed map of the target network. The threat model assumes that the attacker will use reconnaissance tools such as *nmap* [53] to discover active hosts in the network. The information attackers aim at discovering includes operating systems, exposed services and their version, network layout and routing information. They will then leverage this knowledge to plan and execute attacks aimed at exploiting exposed services. It is also assumed that the attacker will use an

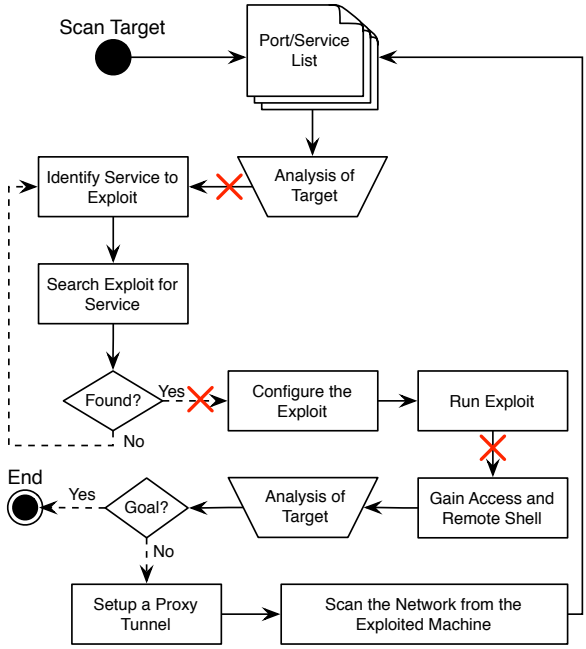


Figure III.2.2 Attacker's strategy flow chart

OS fingerprint technique based on sending valid and invalid IP packets and studying the respective responses without resorting to analysis based on timing. Moreover, the service fingerprinting is limited to the case of TCP probes, as it is the case for common probing tools.

The attacker strategy is depicted in Figure III.2.2. The goal of an attacker is to launch an effective exploit against one of the hosts in the target network. With this exploitation the attacker will gain sufficient privileges to proceed in further analysis of the network. Multiple stages of this attack strategy (marked with a red cross in the figure) can be defeated using the proposed approach. For instance, it will be possible to expose services with no publicly available exploits. On the other hand, for a given service,it will also be possible to expose exploitable vulnerabilities which do not correspond to the actual vulnerabilities of that service.

III.2.1.1 Motivating Example

As a reference example the infrastructure in Figure III.2.3 is considered. It represents the IT infrastructure of a general CPS's processing infrastructure. Some data and reports may be publicly available throughout a public website hosted in the DMZ. The business logic and critical services are implemented in the Intranet and some of these are required to be

accessible throughout the Internet in order to allow other branches to gather and process data as well as perform queries.

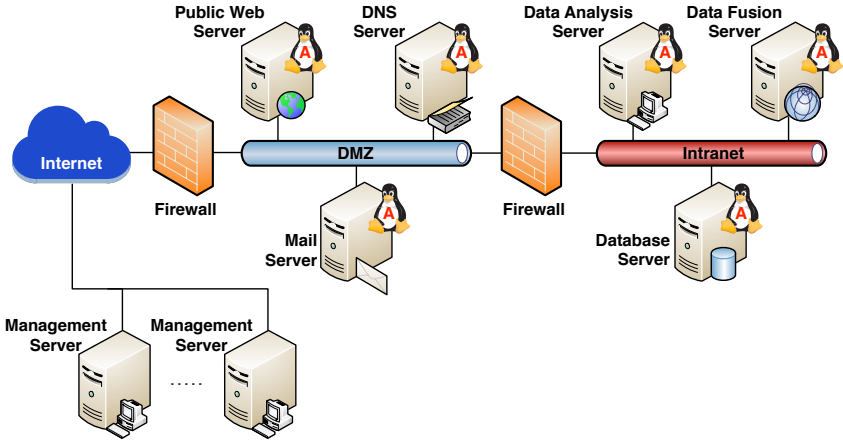


Figure III.2.3 Running Example and System Architecture

The defender goal is to modify the attacker's view of the system and its attack surface. In order to do so, only the system dependent information exposed by protocol ambiguities are modified. By adopting *protocol and fingerprint scrubbers*, like the one presented in the next Chapter and [27, 78], it is possible to alter and filter all the outgoing traffic. Using these techniques and a graph-based strategy to generate different views of the system by repeatedly applying view manipulation primitives, attackers are deceived to induce a different view of the system, such as the one in Figure III.2.4.

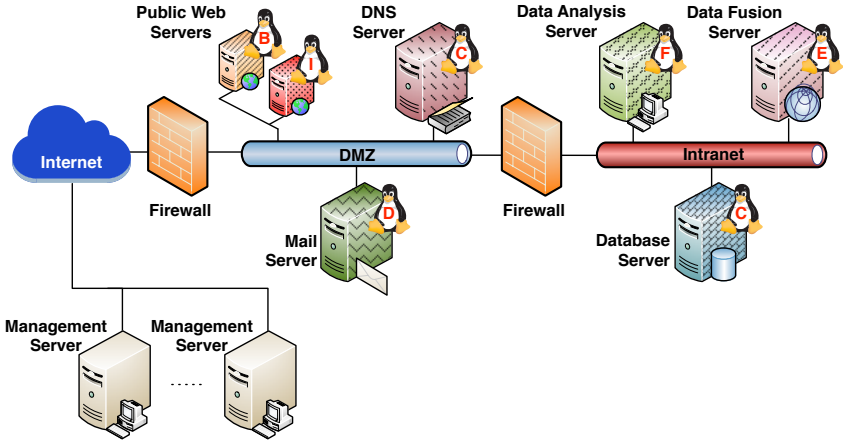


Figure III.2.4 Mutated System Architecture

Figure III.2.4 depicts a *manipulated* service with a different color/texture – compared to Figure III.2.3 – of the machine where it is deployed. A change of an operating system is represented by a different letter on the top right corner of each machine. In this example, applying several manipulation steps to the internal view, one moves from a flat infrastructure, where all the servers have the same operating system, to a view in which each operating system is different from the real one and the others. In the same way, all the services under control are mutated. For instance the *database server* fingerprint is altered so as it will be recognized as an implementation from a different vendor. As for the *public web server*, it will resemble two web servers in a load balancing configuration. To do so the strategy is to mutate, with a certain frequency, both the OS and service fingerprints as well as modify packet level parameters. In this way, one can give an attacker the idea that more than one server is active and the requests are routed between them.

III.2.2 View Model and Problem Statements

In order to achieve the goal of inducing an attacker's view of the system's attack surface that is measurably different from the internal view, it is first required to formalize the notion of view, as well as the notion of manipulation primitive and distance between views (Section III.2.2.1).

III.2.2.1 View Model

In the following, a system is assumed as a set $S = \{s_1, s_2, \ldots, s_n\}$ of devices (e.g., hosts, firewalls), and use Ψ to denote the set of services that can be offered by hosts in S. The defender's and attacker's knowledge of the system is represented by views, as defined below.

Definition 1 (System's View) *Given a system S, a* view *of S is a triple $V = (S_o, C, \nu)$, where $S_o \subseteq S$ is a set of observable devices, $C \subseteq S_o \times S_o$ represents connectivity between elements in S_o, and $\nu : S_o \rightarrow 2^{\Psi}$ is a function that maps hosts in S_o to a set of services.*

Intuitively, a view represents knowledge of a subset of the system's devices and includes information about the topology as well information about services offered by reachable hosts[1].

[1] A more complete definition of view could incorporate information about service dependencies and vulnerabilities, similarly to what proposed in [5].

Definition 2 (Manipulation Primitive) *Given a system S and a set $\mathcal{V}$ of views of S, a* manipulation primitive *is a function $\pi : \mathcal{V} \rightarrow \mathcal{V}$ that transforms a view $V' \in \mathcal{V}$ into a view $V'' \in \mathcal{V}$. Let Π denote a family of such functions. For each $\pi \in \Pi$, the following properties must hold.*

$$\left(\forall V', V'' \in \mathcal{V}\right)\ V'' = \pi\left(V'\right) \neq V' \tag{III.2.1}$$

$$(\forall V \in \mathcal{V})\ (\nexists \langle \pi_1, \pi_2, ..., \pi_m \rangle \in \Pi^m \mid \pi_1\left(\pi_2\left(...\pi_m\left(V\right)...\right)\right) = \pi\left(V\right)) \tag{III.2.2}$$

Intuitively, a manipulation primitive is an atomic transformation that can be applied to a view to obtain a different view.

Example 1 *A possible manipulation primitive is $\pi_{OS_B}(V')$. This primitive transforms a view V' into a view V'' by changing the operating system fingerprint of a selected host[2]. Figure III.2.5 is a graphical representation of the effect of this primitive on the system's view. This primitive can be implemented as defined in [27, 78].*

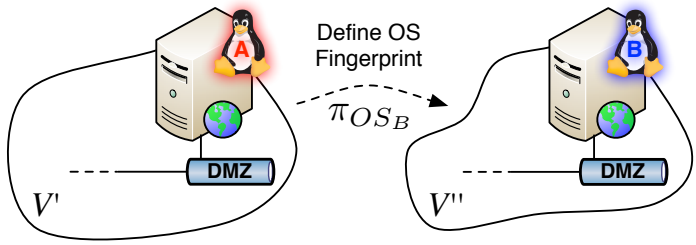


Figure III.2.5 Example of Manipulation Primitive

Definition 3 (View Manipulation Graph) *Given a system S, a set $\mathcal{V}$ of views of S, and a family Π of manipulation primitives, a view manipulation graph for S is a directed graph $G = (\mathcal{V}', \mathcal{E}, \ell)$, where*

- *$\mathcal{V}' \subseteq \mathcal{V}$ is a set of views of S;*
- *$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges;*
- *$\ell : \mathcal{E} \rightarrow \Pi$ is a function that associates with each edge $(V', V'') \in \mathcal{E}$ a manipulation primitive $\pi \in \Pi$ such that $V'' = \pi(V')$.*

The node representing the internal view has no incoming edges. All other nodes represent possible external views.

[2] Each primitive may have a set of specific parameters, which are omit for the sake of notation simplicity.

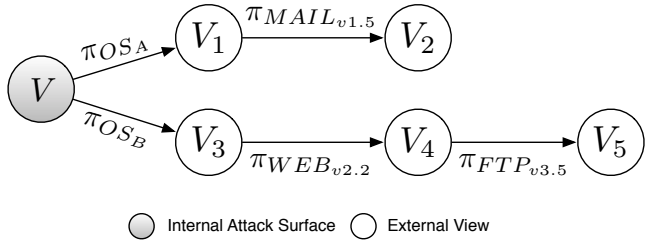


Figure III.2.6 Example of View Manipulation Graph[3]

Example 2 *Figure III.2.6 shows an example of View Manipulation Graph. As one can notice, starting from the internal view V, multiple external views can be derived. After applying any $\pi \in \Pi$, a new view is generated. By analyzing the graph, one can enumerate all the possible ways to generate external views starting from the internal view V of a system S.*

Definition 4 (Distance) *Given a system S and a set $\mathcal{V}$ of views of S, a distance over $\mathcal{V}$ is a function $\delta : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ such that, $\forall\, V', V'', V''' \in \mathcal{V}$, the following properties hold:*

$$\delta\left(V', V''\right) \geq 0 \tag{III.2.3}$$

$$\delta\left(V', V''\right) = 0 \iff V' = V'' \tag{III.2.4}$$

$$\delta\left(V', V''\right) = \delta\left(V'', V'\right) \tag{III.2.5}$$

$$\delta\left(V', V'''\right) \leq \delta\left(V', V''\right) + \delta\left(V'', V'''\right) \tag{III.2.6}$$

Example 3 *The distance can be measured by evaluating the number of elements that change between views. To do so, first consider the difference in the number of hosts between the two views. Then, for each host that is present in both the views, add one if the OS's fingerprint differs and add one if the service's fingerprint changes, that is:*

$$\delta\left(V', V''\right) = \left|V'_{\#hosts} - V''_{\#hosts}\right| + \sum_{h \in Hosts_{V' \cap V''}} \left(1_{OS} + 1_{Service}\right)$$

More sophisticated distance measures can be defined, but this is beyond the scope of this thesis.

[3] π's subscripts denotes generic primitives $\pi \in \Pi$.

Definition 5 (Path Set) *Given a view manipulation graph G, the path set P_G for G is the set of all possible paths (sequence of edges) in G. The k-th path, of length m_k, in P_G is denoted with $p_k = \langle \pi_{i_1}, \pi_{i_2} \ldots, \pi_{i_{m_k}} \rangle$*[4].

Example 4 *Consider the View Manipulation Graph G in Figure III.2.6. In this case, the set of all possible paths is:*

$$P_G = \left\{ \begin{array}{c} p_1 = \langle \pi_{OS_A} \rangle,\ p_2 = \langle \pi_{MAIL_{v1.5}} \rangle, \\ p_3 = \langle \pi_{OS_A}, \pi_{MAIL_{v1.5}} \rangle,\ p_4 = \langle \pi_{OS_B} \rangle, \\ p_5 = \langle \pi_{WEB_{v2.2}} \rangle,\ p_6 = \langle \pi_{FTP_{v3.5}} \rangle, \\ p_7 = \langle \pi_{OS_B}, \pi_{WEB_{v2.2}} \rangle,\ p_8 = \langle \pi_{WEB_{v2.2}}, \pi_{FTP_{v3.5}} \rangle, \\ p_9 = \langle \pi_{OS_B}, \pi_{WEB_{v2.2}}, \pi_{FTP_{v3.5}} \rangle \end{array} \right\}$$

The notation $V_a \xrightarrow{p_k} V_b$ is used to refer to a path p_k originating from V_a and ending in V_b. For instance in Figure III.2.6, the path which goes from V to V_5 is $V \xrightarrow{p_9} V_5 = \langle \pi_{OS_B}, \pi_{WEB_{v2.2}}, \pi_{FTP_{v3.5}} \rangle$.

Definition 6 (Cost Function) *Given a path set P_G, a cost function f_c is a function $f_c : P_G \to \mathbb{R}$ that associate a cost to each path in P_G. The following properties must hold.*

$$f_c(\langle \pi_i \rangle) \geq 0 \tag{III.2.7}$$

$$f_c\left(\langle \pi_{i_j}, \pi_{i_{j+1}} \rangle\right) \geq min\left[f_c\left(\pi_{i_j}\right), f_c\left(\pi_{i_{j+1}}\right)\right] \tag{III.2.8}$$

$$f_c\left(\langle \pi_{i_j}, \pi_{i_{j+1}} \rangle\right) \leq f_c\left(\pi_{i_j}\right) + f_c\left(\pi_{i_{j+1}}\right) \tag{III.2.9}$$

Example 5 *To better explain the meaning of equations III.2.8 and III.2.9, consider the case of adding a server to a view (p_1) and then defining its operating system fingerprint (p_2). The cost associated to this operation is less or equal to applying (p_1) and than (p_2) because it is possible to directly add a server with a certain OS fingerprint. For the same reason it is right consider the cost greater than the minimum of the two.*

If Equation III.2.9 holds strictly, then f_c is said to be an *additive cost function.*

III.2.2.2 Problems Statement

Here the two main problems that will be addressed are formalized.
Problem 1: Given a view manipulation graph G, the internal view V_i and a distance threshold $d \in \mathbb{R}$, find an external view V_d and a path $V_i \xrightarrow{p_d} V_d$ which minimizes $f_c\left(p_d\right)$ subject to $\delta(V_i, V_d) \geq d$.

[4] π_i denotes a generic $\pi \in \Pi$ and the number in the subscript denotes the position in the path.

Problem 2: Given a view manipulation graph G, the internal view V_i and a budget $b \in \mathbb{R}$, find an external view V_b and a path $V_i \xrightarrow{p_b} V_b$ which maximizes $\delta(V_i, V_b)$ subject to $f_c(p_b) \leq b$.

III.2.3 An Algorithmic solution to Identify Views

In this section, the algorithms used to solve the problems defined in Section III.2.2.2 are presented. For both problems it has been decided to adopt a top-k heuristic approach. The algorithms start from the internal view and proceed with the state space exploration by iteratively traversing the most promising outgoing edges until a termination condition is reached. To limit the exponential explosion of the search space only the k-most promising edges are traversed. To quantify the benefit in traversing a given edge, one can define a *benefit score* as the ratio between the distance between the corresponding views and the cost for achieving that distance. For each node in the graph, only the k outgoing edges with the highest values of the benefit score are traversed.

TopK-Distance (Problem 1) To solve this problem, the first step is to generate the view manipulation graph and then process it with the top-k algorithm. For efficiency purposes, it is possible to only generate a sub-graph G_d of the complete view manipulation graph G, such that generation along a given path stops as soon as the distance from the internal view becomes equal to or larger than the minimum required distance d (problem constraint). One can limit the graph generation up to this point because any additional edge in the graph will increase the cost of the solution and so it will not be included in the optimal path.

Algorithm 1 (GenerateGraph) describes how to generate the sub-graph up to the distance d. The algorithm adopts a queue (Q) to store the vertices to be processed. At each iteration of the while loop (Line 3), a vertex v is popped from the queue and the maximum distance from the internal view (O) is updated (Line 4). The constant MAX_INDEGREE (Line 5) is used to test a node that has been fully processed. When the in-degree is equal to MAX_INDEGREE the node has been linked to all the other nodes which have only one element in the configuration that differs. In this case, there are no new vertices to generate from this node. To compute MAX_INDEGREE one just need the set C of all admissible configurations per host:

$$MAX\_INDEGREE = \sum_{h \in Hosts} \#configurationPerHost_h - 1$$

On Line 6 a set of v's predecessors is retrieved from the graph.
The CREATE_COMBINATIONS (Line 7) function generates all possible combinations of configurations that can be created from v and have only one configuration element which differs form v's configuration. Both v and $predecessors$ are excluded from the returned

array. The function GET_ONE_CHANGE_VERTICES (Line 11) returns an array of all the vertices whose configuration differs just for one element from the vertex given as input. Those vertices need to be linked with the vertex v (Lines 12-14). Lines 15-17 check if the maximum distance of the graph has been reached. If this is not the case, the new generated vertex is pushed into the queue to be examined.

Algorithm 1 GenerateGraph(d)

Input: O = Internal Attack Surface; C = Set of Admissible Configurations per host; d = minimum required distance;
Output: G_d = sub-graph up to distance d

```
1:  //Initialization: Q is the queue of vertices to process (initially empty); G_d initially empty.
2:  maxDistance = 0; G_d.addVertex(O); O → Q;
3:  while Q ≠ Empty do
4:      v ← Q; maxDistance = MAX(maxDistance, DISTANCE(O,v));
5:      if v.indegree < MAX_INDEGREE then
6:          predecessors = G_d.GET_PREDECESSORS(v)
7:          newVertices = CREATE_COMBINATIONS(v,predecessors,C)
8:          for all vv ∈ newVertices do
9:              G_d.addVertex(vv)
10:             G_d.addDirectedEdge(v, vv, COST(v,vv), DISTANCE(v,vv));
                //Add an edge from v to vv
11:             verticesToLink = GET_ONE_CHANGE_VERTICES(G_d, vv)
12:             for all vtl ∈ verticesToLink do
13:                 G_d.addBidirectionalEdge(v, vv, COST(vv,vtl), DISTANCE(vv,vtl)); //Add a bidirectional edge between vv and vtl
14:             end for
15:             if maxDistance ≤ d then
16:                 vv → Q
17:             end if
18:         end for
19:     end if
20: end while
```

Once the sub-graph has been generated the top-k analysis can start so as to solve the problem. Algorithm 2 (TopK-Distance) is recursive and implements the top-k analysis on the generated sub-graph. The symbol v is used to denote the vertex under evaluation in each recursive call.

Line 1 creates an empty list to store all the path discovered from v. Line 2 is one of the two termination conditions. It checks if the current distance is greater than equal to d or no other nodes can be reached. The second term of the condition in the if statement takes into account both the case of a node with no outgoing edges and the case of a node whose successors are also its predecessors. Edges directed to predecessors are not considered in

order to construct a loop-free path.
In case the termination condition is satisfied, a solution has been found and a path can be constructed starting from v (the leaf) up to O by closing the recursion stack. Line 3 creates an empty path and adds v as the leaf of this path. Subsequently, the path list is updated and returned.
On Line 6 all the outgoing edges originating from v are sorted by decreasing benefit (distance/cost). Then on Lines 7-13 the top-k analysis takes place. For each of the best k destinations, *TopK-Distance* is recursively invoked. The result of this evaluation is a list of paths. The vertex v is then set as the origin of each of these paths (Lines 10-12). Line 14 contains the last termination condition where the updated path list is returned.

Algorithm 2 TopK-Distance

Input: G = graph; v = vertex to evaluate; k; d = minimum distance to reach;
cc = current cost; cd = current distance;
Output: List of Paths (Solutions)

```
1:  pathList = {}
2:  if cd ≥ d OR v.successors-(v.successors ∩ v.predecessors) = ∅ then
3:     p = emptyPath; p.addVertex(v); pathList ← p;
4:     return pathList
5:  end if
6:  SORT(v.outgoingEdgesList); numToEval = MIN(k, v.outgoingEdgesCount);
7:  for n ≤ numToEval do
8:     vv = v.outgoingEdgesList[n].getDestination()
9:     eval = TopK-Distance(G, vv, k, d, UPDATE(cc), UPDATE(cd))
10:    for all Path p ∈ eval do
11:       p.addVertex(v);
          p.addDirectedEdge(v, vv, DISTANCE(v, vv), COST(v, vv));
          //Add an edge from v to vv
12:    end for
13: end for
14: return pathList
```

TopK-Budget (Problem 2)

Algorithm 3 (TopK-Budget) implements both graph generation and exploration in order to improve time efficiency in the resolution of budget problems. It is very reasonable that the defender will invest a big budget, subsequently it will be very likely that a solution will be far deep in the graph.

The approach is to generate the graph only in the k-most profitable directions in order to limit as much as possible graph generation. This algorithm uses a queue to store the examined path that may be a solution. Line 5 retrieves the path p under examination

from the queue and, from this, its leaf v is then retrieved in order to further generate the graph (Lines 6-17). The generation process is similar to the one described in Algorithm 1 (GenerateGraph).

Then, the algorithm tries to estimate how good are the solutions it may find going further in the exploration through v. The estimation is based on v's benefit score and the maximum benefit score of v's successors. The estimation provides an insight knowledge about the solution it may discover going further in the exploration through v. On Line 18, all the successors of v (generated or linked at this stage) have been computed. The estimation is done by the function ESTIMATE which returns the maximum benefit value v's successors (vv). The estimation, the successor vv and the path under examination p are then added to a list (Lines 20-21).

On Line 24 the list is sorted by decreasing estimation value. Line 25 checks if there is no further exploration to perform. In this case the path under examination is added to the solutions list. Otherwise (Lines 28-31) new paths are generated from p. Each of the path is p plus a new leaf node that is in the top-k successors of v. All the new generated paths are then pushed into the queue for further examination. When the queue will be empty the complete list of solutions is returned.

Algorithm 3 TopK-Budget

Input: O = Internal Attack Surface; C = Set of Admissible Configurations per host; b = budget;
Output: List of Paths (Solutions)

1: //Initialization: Q is the queue of paths to process (initially empty); $solutions$ = {}; G is a graph and contains O
2: $path_O \rightarrow Q$ //$path_O$ is a trivial path containing just the internal attack surface
3: **while** $Q \neq Empty$ **do**
4: $toExploreDataHolder$ = {}
5: $p \leftarrow Q$; $v = p$.getLeaf();
6: **if** v.indegree < $MAX\_INDEGREE$ **then**
7: predecessors = $G.GET\_PREDECESSORS(v)$
8: newVertices = CREATE_COMBINATIONS(v,predecessors,C)
9: **for all** $vv \in$ newVertices **do**
10: G.addVertex(vv)
11: G.addDirectedEdge(v, vv, COST(v,vv), DISTANCE(v,vv)); //Add an edge from v to vv
12: verticesToLink = GET_ONE_CHANGE_VERTICES(G, vv)
13: **for all** $vtl \in$ verticesToLink **do**
14: G.addBidirectionalEdge(v, vv, COST(vv,vtl), DISTANCE(vv,vtl)); //Add a bidirectional edge between vv and vtl
15: **end for**
16: **end for**
17: **end if**
18: **for all** vv: v.get_direct_successors() **do**
19: **if** NOT v.isPrecedessor(vv) AND NOT p.contains(vv) AND (p.totalCost+COST(v,vv)$\leq b$) **then**
20: estimation = DISTANCE(v,vv)/COST(v,vv) + ESTIMATE(vv, p, C)
21: $toExploreDataHolder \leftarrow$ [vv, p, estimation]
22: **end if**
23: **end for**
24: SORT($toExploreDataHolder$); numToEval = MIN(k, $toExploreDataHolder$.size);
25: **if** numToEval = 0 **then**
26: $solutions \leftarrow p$
27: **end if**
28: **for** n leq numToEval **do**
29: newPath = $toExploreDataHolder$[n].p; newPath.addAsLeaf(vv)
30: newPath $\rightarrow Q$
31: **end for**
32: **end while**
33: **return** $solutions$

III.2.4 Experimental Evaluation

This section reports some experimental results to validate the proposed algorithmic approach to the problems of Section III.2.2.2. The performance of the algorithms have been evaluated in terms of processing time and approximation ratio for different numbers of hosts and numbers of different admissible configurations for each host. All the results reported in this section are averaged over multiple graphs with the same number of hosts and configurations for each node but different costs and distances among the vertices.

III.2.4.1 Evaluation of TopK-Distance

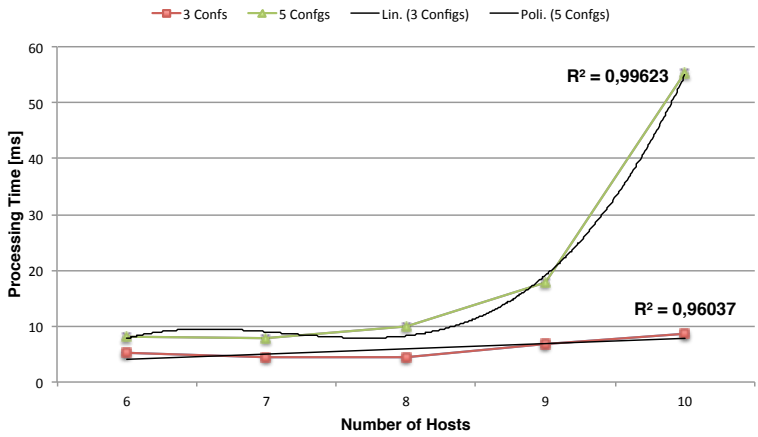


Figure III.2.7 TopK-Distance – Processing time vs. Number of hosts

As expected, the processing time increases when the number of configurations per nodes increases. Figure III.2.7 shows the processing time trends for the case of $k = 7$ and a required minimum distance $d = 5$. The processing time is practically linear in the number of hosts for the case of three configurations per host but, as soon as the number of configurations increases, it becomes polynomial as shown for the case of five configurations per host. Figure III.2.8 shows the processing time vs. graph size for different values of k. The graph size is intended as the number of nodes that have a distance from the internal view that is less than or equals to d.

Comparing the trends for $k = 3,4,5$ one can notice that the algorithm is polynomial for $k = 3$ and linear for $k = 4,5$. This can be explained by the fact that for $k = 3$ it is necessary to explore the graph more deeply than in the case of $k = 4,5$. Moreover, considering values of k bigger than five, the trend is again polynomial due to the fact that the algorithms starts to

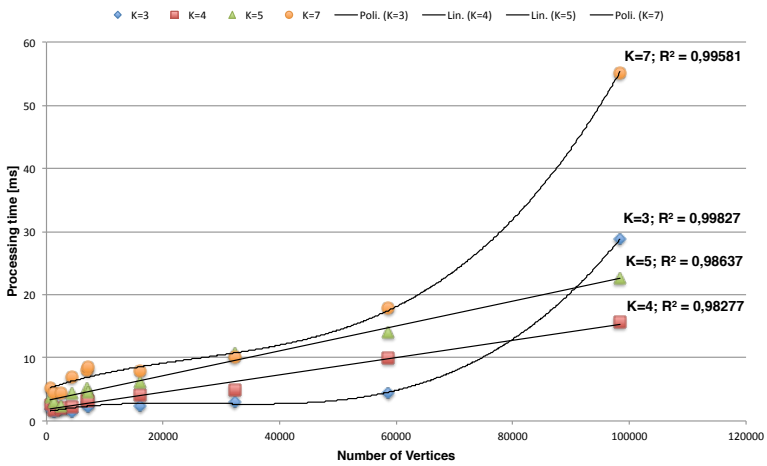


Figure III.2.8 TopK-Distance – Processing time vs. Graph size

explore the graph more broadly. Indeed, as it will be shown shortly, relatively small values of k provide a good trade-off between approximation ratio and processing time, therefore this result is extremely valuable.

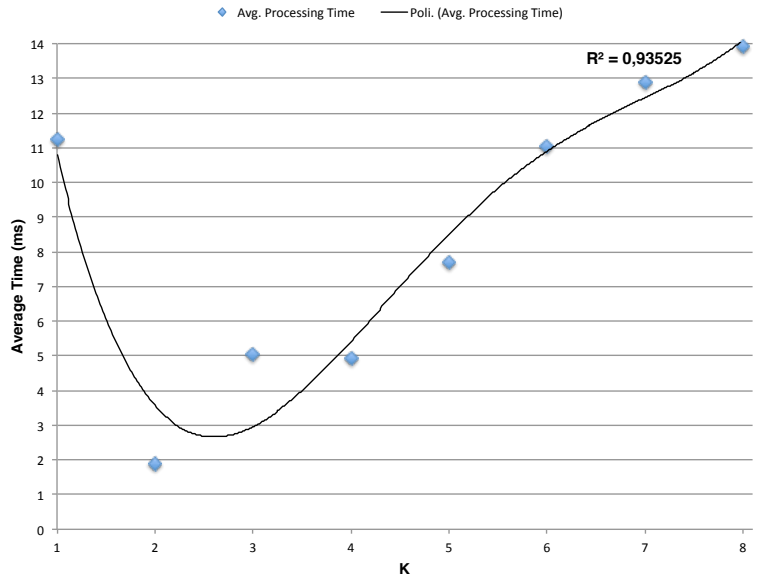


Figure III.2.9 TopK-Distance – Average processing time vs. k

To better visualize the relationship between processing time and k, Figure III.2.9 plots the average time to process different families of graphs against k. The trend can be approximated

by a polynomial function and the minimum is for $k = 2 \div 3$. For k greater than 4, the Average Time to process the graph will increase almost linearly. Moreover, the approximation ratio achieved by the algorithm has been evaluated. To compute the approximation ratio the cost of the algorithm's solution is divided by the optimal cost. In order to compute the optimal solution a function exhaustively measured the shortest path (in term of costs) from the internal attack surface to all the solutions with a distance greater than the minimum required d, and sorted those results by increasing cost. The optimal solution has the maximum distance and the minimum cost. When the algorithm could not find a solution (none of the discovered paths has a distance greater than the minimum required d), an infinite approximation has been assigned.

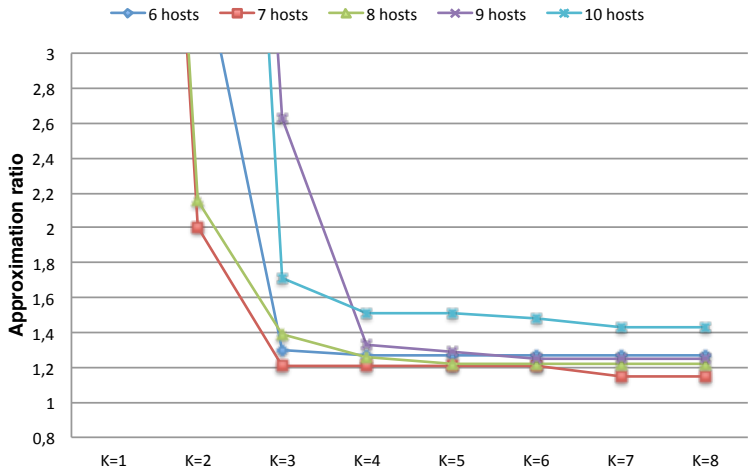


Figure III.2.10 TopK-Distance – Approximation Ratio vs. k

Figure III.2.10 shows how the ratio changes when k increases in the case of a fixed number of configurations per node (5 configurations per node) and for increasing number of hosts. It is clear that the approximation ratio improves when k increases. Relatively low values of k ($k = 3 \div 6$) are sufficient to achieve a reasonably good approximation ratio in a time-efficient manner. Finally, it has been measured the number of solutions found by the algorithm for increasing values of k.
Figure III.2.11 shows how the number of discovered solutions increases while k increase. The plot refers to the case of a fixed number of configurations per node (5 configurations per node) and for different number of hosts.

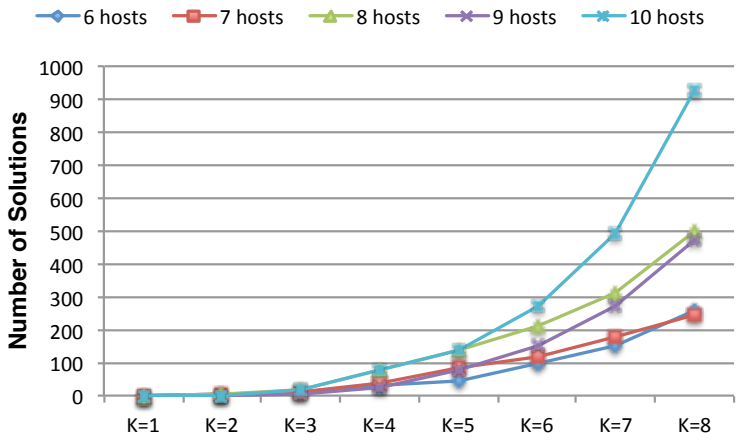


Figure III.2.11 TopK-Distance – Number of Solutions vs. k

III.2.4.2 Evaluation of TopK-Budget

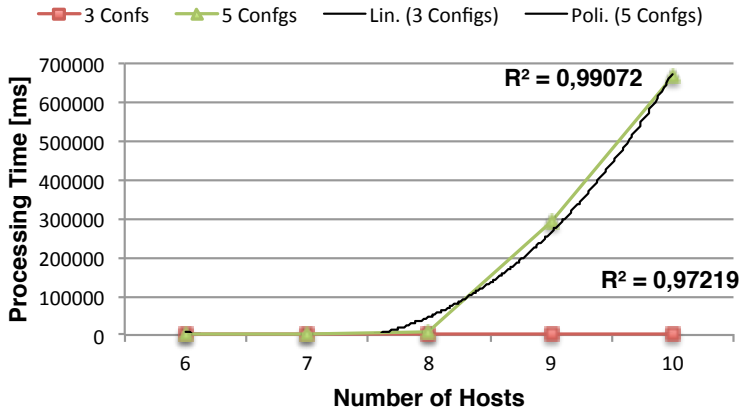


Figure III.2.12 TopK-Budget – Processing time vs. Number of hosts

As done for the previous case, measurements show that, as expected, the processing time increases when the number of configurations per nodes increases. Figure III.2.12 shows the processing time trends for the case of $k = 2$ and a budget $b = 18$. The processing time is practically linear in the number of hosts for the case of three configurations per host. In this case the minimum time (six hosts) is ~150ms and the maximum time (10 hosts) ~3500ms. When the number of configurations increases the time rapidly increases due to the time spent in the generation of the graph.

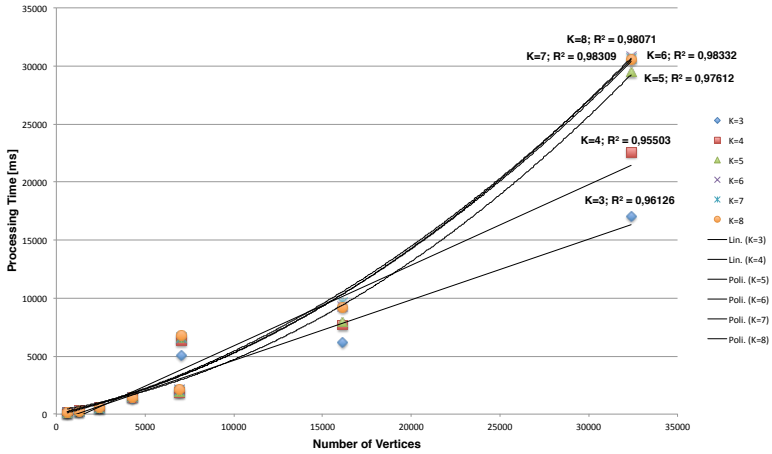


Figure III.2.13 TopK-Budget – Processing time vs. Graph size

Figure III.2.13, shows a scatter plot of average processing times against increasing graph size. This chart suggests that, in practice, processing time is linear in the size of the graph for small values of k.

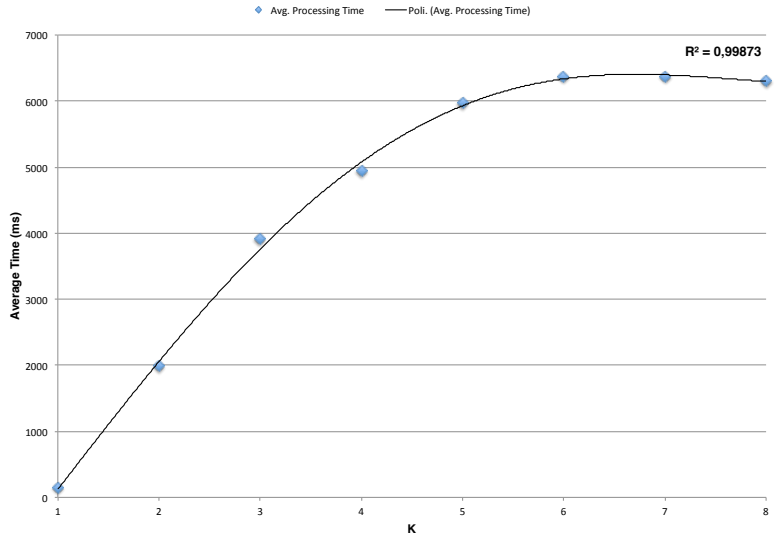


Figure III.2.14 TopK-Budget – Average processing time vs. k

Similarly, Figure III.2.14 shows how processing time increases when k increases and for a fixed value of budget ($b = 18$) and different graph families. The trend is approximated by a polynomial function and tends to saturate for values of $k \geq 6$. This can be explained

by the fact that with higher values of k most of the time is spend in the graph generation and starting from $k = 6$ it is generated almost completely. Even in this case the important result is that low values of k have a linear time, moreover, for these values, the algorithm can achieve a good approximation ratio.

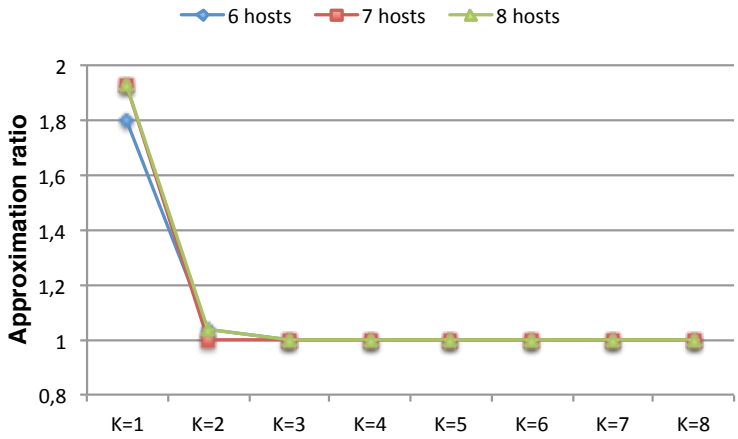


Figure III.2.15 TopK-Budget – Approximation Ratio vs. k

Figure III.2.15 shows how the ratio changes when k increases in the case of a fixed number of configurations per node (5 configurations per node) and for increasing number of hosts. The approximation ratio is good even for $k = 1$, but to obtain a more accurate solution it will be better to use $k = 2,3$. Greater values of k will be less good in terms of time-efficiency.

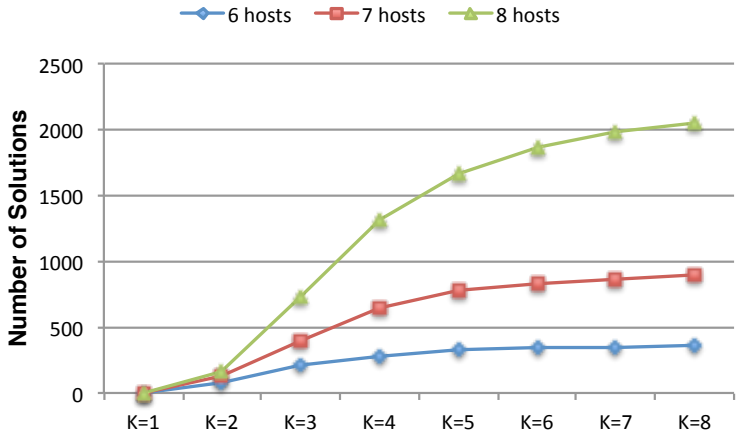


Figure III.2.16 TopK-Budget – Number of solutions vs. k

To compute the approximation ratio the optimal distance is divided by distance of the algorithm's solution. In order to compute the optimal solution a function exhaustively measured the shortest path (in term of distances) from the internal attack surface to all the solutions in a given graph. Due to the fact that it would be unfeasible to generate an exhaustive graph, the function generates a sub-graph up to a maximum number of nodes. Then the paths are ordered for decreasing values of distance and saved the cost needed to reach the solution. Then the algorithm is started with a budget equal to the saved cost. Finally, it has been measured the number of solutions found by the algorithm for increasing values of k (Figure III.2.16). The important result is that for values of k greater than four, the number of solutions increases linearly.

CHAPTER III.3

Steering Attackers away from resources using Fingerprint Deception

As it has already been discussed in the previous Chapter, attackers aim at collecting valuable information about the target system, including information about network topology, service dependencies, operating systems and unpatched vulnerabilities, in order to to engineer effective attacks against CPS's processing infrastructures and more in general against networked infrastructures. The previous Chapter discussed the problem of disrupting an attacker's reconnaissance efforts from a control perspective and proposed a graph-based approach to design a different view of the system so as to manipulate the attacker's perception of a system's attack surface. The idea behind this approach is that if one is able to alter the information used by attackers to fingerprint a system while maintaining the offered services unmodified for legitimate user, it will be possible to shift the attack surface (*virtual attack surface*) without actually modifying the system (*real attack surface*). A system's attack surface has been defined as the "subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack" [54]. Dynamically altering or shifting a system's attack surface has proven to be an effective strategy to thwart or significantly delay attacks. However, this type of approaches can potentially introduce significant overhead for the defender [43, 44].

This Chapter goes beyond simply introducing uncertainty for the attacker, and propose an approach to deceive potential intruders into making incorrect inferences about important system characteristics, including operating systems and active services. Differently

from many existing techniques, the proposed approach does so without changing the actual configuration of the system. In fact, it mainly consists in manipulating outgoing traffic such that, not only important details about operating systems and services are not revealed, but the traffic also resembles traffic generated by hosts and networks with different characteristics. Experiments conducted on a prototypal implementation show that the overhead introduced by the proposed approach is negligible, thus making this solution completely transparent to legitimate users. At the same time, by analyzing the results of attacker's tool while the approach is enforced it will be shown that it can effectively deceive the attackers, and steer them away from critical resources one wish to protect.

III.3.1 Reconnaissance and Fingerprinting

Most of the attacker's reconnaissance effort ,to identify systems hardware and software architectures, resort to fingerprinting tools. Those tools are able to identify the nature of a system by analyzing different aspects of his behavior, such as how it responds to certain requests, if it shows banner informations or fine-prints, if it exposes certain services and also how he responds under different loads. Authors in [75] categorize fingerprinting approaches in:

Classical Fingerprinting Even without resorting to stealth techniques of any kind, hosts will often announce their OS to anyone making a connection to them through welcome banners or header information. For example, when connecting to a host via the standard Telnet protocol the OS version is often sent to the client as part of a welcome message. FTP will also often provide this information either as a welcome banner or if prompted via a SYST command. HTTP can also be used by connecting to the host and initiating a simple GET / HTTP/1.0 query. If the Simple Network Management Protocol (SNMP) service is present on the machine (UDP/161), it is often left with the default public community name, which will allow remote detailed querying of the service and hence host. Other services that send back free useful information include IMAP, POP2, POP3, SMTP, SSH, NNTP and FINGER. A slightly more subtle approach is to use the anonymous ftp account (if supported), download a public binary required for ftp to function (eg /bin/ls) and examine it to determine what platform it was built for. A more primitive approach is to port scan the machine using any of the common port scanners freely available (eg Nessus, SAINT, nmap) and examine the returned list of listening ports for patterns common to a particular OS. Note that this approach is only effective against less secure hosts; particularly earlier Windows servers and those that do not implement accepted basic computer security concepts.

Active IP Packet Fingerprinting This is the predominant form of identifying operating system implementations, provoking the target into eliciting a response and analyzing it

carefully. A huge amount of information can be gleaned about the response to a carefully crafted network packet. The three common IP packet types (ICMP, TCP, UDP) are all used in this technique and various valid (and invalid) packets are sent to the host to refine the guess of the OS. The information used are from IP Header and TCP Header. Particularly, for TCP Options, interestingly it is not just the number of options a stack supports that can identify it, but also the order in which the options are returned, more details will be given in Section III.3.2.

Passive IP Packet Fingerprinting Passive fingerprinting is where a sniffer is set up on a network and passively collects/analyses packets in the flow of information between hosts. The techniques employed to identify the source OS are usually a subset of those utilized by Active IP packet fingerprinting, as these tools obviously monitor predominantly legitimate traffic, and hence cannot introduce anomalous packets. Nevertheless, substantial tools can be built using this technology.

Fingerprinting Through Service (Daemon) Querying Services that appear on a port-scan list can be further queried to produce identifying information. One notable tool is **Amap**, which proudly proclaims to send a trigger packet to an unknown port and again compares the response to a database of fingerprint packets. It makes little assumptions as to convention and, as such, claims to be able to detect daemons listening on non-standard ports. It is worth noting that this approach relies heavily on a direct match of the default application daemon to the underlying OS as you are inferring the nature of the OS rather than specifically testing it.

III.3.2 Technical Background

Operating System (OS) fingerprinting is the practice of determining the operating system of a remote host on a network. This may be accomplished either *passively* – by sniffing and analyzing network packets traveling between hosts – or *actively* – by sending carefully crafted packets to the target host and analyzing the responses [75].

Active fingerprinting approaches are typically more sophisticated than *passive fingerprinting*. In the simplest scenario, the attacker does not resort to stealth techniques and gathers information about the OS by trying to connect to the host. For instance, while establishing a connection via the standard Telnet or SSH protocol, the OS version is often sent to the client as part of a welcome message. Moreover, some FTP server implementations allow to retrieve this information through the `SYST` command.

In general, active fingerprinting techniques trigger the target into sending one or more responses, which are then analyzed by the attacker to infer the type and version of the OS installed on the remote host. Carefully crafted ICMP, TCP and UDP packets are sent to the

Operating System	IP Initial TTL	TCP Window Size
Linux Kernel 2.4/2.6	64	5,840
Windows XP	128	65,535
Windows 7	128	8,192

Table III.3.1 OS-dependent IP and TCP parameters

target in order to observe how it responds to both valid and invalid packets. For instance, in the case of TCP probes, features that can be used to distinguish between different operating systems include: (i) the order in which the TCP Options are formatted and (ii) the total length of TCP Options. Additionally, the IP header may reveal some information about the nature of the OS.

Conversely, *passive fingerprinting* consists in using a packet sniffer to passively collect and analyze packets traveling between hosts. A simple passive method consists in inspecting the Time To Live (TTL) field in the IP header and the TCP Window Size of the `SYN` or `SYN+ACK` packet in a TCP session. The values of both the initial TTL and the TCP Window Size depend on the specific OS implementation, as shown in Table III.3.1. One reason for this is that RFC specifications define intervals of values and recommended values, but do not mandate specific values. For instance, RFC 1700 recommends to initialize TTL to 64. Of course, relying only on the TTL value is not sufficient to determine the OS because, given the nature of this parameter, the TTL decreases as a packet traverses the network, and inferring the correct initial TTL may not always be possible.

Many different fingerprinting tools are available today. To better assess the impact of the approach presented in this thesis, a taxonomy to classify existing fingerprinting tools based on the different approaches they implement has been defined. Figure III.3.1 shows the proposed taxonomy. Given the variety of existing tools, it is practically impossible to develop a single technique that would defeat all of them. However, the proposed approach is effective against the most widely fingerprinting used tools. The tools that are explicitly targeted in this work are shown in boldface, whereas other tools that are at least partially impacted by the deception techniques are shown with a colored background.

III.3.3 Fingerprintg Tools and Mechanisms

Nmap Nmap OS fingerprinting works by sending up to 16 TCP, UDP, and ICMP probes to known open and closed ports on the target host [53]. These probes are specially designed to exploit the differences in the way different operating systems implement aspects of the protocols where standard specifications leave some degree of freedom to developers. After

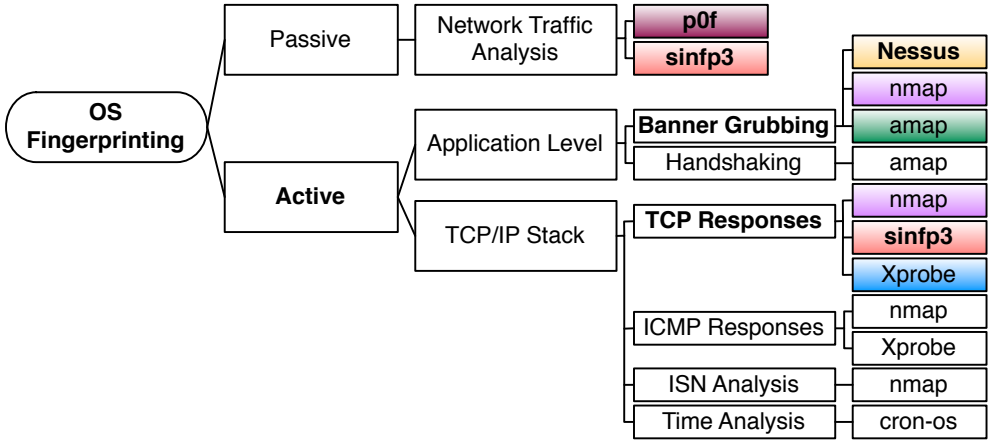


Figure III.3.1 Taxonomy of OS fingerprinting tools

sending probes, Nmap listens for responses. Dozens of attributes in those responses are analyzed and combined to generate a fingerprint.
A series of 6 TCP probes is sent to generate 4 test response lines used for gathering information about sequence number generation. Then an ICMP test follows. This test involves sending two ICMP echo request packets to the target. The results of these two probes are combined. Then 6 TCP tests follow. The last test is an UDP packet sent to a closed port. The character "C" (0x43) is repeated 300 times in the data field. If the port is truly closed and there is no firewall in place, Nmap expects to receive an ICMP *port unreachable* message.
Limitations: The probes are well known, so they can be identified and filtered. The Nmap approach is flawed by design [8]. If the target is behind multiple filtering devices, each with a different configuration policy, the attacker may end up with a signature that is built using responses generated by different systems. For instance, assume the first filtering device – instead of the true target – answers to TCP `SYN+ACK` packets with TCP `RST+ACK` and also spoofs the IP address of the true target: the attacker will then derive a signature that is built with some packets sent by the true target and other packets sent by different systems.

SinFP3 The development of SinFP was prompted by the need to reliably identify a remote host's operating system under worst-case network conditions [8]. Worst-case network conditions can be assumed to be characterized by the following facts: (i) only one TCP port is open; (ii) traffic to all other TCP and UDP ports is dropped by filtering devices; (iii) stateful inspection is enabled on the open port; and (iv) filtering devices with packet normalization are in place. In this scenario, only standard probe packets that use the TCP protocol can reach the target and elicit a response packet.
SinFP uses three probes: the first probe **P1** is a standard packet generated by the `connect()` system call, the second probe **P2** is the same as **P1** but with different TCP Options, and the

third probe **P3** has no TCP Options and has the TCP `SYN` and `ACK` flags set. The first two probes elicit two TCP `SYN+ACK` responses from the target. The third probe has the objective of triggering the target into sending a TCP `RST+ACK` response. After the three probes have been sent and the three replies have been received, a signature is built from the analysis of the response packets. Then, a signature matching algorithm searches in a database for a corresponding operating system fingerprint. The analysis of the responses considers both IPv4 headers[1] and TCP headers. With respect to IPv4 headers, the following fields are analyzed [8]:

- *TTL:* Some systems may use different initial TTL values for different types of packets. Specifically, the initial TTL for a TCP `SYN+ACK` may be different from the initial TTL for a TCP `RST+ACK`. Therefore, the difference between the TTL values of the responses to probes **P2** and **P3** can be used in the computation of the fingerprint.
- *ID:* The ID of the request is compared to the ID of the response. In the computation of the fingerprint, the following cases are considered: (i) the response's ID is 0; (ii) the response's ID is the same as the request's ID; (iii) the response's ID is obtained by adding 1 to the request's ID.
- *Don't Fragment bit:* Fingerprint computation considers whether the response has the `Don't Fragment` bit set or not.

With respect to TCP headers, the following fields are analyzed [8]:

- *Sequence number:* The TCP sequence number of the request is compared with the sequence number of the response. In the computation of the fingerprint, the following cases are considered: (i) the response's sequence number is 0; (ii) the response's sequence number is the same as the request's sequence number; (iii) the response's sequence number is obtained by adding 1 to the request's sequence number.
- *Acknowledgment number:* The same analysis as for the sequence number is applied.
- *TCP flags and TCP window size:* They are used as part of the signature.
- *TCP Options:* They are also used as part of the signature. Specifically, if either the MSS (Maximum Segment Size) value or the Window Scale value are specified they are used to create their own signature's elements.

An example of SinFP report against a Windows 7 target is reported in Figure III.3.2. The packets exchanged during the scan are shown in Figure III.3.3.

Limitations: When there are too few TCP Options in **P2**'s response, the signature's entropy becomes weak [8]. In fact, TCP Options are the most discriminant characteristics that compose a signature. That is because virtually no two systems implement exactly the same TCP Options, nor in the same order. Thus, when only the MSS option is in the TCP header, the risk of misidentification is high. SinFP also suffers from the same limitation of

[1] It also supports IPv6.

```
...
score 100: Windows: Microsoft: Windows: Vista (RC1)
score 100: Windows: Microsoft: Windows: Server 2008
score 100: Windows: Microsoft: Windows: 7 (Ultimate)
score 100: Windows: Microsoft: Windows: Vista
...
```

Figure III.3.2 SinFP report against a Windows 7 host

Figure III.3.3 Packets exchanged during a SinFP scan

all knowledge-based fingerprinting tools. Their capability to identify a system is limited by the existence of a corresponding fingerprint in the database.

Xprobe Xprobe2 is an active operating system fingerprinting tool with a different approach to OS fingerprinting. Xprobe2 relies on fuzzy signature matching and probabilistic guesses. ***Limitations:*** It appears that Xprobe2 is not maintained anymore, with the last version released in 2005. As a consequence, it can only be used to scan outdated legacy systems. Tested against a Ubuntu 12.04 machine (Kernel version 3.02), it returns `Running OS: "Linux Kernel 2.4.22" (Guess probability: 100%)`. Tested against a Windows 7 machine (with no firewall), it returns `Running OS: "Microsoft Windows 2003 Server Standard Edition" (Guess probability: 93%)`.

p0f p0f (v3) is a tool that utilizes an array of sophisticated, purely passive traffic fingerprinting mechanisms to identify the players behind any incidental TCP/IP communication [82]. Its fingerprint contains:

1. **ver:** IP protocol version.
2. **ittl:** Initial TTL used by the OS.
3. **olen:** Length of IPv4 options or IPv6 extension headers.
4. **mss:** Maximum Segment Size (MSS), if specified in TCP Options.

5. **wsize:** Window Size, expressed as a fixed value or a multiple of MSS, of MTU, or of some random integer.
6. **scale:** Window Scaling factor, if specified in TCP Options.
7. **olayout:** Comma-delimited layout and ordering of TCP Options, if any. Supported values: explicit end of options, no-op option, maximum segment size, window scaling, selective ACK permitted, timestamp.
8. **quirks:** Comma-delimited properties observed in IP or TCP headers.
9. **pclass:** Payload size.

Limitation: The initial TTL value is often difficult to determine since the TTL value of a sniffed packet will vary depending on where it is captured. The sending host will set the TTL value to the OS's default initial TTL value, but this value will then be decremented by one for every router the packet traverses on its way to the destination IP address. An observed IP packet with a TTL value of 57 can therefore be expected to be a packet with an initial TTL of 64 that has done 7 hops before it was captured. Additionally, this tool also suffers from the TCP Options entropy issue described for SinFP.

amap Amap [65] is an application-level service fingerprinting tool that probes services running on a remote server on a given port to identify the specific application that is listening on that port. Its purpose is to identify services that are not running on standard ports. Amap has a list of "triggers" which include binary as well as text handshake messages. Amap has been used as an indirect OS fingerprinting tool, meaning that it could infer the OS type from the services it is running [75]. It is worth noting that this approach relies heavily on a direct match between the default application daemon and the underlying OS as it is inferring the nature of the OS rather than specifically testing it.

Limitation: Amap is not very stealthy [75]: 11 parallel connections sending an unexpected message at the application protocol level are surely recorded in the application log file, provided that the application maintains a good logging level. Apart from logging at the application level, it is difficult to detect an Amap probe since it uses the OS system calls to the TCP/IP stack and therefore no signature can be found at the level of the TCP, UDP or IP packets. Nevertheless, it is still possible to write IDS rules that are able to detect probes.

Nessus Nessus provides a comprehensive analysis of a target, including information about its OS and vulnerabilities. Nessus does not implement its own OS fingerprint mechanism but relies on the output of several different tools. It is interesting to study how Nessus performs the OS detection because each of the methods being used can also be adopted independently by an attacker. For instance, an attacker can perform a scan through nmap or Nessus, identify the open ports and then use SinFP or other scripts to infer the operating system listening on a specific port. In the following † is used to denote a method that is likely to be adopted by an attacker who is targeting a system behind firewall and aims at

discovering the target OS fingerprint. Tenable Research introduced a highly accurate form of operating system identification [38]. This method combines the outputs of various other plugins that execute separate techniques to guess or identify a remote operating system. Specifically, this process takes inputs from the following other scripts, each reporting its own OS guessing:

- *os_fingerprint_ftp*† Uses the remote FTP banner to attempt to identify the underlying operating system.
- *os_fingerprint_html*† Uses the HTML content returned by certain HTTP requests to fingerprint the remote OS.
- *os_fingerprint_http*† Uses the remote web server signature to infer the version of the Windows or Linux distribution running on the remote host.
- *os_fingerprint_mdns*†If an mDNS server is present, will perform a highly accurate identification of Apple OS X systems.
- *os_fingerprint_msrprc*Identifies the remote version and service pack of Windows by making certain MSRPC requests against the remote Windows box.
- *os_fingerprint_ntp* Queries the Network Time Protocol daemon to perform a highly accurate OS guess.
- *os_fingerprint_sinfp*† Implements the SinFP TCP/IP fingerprinting algorithm. Only requires one open port to fingerprint an OS.
- *os_fingerprint_smb* Identifies the remote Windows OS based on a query to SMB.
- *os_fingerprint_snmp* If credentials are available to perform an SNMP query, data from the 'sysDesc' parameter is reported.
- *os_fingerprint_ssh*† Attempts to identify the remote OS from the SSH banner.
- *os_fingerprint_telnet*† Attempts to identify the remote OS from the Telnet banner.
- *os_fingerprint_uname* If SSH credentials of the remote UNIX hosts are provided, the results of 'uname -a' are obtained.
- *os_fingerprint_linux_distro* If SSH credentials of the remote Linux host are provided, the specific release is obtained.
- *os_fingerprint_xprobe*† Attempts to identify the OS type and version by sending more or less incorrect ICMP requests using the techniques outlined in [7].

Each of these plugins reports a confidence level for their scan results. An example of Nessus output for OS identification is reported in Figure III.3.4.

Limitation: Nessus's approach to fingerprinting can be very effective when used during a "credentialed" scan. Otherwise, it will report partial information and in some cases it will not use all the plugins it is equipped with. In the case one is targeting a host behind firewall and NAT translates port 22 to a server and port 80 to another, the Nessus result is likely to report only the OS information that has been gathered through the *os_fingerprint_ssh* plugin. Moreover, in case the *os_fingerprint_sinfp* plugin is used, only the first entry of SinFP's result,

```
The remote host is running Linux Kernel 2.4
Confidence Level : 70
Method : SinFP

The remote host is running Linux Kernel 2.6
Confidence Level : 60
Method : ICMP
```

Figure III.3.4 Nessus OS identification report

which has a confidence level greater than 70%, is reported. For instance, due to the fact that Windows 2008 Server, Windows Vista and Windows 7 share the same fingerprint, a Windows 2008 Server can be misidentified as a window Vista host (see Figure III.3.2).
Nessus's approach to identify vulnerabilities is strictly dependent on service banners and welcome messages. Generally, Nessus merely checks if the service's version present in the service's banner belongs to a certain interval. For instance, if a vulnerability is know to be present in a service up until version 2.0, it is really simple to make Nessus generate false negatives by exposing a fake service banner claiming that the service version is higher than 2.0.

III.3.4 An approach to defeat fingerprinting efforts

With respect to OS fingerprinting, this thesis proposal to deceive attackers relies on modifying outgoing traffic in a way that such traffic resembles traffic generated by a different protocol stack implementation. As pointed out in Section III.3.2, protocol specifications may leave some degrees of freedom to developers. The choices that a developer makes with respect to (i) default values (e.g., initial TTL, size of the TCP window), (ii) length of TCP Options, or (iii) order of the TCP Options may reveal the nature of the operating system or even the type of device (e.g., firewall, switch, router, printer or general purpose machine).
All the information required to impersonate a certain operating system or device can be extracted from the SinFP's signature database. All the outgoing packets that may reveal relevant information about the OS are modified to reflect the deceptive signature, as shown in Figure III.3.5. The most critical step in this process is the manipulation of the TCP Options. In order to present the attacker with a deceptive signature, not only it is needed to modify some parameters, but also to reorder the options and correctly place no-operation[2] to obtain the right options length. The TCP Options format can be inferred from the signature. Modifying the Options length requires to adjust the total length field in the IP header, the Offset value in the TCP header and subsequently adjust the sequence numbers. On the

[2]As specified in RFC 793, this option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary.

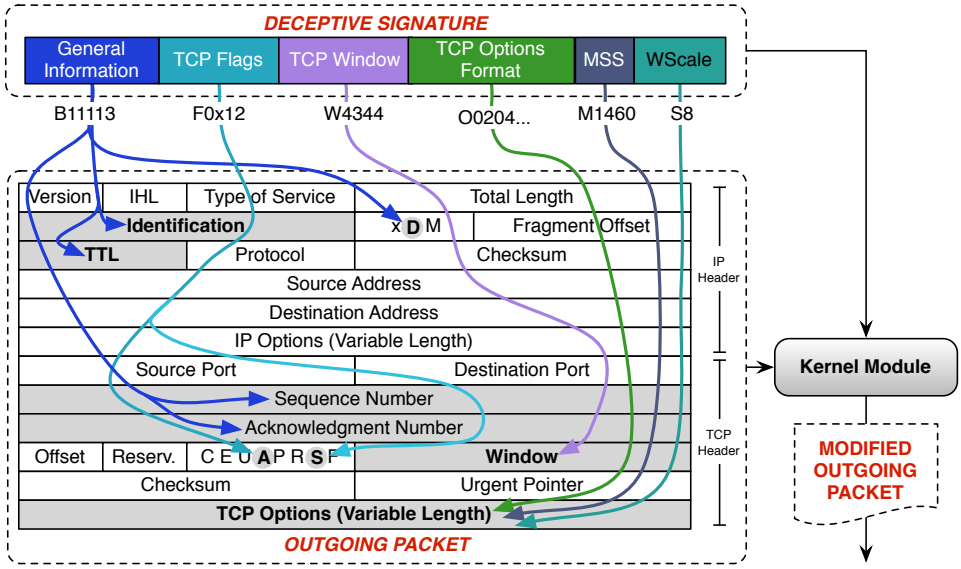


Figure III.3.5 Manipulation of outgoing traffic to reflect deceptive signatures

bright side, it has been noticed that the majority of commonly used operating systems share the same length for TCP Options. See Section III.3.6.2 for some examples of deceptions that can be created with this method even considering the case of different TCP Options lengths.

With respect to service fingerprinting, one should consider the following two cases: (i) the service banner can be modified through configuration files, and (ii) the service banner is hard-coded into the service executable. Being able to modify the packet carrying the identifying information before it leaves the host (or the network) enables to successfully address both scenarios. Moreover, even if services are under direct control, it is preferred to alter service banners in a completely transparent way. Our long term goal is to develop a network appliance that can be deployed at the network boundary and which is able to transparently manipulate services and operating system fingerprints.

It is worth noting that, when the original service banner is replaced with a deceptive banner of a different length, it is needed to: (i) adjust the size of the packet based on the difference in length between the two banners, (ii) modify the total length field in the IP header, (iii) modify the sequence numbers in order to be consistent with the actual amount of bytes sent, and (iv) correctly handle the case of fragmented packets, which requires to reassemble a packet before modifying it.

However, this approach is not applicable to all categories of services. Services that actively use the banner information during the connection process (such as SSH) require us to use a non-transparent approach. For instance, the SSH protocol actively uses the banner information while generating hashes in the connection phase. The banner format is:

"SSH-protocolversion-softwareversioncomments\r\n".

Even though this approach can deceive tools like nmap and amap, modifying the banner will cause legitimate user to receive termination messages from the server[3] such as: (i) Bad packet length or (ii) Hash Mismatch.

In summary, defeating passive tools requires to modify all outgoing packets, whereas defeating active tools only requires to alter those packets that are likely to be part of an attacker's probes.

III.3.5 Solution Implementation

The implemented solution is based on a kernel module that uses the Netfilter POST_ROUTING (Figure III.3.6) hook to process and modify relevant information in the IP header, TCP header and TCP payload. Netfilter is a packet handling engine introduced in Linux Kernel 2.4. It enables the implementation of handlers to redirect, reject or alter incoming and outgoing packets. Netfilter can be extended with hooks. A hook is a function handler that allows specific kernel modules to register callback functions within the kernel's network stack. A registered callback function is then called back for every packet that traverses the net filter stack. The decision is to use the POST_ROUTING hook because it allows to alter the packets just before they are finally sent out.

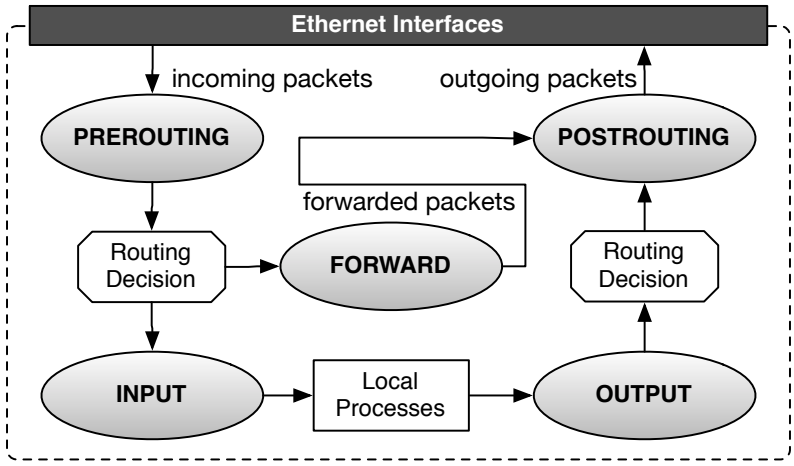


Figure III.3.6 Netfilter packet handling and hooks

[3] Errors occurs only during the connection phase, and altering the banner will not affect previously established connections.

Specifically, it has been implemented an **operating system fingerprint module** to modify the responses to the SinFP's probes and a **service fingerprint module** to modify banner information for specific services.

III.3.5.1 Operating system fingerprint module

The hook function checks if the packet is a response to the first SinFP's probe (**P1**): an ACK+SYN packet with a length of 44 bytes. If this is the case, the packet is altered in order to mimic a particular operating system (more details are provided later), otherwise the module checks whether the packet is an ACK+SYN with a length 60 bytes. This packet is used in most TCP connections and may be a response to the second SinFP's probe (**P2**). If so, the packet is modified accordingly, based on the target OS fingerprint.
Additionally, it has been verified that this approach can deceive the **p0f** tool if the kernel module modifies the TTL value of all IP packets and the window size value of all TCP packets.
During the packet manipulation stage, the module tracks whether any of the following has been altered: the IP Header, the TCP header, the length of TCP Options, the TCP payload or its size. Based on this information, it will:

- modify the *IP Total length* value, if the size of the TCP payload has changed;
- recompute the *TCP Offset* value in the TCP header and the *IP Total length*, if the length of the TCP Options has changed;
- recompute the TCP checksum, if the TCP header and/or the TCP payload have been altered;
- recompute the IP checksum, if the IP header has been altered.

In order to modify the responses such that they appear to have been generated by a specific OS, a script has been created. It (i) extracts the required characteristics of the responses to the first and second probe from SinFP's signature database, and (ii) generates the C code necessary to alter the responses. The script determines how the following policies should be set up:

- **ID policy:** the ID could be a fixed value different from zero, zero or a random number.
- **Don't fragment bit policy:** the DF bit can be enabled or disabled.
- **Sequence Number policy:** the sequence number can be zero or not altered.
- **Ack Number policy:** the ack number can be zero or not altered.
- **TCP Flags policy:** the TCP flags value is copied from the signature.
- **TCP Window Size policy:** the Window size is copied from the signature.
- **TCP MSS Option policy:** the MSS value is copied from the signature.
- **TCP WScale Option policy:** the WScale is copied from the signature.
- **TCP Options policy:** the TCP Options layout is copied from the signature.

The generated code is then compiled in order to build the actual kernel module. The scheme of the resulting kernel module is presented in Listing III.3.1[4].It is assumed that all the *set* and *get* functions are able to access the packet and track if the IP or TCP header have been modified.

```
if(ip->protocol == TCP && ip->len == 44 && tcp->ack == 1 && tcp->syn == 1)
{
//Probably 1st sinfp3's probe Response
set_id();
set_df_bit();
set_ttl();

set_tcp_window();
set_tcp_flags();
set_tcp_sequence();
set_tcp_ack();

if(new_option_len != option_len)
{
modify_packet_size(); //expands or shrinks
//packet and updates IP Lenght and Offset
}

set_tcp_options(MSS, WScale, Option_Layout);
}
else if(ip->protocol == TCP && ip->len == 60 && tcp->ack == 1 && tcp->syn == 1)
{
//Probably 2nd sinfp3's probe Response

//Extract the timestamp value from the packet and save it for re-injecting
//it in the right position later
timestamp = get_tcp_timestamp();

set_id();
set_df_bit();
set_ittl();
set_tcp_window();
set_tcp_flags();
set_tcp_sequence();
set_tcp_ack();

if(new_option_len != option_len)
{
modify_packet_size(); //expands or shrinks
//packet and updates IP Lenght and Offset
}
set_tcp_options(timestamp, MSS, WScale, Option_Layout);
}

if(tcpHeader_modified)
{
tcp->check = 0;
tcp->check = tcp_csum();
}
```

[4]The code dealing with sequence numbers adjustment is omitted.

```
50
51 if(ipHeader_modified)
52 {
53 ip->check = 0;
54 ip->check = ip_csum();
55 }
```

Listing III.3.1 OS deception kernel module

III.3.5.2 Service Fingerprint Module

In order to alter the service fingerprint, one should modify the banner sent by the application either at the time of establishing a connection or in the header of each application-level protocol data unit. Packets matching the service source port one wants to protect are analyzed. If a packet contains data, the banner string is searched and subsequently replaced. When replacing the banner, the packet size can vary: the packet is then resized according to the specific case. Listing III.3.2 shows the sample pseudo-code for the case of an Apache Server[5].

```
1  #define FAKE_APACHE_BANNER "Apache/1.1.23"
2  ...
3  if (ntohs(tcph->source) == 80 && len > 0)
4  {
5  // Pointers where to store the start and end address of the Apache Banner String
       for substitution
6  char *b = NULL, *l = NULL;
7
8  // Pointer to the TCP payload
9  char *p= (char *)((char *)tcph+(uint)(tcph->doff*4));
10
11 b = strstr(p, "\r\nServer:"); //String Search
12 if (b != NULL) l = strstr( ((char *)b + 10), "\r\n");
13
14 if (b != NULL && l != NULL)
15 {
16 //b points to \r\nServer: x, so we add 10 to move to the beginning of x
17 uint8_t signature_len = l - (b + 10);
18
19 if (signature_len != (sizeof(FAKE_APACHE_BANNER)-1))
20 {
21 resize_packet();
22 }
23 copy(b + 10, FAKE_APACHE_BANNER,sizeof(FAKE_APACHE_BANNER)-1);
24 }
25 ...
26 }
```

Listing III.3.2 Service deception kernel module

[5] For the sake of conciseness, the code for checksum recomputation is omitted.

III.3.6 Evaluation

III.3.6.1 Legitimate User Perspective

In the first set of experiments, the approach is evaluated from the point of view of legitimate users interacting with the system being defended. The goal is to make manipulation of outgoing traffic completely transparent to users from both a functional and a performance perspective. To this end, performance tests with the Apache Benchmark are run in order to asses the server's ability to process 20,000 requests, with a maximum of 200 simultaneous active users. The results are shown in Figure III.3.7.

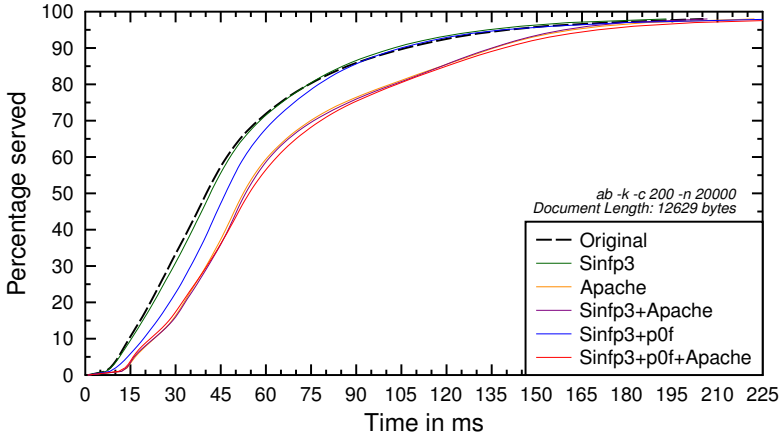


Figure III.3.7 Apache Benchmark

The tests here performed involved different system configuration scenarios: (i) the behavior of the system is not altered (Original); (ii) the kernel module to alter the OS fingerprint and deceive only *active fingerprinting tools* is enabled (Sinfp3); (iii) the kernel module to alter the service fingerprint is enabled (Apache); (iv) both modules from scenarios (ii) and (iii) are enabled (Sinfp3+Apache); (v) the kernel module to alter the OS fingerprint and deceive both *active* and *passive fingerprinting tools* is enabled (Sinfp3+p0f); and (vi) both modules from scenarios (iii) and (v) are enabled.

The performance degradation for scenario (ii) is negligible, as only two packets need to be altered for each connection. On the other hand, when the OS fingerprint kernel module alters all the outgoing packets (scenario (v) above), there is a slight delay in the response time due to the greater number of packets that need to be altered. When the Service Fingerprint Kernel Module is enabled (scenario (iii) above), the response time increases due to the string comparison operations performed to identify and replace the banner information. It is

clear from Figure III.3.7 that the Service Fingerprint Kernel Module has the largest impact on the performance of the system. However, even in the worst scenario, the performance degradation is limited.

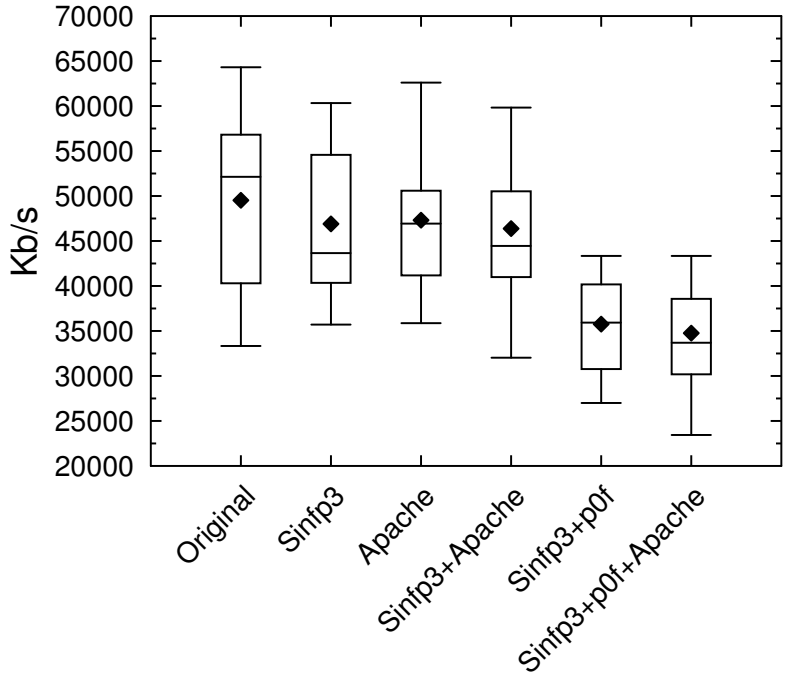


Figure III.3.8 FTP Transfer Rate (500 MB file)

Using the same scenarios, also the overhead introduced by the kernel modules on large data transfers has been tested. The evaluation consists in uploading a 500MB file to an FTP server. Most packets will not be altered, but the `if` statements in the kernel modules need to be evaluated, thus adding some overhead, which will eventually affect the net transfer rate. Moreover, as it will discussed in Section III.3.6.3 some alteration may affect the protocol performance. As one can see from Figure III.3.8, the more conditions need to be evaluated the greater the effect on performance is going to be.

III.3.6.2 Attacker Perspective

In the second set of experiments, the approach is evaluated from the point of view of an attacker trying to determine the operating system of a remote host or the type of services running on it.

	Without Deception	**With Deception**
Device Type	General Purpose 85%	General Purpose 65%
OS	Ubuntu 12.04 85%[6]	Windows Vista 65%[7]
Info	13	15 60% True Positives; 13.33%: False Positives; 26.67%: Induced Info
Low	0	0
Medium	0	2 (100%: False Positives)
High	0	2 (100%: False Positives)
Critical	0	0

Table III.3.2 Results of Nessus scans

Nessus In order to test how the approach can deceive an attacker using Nessus, the system has been audited with and without the deceptive Kernel Module enabled. Table III.3.2 shows the results of the respective Nessus scans. The original system is a fully patched Ubuntu 12.04 server and has no known vulnerabilities. When no deception is used, the system is correctly identified, and all the information derived by Nessus is accurate.

Next, both OS and service fingerprinting are deceived by exposing a Windows 7/Vista OS fingerprint and an Apache 2.2.1 service fingerprint. When the deception mechanism is enabled, the OS is misidentified accordingly. Moreover, deceiving service fingerprinting leads to the false positives in the identification of vulnerabilities.

Fingerprinting Tools Table III.3.3 reports the results of scans performed with different fingerprinting tools. As one can see, the approach is able to effectively deceive several fingerprinting tools. For instance, it is able to alter the perception of the target system even when the attacker uses either nmap or Xprobe++, which adopt a different probing scheme.

Unconventional Fingerprints By intelligently crafting responses to SinFP probes it is possible to force attackers into misclassifying a remote host as any of a broad variety of networked assets. For instance, a conventional Linux-based Server can be fingerprinted as a network switch, an ADSL gateway or even a printer. Of course these unconventional fingerprints will force the attackers to derive an inconsistent map of the target network: it is really unlikely that a company web server is hosted on a Laserjet Printer. Even if one

[6] Method HTTP
[7] Method SinFP
[8] ID=0; Windows=1460;
[9] ID=0; Windows=1460; WScale = 2

Deception \ Tool	Sinfp3	p0f	nmap	Xprobe++
None	Linux 3.0.x-3.2.x (94%) Linux 2.4.x-2.6.x (73%)	Linux 3.x	Linux 2.6.x-3.x	Linux 2.4.19-28 (94%)
Windows Server 2008	Server 2008/Vista/7 (100%) FreeBSD 7.0-9.0 (73%)	Windows 7/8	Unknown	Linux 2.4.26 (78%)
Firewall Fortigate	Firewall Fortigate 100%)	Unknown	Unknown	Linux 2.4.23 (94%)
NetBSD 5.0.2	NetBSD 5.0.2 (98%)	Unknown	Unknown	Linux 2.4.21 (92%)
Windows Server 2008 (Partial[8])	Server 2008/Vista/7 (98%) FreeBSD 7.0-9.0 (73%)	Windows 7/8	Unknown	Linux 2.4.26 (81%)
Windows Server 2008 (Partial[9])	Server 2008/Vista/7 (88%)	Unknown	Unknown	Linux 2.4.14 (81%)

Table III.3.3 OS Fingerprinting Tools Deception

forces a consistent network map (think to the case of a network switch listening on the ssh port) he/she may encounter some difficulties in being convincing. The problem is that most of the fingerprinting tools does not have fingerprints for the same non-general-purpose devices. In this case with unconventional fingerprints will be known for some tools and unknown for others.

As an example, here is show how one can successfully create SinFP deceptions for different network monitoring appliances, firewalls and printers. A partial SinFP output for the case of an *HP Officejet 7200 Printer* is reported in Figure III.3.9, whereas Figure III.3.10 illustrates the steps involved in forcing the attacker to believe that the target device is a printer.

```
...
score: 100: Printer: HP Officejet 7200
score: 100: Printer: HP Officejet Pro L7600
score: 73: Appliance: APC AP9319
...
```

Figure III.3.9 HP Officejet 7200 Printer

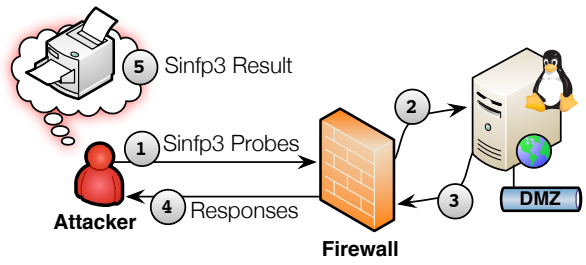


Figure III.3.10 SinFP Printer Deception

III.3.6.3 Drawbacks

Altering some parameters of the TCP header can affect the connection performances, when legitimate users actively use the protocol based on the modified parameters. In such cases, the proposed mechanism is not completely transparent, and drawbacks include those listed in the following.

Maximum Segment Size (MSS) This parameter defines the largest unit of data that can be received by the destination of the TCP segment. Modifying this value makes the host to announce a different limit for its capabilities. Consider two hosts h_A and h_B, where h_B is the host being altered. h_A sends a SYN packet with an MSS of 1,460 and h_B responds with a SYN/ACK that has an MSS of 1,480. h_A will not send any packets with a segment larger than 1,480 bytes to h_B. Note that h_A is not required to send segments of exactly 1,480 but it is required not to exceed this limit. For the same reason, it is not possible to advertise a larger value than what hosts are able to handle.

Window and Windows Scale Factor These two parameters affect the TCP flow control, which regulates the amount of data a source can send before receiving an acknowledgment from the destination. A sliding window is used to make transmissions more efficient and control the flow so that the destination is not overwhelmed with data. The TCP Window scale factor is used to scale the window size by a power of 2. The window size may vary during the data transfer while the scale factor is determined once at the time of establishing a connection. Modifying the window size can alter the throughput: if the window is smaller than the available bandwidth multiplied by the latency then the sender will send a full window of data and then sit and wait for the receiver to acknowledge the data. This results in lower performance.

Selective ACK Selective acknowledgment allows the sender to have a better idea of which segments are actually lost and which have arrived out of order. If one disables the SACK permitted option for a host that supports it, he/she may limit the performances depending on the capabilities of the counterpart.

CHAPTER III.4

Summary

In this part of the thesis, the adoption of security adaptation techniques has been discussed for the case of CPS's processing infrastructure and more in general for the case of networked systems. With the aim of disrupting attacker's effort to engineer effective attacks, a deception mechanism has been presented. In Chapter III.2, the problem of disrupting an attacker's reconnaissance efforts has been looked from a control perspective and it has been proposed a graph-based approach to manipulate the attacker's view of a system's attack surface. In order to achieve this objective, the notion of system view and distance between views have been formalized. Then it has been defined a principled approach to manipulate responses to attacker's probes so as to induce an external view of the system that satisfies certain properties. In particular, it has been proposed an efficient algorithmic solutions to different classes of problems, namely (i) inducing an external view that is at a minimum distance from the internal view while minimizing the cost for the defender; (ii) inducing an external view that maximizes the distance from the internal view, given an upper bound on the admissible cost for the defender. Experiments conducted on a prototypal implementation of the proposed algorithms have confirmed that the approach is efficient and effective in steering the attackers away from critical resources. As part of future research plans, we will further investigate the notion of distance between views, and will define analytical approaches to quantify the uncertainty introduced for the attacker as well as the ability of this approach to deceive attackers. It is possible that future works will try to address the problem through game theory approaches.

Basing on the proposed formalization, Chapter III.3 describes a solution to implement the deception mechanism and designs a Linux kernel module able to alter system responses to attacker's probe. As all cyber attacks are typically preceded by a reconnaissance phase in which attackers aim at collecting critical information about the target system, disrupting this phase of an attack may be extremely effective in thwarting or at least delaying the attack. Specifically, the aim of the presented solution is defeating operating system fingerprinting and service fingerprinting, which are respectively used to determine the operating system of a remote host and the type of services running on it. With respect to OS fingerprinting, the approach consists in modifying outgoing traffic so that it resembles traffic generated by a host with a different operating system. As for service fingerprinting, the approach consists in modifying the service banner by intercepting packets before they leave the host or network. Experiments, aimed at verifying the effectiveness of the approach, tested a prototypal implementation against real reconnaissance tools used by attacker's and have shown that the approach is effective and introduces only a negligible overhead for defenders and legitimate users. The principal difference between the proposed approach and state of art solutions is the transparency. The legitimate user and the servers are not aware that this security mechanism is in place and it does not require modification to the network architecture or the adoption of software defined networks that is typical of moving target defenses and adaptation techniques.

Conclusion

This thesis described the advances to the state of art in the design of a secure Cyber-Physical System with respect to three main contribution: (i) the proposal of a multiformalism modeling framework for the design and validation of large scale monitoring infrastructures, (ii) the proposal and prototyping of new security mechanisms to protect both the software and hardware of embedded nodes used to monitor physical processes, and (iii) an improvement to the processing infrastructure security through the enforcement of defenses against reconnaissance attacks.

As regards the definition of modelling approaches to design monitoring infrastructures, a solution to the challenge of modelling and simulating large scale monitoring infrastructures has been proposed through the adoption of compositional and multiformal models. This approach ability to scale and to provide the required amount of details has been shown through examples validated with real world data and its advantage for designers has been discussed in terms of how they are allowed to conduct what-if analysis and early measurements in the design phase. The discussion specifically focused on the challenging case of monitoring infrastructures deployed through wireless sensor embedded nodes. In this domain, the contribution of the proposed compositional and multiformal approach is twofold. A comprehensive and detailed model of sensing infrastructure embedded node has been presented: even if the architecture of the model is quite generic and can be adapted to several kinds of nodes, it has been proven to be consistent to real world architectures by comparing simulation results with real world measurements for both the cases of Strago S.p.A. architecture and commercial TelosB. Moreover, it has been show that Markovian Agent Model can be very useful in the detection of quality/reliability bottleneck of the network and on the definition of proper and cost effective maintenance policies.

As regards advances for in software and hardware security monitoring infrastructures, the thesis prosed, prototyped and evaluated the adoption of novel security mechanisms. Focusing on software level mechanisms, adaptation techniques have been exploited to gain advantage of proactive security mechanisms. The idea behind these techniques is to alter the system during its lifetime in order to introduce uncertainty for attackers and limit vulnerability exposure. Adaptation Techniques have been proven effective in other context of information technology but their adoption in a constrained resource domain, such as embedded sensor nodes, have not been exhaustively investigated. This thesis proposed and implemented a reconfiguration framework, specifically designed for embedded nodes, and its evaluation proved its applicability and efficacy in the CPS's monitoring infrastructure context.
Focusing on hardware level security mechanisms, advances to embedded nodes physical protection have been presented with regard to the implementation of a FPGA-based sensing node architecture. The node architecture has been detailed and it has been shown how it can provide secure primitives to asses the hardware and software integrity as well as offering cryptographic operation allowing the implementation of trusted infrastructures. The main component in the node architecture is the Trusted-Platform Module which can enforce a chain of trust through the hardware and software of a node. Disrupting physical attacks makes the injection of malicious traffic, as well as the information leakage, more difficult or even unfeasible. Moreover, being able to trust a node computation and detect if it has been tampered, improves the overall sensing network security.

Even though the monitoring infrastructure seems the weakest point due to not so mature technological solutions, advances are still to do in the protection of the processing infrastructure. In the last Part, the thesis has addresses this challenge by analyzing how adaptive techniques can be enforced to dynamically change aspects of a networked-system's configuration in order to introduce uncertainty for the attacker. Has it has been shown also for the case of monitoring infrastructures, Adaptation Techniques can offer defenses to different kind of attacks. As regards CPS's processing infrastructure security, the thesis focused on attacks aimed at discovering the processing infrastructure architecture and how AT can disrupt reconnaissance attack targeting CPS' back-ends. It has been proposed a graph-based algorithmic solution to manipulate attacker's view of processing systems. This formalization has been then used to design and implement a deception based defense to defeating networked systems reconnaissance. Specifically, the aim of the presented solution is defeating operating system fingerprinting and service fingerprinting, which are respectively used to determine the operating system of a remote host and the type of services running on it. With respect to OS fingerprinting, the approach consists in modifying outgoing traffic so that it resembles traffic generated by a host with a different operating system. As for service fingerprinting, the approach consists in modifying the service banner by intercepting pack-

ets before they leave the host or network. Experiments have been discussed and verified the effectiveness of the approach. A prototypal implementation of the defense mechanisms has been enforced against real reconnaissance tools used by attacker's and it has been shown effective and able to introduce only a negligible overhead for defenders and legitimate users.

Future research effort will be focused on advancing the proposed hardware and software prototypes to a higher level of engineering as well as evaluating the adoption of other proactive security solution to defend cyber-physical systems.

References

[1] ABBASI, F. H., HARRIS, R. J., MORETTI, G., HAIDER, A., AND ANWAR, N. Classification of malicious network streams using honeynets. In *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)* (2012), IEEE, pp. 891–897.

[2] AKYILDIZ, I. F., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. Wireless sensor networks: a survey. *Computer Networks 38* (2002), 393–422.

[3] AKYILDIZ, I. F., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. Wireless sensor network simulators - a survey and comparisons. *International Journal of Computer Networks 2*, 6 (2011), 249–265.

[4] ALBANESE, M., BATTISTA, E., JAJODIA, S., AND CASOLA, V. Manipulating attacker's view of a system's attack surface. In *Communications and Network Security (CNS), 2014 International Conference on* (2014), IEEE.

[5] ALBANESE, M., JAJODIA, S., PUGLIESE, A., AND SUBRAHMANIAN, V. S. Scalable analysis of attack scenarios. In *Proceedings of the 16th European Symposium on Research in Computer Security (ESORICS 2011)* (Leuven, Belgium, September 2011), Springer, pp. 416–433.

[6] ANAND, M., CRONIN, E., SHERR, M., BLAZE, M., IVES, Z., AND LEE, I. Security challenges in next generation cyber physical systems. *Beyond SCADA: Networked Embedded Control for Cyber Physical Systems* (2006).

[7] ARKIN, O. ICMP usage in scanning - the complete know-how. http://althing.cs.dartmouth.edu/local/ICMP_Scanning_v3.0.pdf, June 2001.

[8] AUFFRET, P. SinFP, unification of active and passive operating system fingerprinting. *Journal in Computer Virology 6*, 3 (August 2010), 197–205.

[9] BARBARESCHI, M., BATTISTA, E., AND CASOLA, V. On the adoption of fpga for protecting cyber physical infrastructures. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on* (2013), IEEE, pp. 430–435.

[10] BATTISTA, E., BARBARESCHI, M., MAZZEO, A., AND VENKATESAN, S. Advancing wsn physical security adopting tpm-based architectures. In *IICPS 2014, colocated with the 15th IEEE International Conference on Information Reuse and Integration (IEEE IRI 2014)* (2014), IEEE.

[11] BATTISTA, E., CASOLA, V., MARRONE, S., MAZZOCCA, N., NARDONE, R., AND VITTORINI, V. An integrated lifetime and network quality model of large wsns. In *Measurements and Networking Proceedings (M&N), 2013 IEEE International Workshop on* (2013), IEEE, pp. 132–137.

[12] BATTISTA, E., CASOLA, V., MAZZEO, A., AND MAZZOCCA, N. Siren: a feasible moving target defence framework for securing resource–constrained embedded nodes. *International Journal of Critical Computer-Based Systems 4*, 4 (2013), 374–392.

[13] BATTISTA, E., CASOLA, V., MAZZOCCA, N., NARDONE, R., AND MARRONE, S. A compositional modelling approach for large sensor networks design. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on* (2013), IEEE, pp. 422–429.

[14] BECCUTI, M., CODETTA-RAITERI, D., AND FRANCESCHINIS, G. Multiple abstraction levels in performance analysis of WSN monitoring systems. In *Proc. of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools* (Belgium, 2009), ICST, pp. 73:1–73:10.

[15] BEN-OTHMAN, J., DIAGNE, S., MOKDAD, L., AND YAHYA, B. Performance evaluation of a hybrid MAC protocol for wireless sensor networks. In *Proc. of the 13th ACM Int. Conf. on Modeling, analysis, and simulation of wireless and mobile systems* (2010), ACM, pp. 327–334.

[16] CARDENAS, A. A., AMIN, S., AND SASTRY, S. Secure control: Towards survivable cyber-physical systems. *System 1*, a2 (2008), a3.

[17] CASOLA, V., DE BENEDICTIS, A., AND ALBANESE, M. A moving target defense approach for protecting resource constrained distributed devices. In *IEEE Proceedings of IRI 2013* (2013).

[18] CASOLA, V., DE BENEDICTIS, A., DRAGO, A., AND MAZZOCCA, N. Analysis and comparison of security protocols in wireless sensor networks. In *Proc. of the IEEE Symposium on Reliable Distributed Systems Workshops, SRDSW 2011* (2011), IEEE Computer Society, pp. 52–56.

[19] CASOLA, V., DE BENEDICTIS, A., MAZZEO, A., AND MAZZOCCA, N. Sensim-sec: security in heterogeneous sensor networks. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on* (2011), pp. 1–8.

[20] CASOLA, V., GAGLIONE, A., AND MAZZEO, A. A reference architecture for sensor networks integration and management. In *Proc. of GSN 2009* (2009), vol. 5659 LNCS, Lecture Notes in Computer Science, Springer, pp. 158–168.

[21] CERTICOM RESEARCH. *Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography*, September 2000. Version 1.0.

[22] CHEMINOD, M., DURANTE, L., AND VALENZANO, A. Review of security issues in industrial networks. *IEEE Transactions on Industrial Informatics 9*, 1 (2013), 277–293. cited By (since 1996) 0.

[23] CHEN, C.-M., CHENG, S.-T., AND ZENG, R.-Y. A proactive approach to intrusion detection and malware collection. *Security and Communication Networks 6*, 7 (2013), 844–853.

[24] CONTIKI DEVELOPERS. Cooja simulator. http://www.contiki-os.org/start.html, 2011.

[25] CROSSBOW. Micaz datasheet. http://www.xbow.com/Products/productdetails.aspx?sid=164.

[26] D'ARIENZO, M., IACONO, M., MARRONE, S., AND NARDONE, R. Estimation of the energy consumption of mobile sensors in WSN environmental monitoring applications. In *Proceedings of the 2013 27th International Conference on Advanced Information Networking and Applications Workshops* (Washington, DC, USA, 2013), WAINA '13, IEEE Computer Society, pp. 1588–1593.

[27] DAVID BARROSO, B. A practical approach for defeating nmap os-fingerprinting. http://nmap.org/misc/defeat-nmap-osdetect.html, Jan 2013.

[28] DEAVOURS, D. D., CLARK, G., COURTNEY, T., DALY, D., DERISAVI, S., DOYLE, J. M., SANDERS, W. H., AND WEBSTER, P. G. The Möbius framework and its implementation. *IEEE Trans. Softw. Eng. 28* (October 2002), 956–969.

[29] DERLER, P., LEE, E. A., AND VINCENTELLI, A. S. Modeling cyber–physical systems. *Proceedings of the IEEE 100*, 1 (2012), 13–28.

[30] DI MARTINO, C., CINQUE, M., AND COTRONEO, D. Automated generation of performance and dependability models for the assessment of wireless sensor networks. *Computers, IEEE Transactions on 61*, 6 (2012), 870–884.

[31] DUAN, Q., AL-SHAER, E., AND JAFARIAN, H. Efficient random route mutation considering flow and network constraints. In *Proceedings of the IEEE Conference on Communications and Network Security (CNS)* (cc, 2013), IEEE, pp. 260–268.

[32] DUNLOP, M., GROAT, S., MARCHANY, R., AND TRONT, J. Implementing an ipv6 moving target defense on a live network. In *Proceedings of the National Moving Target Research Symposium* (Annapolis, MD, USA, June 2012).

[33] DUTTA, P., HUI, J., CHU, D., AND CULLER, D. Securing the deluge network programming system. In *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on* (0-0 2006), pp. 326 –333.

[34] EVANS, D., NGUYEN-TUONG, A., AND KNIGHT, J. C. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats.* Springer, 2011, ch. Effectiveness of Moving Target Defenses, pp. 29–48.

[35] EXECUTIVE OFFICE OF THE PRESIDENT, NATIONAL SCIENCE AND TECHNOLOGY COUNCIL. Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program. http://www.whitehouse.gov/, December 2011.

[36] GRAWROCK, D. Tcg specification architecture overview revision 2.0. http://www.trustedcomputinggroup.org/resources/tpm_library_specification, March 2014.

[37] GRIBAUDO, M., CEROTTI, D., AND BOBBIO, A. Analysis of on-off policies in sensor networks using interacting markovian agents. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on* (2008), pp. 300–305.

[38] GULA, R. Enhanced operating system identification with nessus. http://www.tenable.com/blog/enhanced-operating-system-identification-with-nessus, Feb 2009.

[39] HU, W., TAN, H., CORKE, P., SHIH, W. C., AND JHA, S. Toward trusted wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN) 7*, 1 (2010), 5.

[40] HUI, J. *TinyOS Network Programming*. University of Virginia Engineering, 2004.

[41] HUI, J. W., AND CULLER, D. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (New York, NY, USA, 2004), SenSys '04, ACM, pp. 81–94.

[42] JAFARIAN, J. H., AL-SHAER, E., AND DUAN, Q. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the 1st workshop on Hot topics in Software Defined Networks* (New York, NY, USA, 2012), ACM, pp. 127–132.

[43] JAJODIA, S., GHOSH, A. K., SUBRAHMANIAN, V. S., SWARUP, V., WANG, C., AND WANG, X. S., Eds. *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, 1st ed., vol. 100 of *Advances in Information Security*. Springer, 2013.

[44] JAJODIA, S., GHOSH, A. K., SWARUP, V., WANG, C., AND WANG, X. S., Eds. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, 1st ed., vol. 54 of *Advances in Information Security*. Springer, 2011.

[45] JAJODIA, S., GHOSH, A. K., SWARUP, V., WANG, C., AND WANG, X. S., Eds. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, 1st ed., vol. 54 of *Advances in Information Security*. Springer, 2011.

[46] JEONG, J., AND CULLER, D. Incremental network programming for wireless sensors. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on* (oct. 2004), pp. 25 – 33.

[47] KARLOF, C., SASTRY, N., AND WAGNER, D. Tinysec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (New York, NY, USA, 2004), SenSys '04, ACM, pp. 162–175.

[48] KORKALAINEN, M., SALLINEN, M., KÄRKKÄINEN, N., AND TUKEVA, P. Survey of wireless sensor networks simulation tools for demanding applications. In *Proc. of the 5th Int. Conf. on Networking and Services* (2009), pp. 102–106.

[49] KRAUSS, C., STUMPF, F., AND ECKERT, C. Detecting node compromise in hybrid wireless sensor networks using attestation techniques. In *Security and Privacy in Ad-hoc and Sensor Networks*. Springer, 2007, pp. 203–217.

[50] LARA, J., AND VANGHELUWE, H. Atom^3: A tool for multi-formalism and meta-modelling. In *Fundamental Approaches to Software Engineering*, R.-D. Kutsche and H. Weber, Eds., vol. 2306 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002, pp. 174–188.

[51] LEVIS, P., AND CULLER, D. Mate': a tiny virtual machine for sensor networks. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems* (New York, NY, USA, 2002), ASPLOS-X, ACM, pp. 85–95.

[52] LUK, M., MEZZOUR, G., PERRIG, A., AND GLIGOR, V. MiniSec: A secure sensor network communication architecture. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks* (2007), IPSN '07, pp. 479–488.

[53] LYON, G. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.Com, January 2009.

[54] MANADHATA, P. K., AND WING, J. M. An attack surface metric. *IEEE Transactions on Software Engineering 37*, 3 (May 2011), 371–386.

[55] MARRONE, S., MAZZOCCA, N., NARDONE, R., PRESTA, R., ROMANO, S. P., AND VITTORINI, V. A san-based modeling approach to performance evaluation of an ims-compliant conferencing framework. In *Transactions on Petri Nets and Other Models of Concurrency VI*. Springer, 2012, pp. 308–333.

[56] MARY, W., WANG, H., TAN, C. C., LI, Q., WANG, H., SHENG, B., TAN, C., AND LI, Q. Wm-ecc: an elliptic curve cryptography suite on sensor motes. Tech. rep., In, 2007.

[57] MOTEIV CORPORATION. Telos (rev b) datasheet. http://www.moteiv.com, 2004.

[58] NATIONAL SCIENCE FOUNDATION (NSF), NATIONAL SCIENCE AND TECHNOLOGY COUNCIL. Cyber-physical system (cps) program solicitation nsf 15-541. http://www.nsf.gov/pubs/2015/nsf15541/nsf15541.pdf, 2014.

[59] NEUMAN, C. Challenges in security for cyber-physical systems. In *DHS: S&T workshop on future directions in cyber-physical systems security* (2009), Citeseer.

[60] NICTA. Castalia simulator. https://castalia.forge.nicta.com.au/index.php/en/, 2011.

[61] PADMAVATHI, D. G., SHANMUGAPRIYA, M., ET AL. A survey of attacks, security mechanisms and challenges in wireless sensor networks. *arXiv:0909.0576* (2009).

[62] PANTA, R. K., BAGCHI, S., AND MIDKIFF, S. P. Zephyr: efficient incremental reprogramming of sensor nodes using function call indirections and difference computation. In *Proceedings of the 2009 conference on USENIX Annual technical conference* (Berkeley, CA, USA, 2009), USENIX'09, USENIX Association, pp. 32–32.

[63] PERRIG, A., SZEWCZYK, R., TYGAR, J. D., WEN, V., AND CULLER, D. E. Spins: security protocols for sensor networks. *Wirel. Netw. 8*, 5 (Sept. 2002), 521–534.

[64] QUAGLIETTA, E., D'ACIERNO, L., PUNZO, V., NARDONE, R., AND MAZZOCCA, N. A simulation framework for supporting design and real-time decisional phases in railway systems. *Proc. of IEEE Conf. on Intelligent Transportation Systems, ITSC* (2011), 846–851.

[65] RANA, A. What is amap and how does it fingerprint applications? http://www.sans.org/security-resources/idfaq/amap.php, Mar 2014.

[66] RIVEST, R., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21* (1978), 120–126.

[67] ROMER, K., AND MATTERN, F. The design space of wireless sensor networks. *Wireless Commun. 11*, 6 (Dec. 2004), 54–61.

[68] SANDERS, L. Secure boot of zynq-7000 all programmable soc (xapp1175). http://www.xilinx.com/support/documentation/application_notes/xapp1175_zynq_secure_boot.pdf, Sep 2013.

[69] SANDERS, W., AND MEYER, J. Stochastic activity networks: Formal definitions and concepts. In *Lectures on Formal Methods and PerformanceAnalysis*, vol. 2090 LNCS. Springer Berlin Heidelberg, 2001, pp. 315–343.

[70] SHAREEF, A., AND ZHU, Y. Energy modeling of processors in wireless sensor networks based on petri nets. In *Proc. of the 2008 Int. Conf. on Parallel Processing - Workshops* (2008), IEEE Computer Society, pp. 129–134.

[71] SHU, G., AND LEE, D. Network protocol system fingerprinting - a formal approach. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)* (Apr 2006), IEEE, pp. 1–12.

[72] STANFORD UNIVERSITY. Tossim simulator. http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM, 2013.

[73] TANG, L.-A., YU, X., KIM, S., GU, Q., HAN, J., LEUNG, A., AND PORTA, T. L. Trustworthiness analysis of sensor data in cyber-physical systems. *Journal of Computer and System Sciences 79*, 3 (2013), 383 – 401.

[74] TRIVEDI, K. SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks* (Washington, DC, USA, 2002), DSN '02, IEEE Computer Society.

[75] TROWBRIDGE, C. An overview of remote operating system fingerprinting. SANS Institute InfoSec Reading Room, July 2003.

[76] VANDERBILT UNIVERSITY. Jprowler simulator. http://w3.isis.vanderbilt.edu/projects/nest/jprowler/, 2013.

[77] VITTORINI, V., IACONO, M., MAZZOCCA, N., AND FRANCESCHINIS, G. The OsMoSYS approach to multi-formalism modeling of systems. *Software and Systems Modeling 3* (2004), 68–81.

[78] WATSON, D., SMART, M., MALAN, G. R., AND JAHANIAN, F. Protocol scrubbing: network security through transparent flow modification. *IEEE/ACM Transactions on Networking (TON) 12*, 2 (2004), 261–273.

[79] XILINX. Security solutions for zynq-7000 all programmable soc. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/security.html, Jun 2014.

[80] XIONG, X., WONG, D. S., AND DENG, X. Tinypairing: A fast and lightweight pairing-based cryptographic library for wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference* (Apr. 2010), WCNC '10, pp. 1–6.

[81] YANG, Y., DENG, R. H., ZHOU, J., AND QIU, Y. Achieving better privacy protection in wireless sensor networks using trusted computing. In *Information Security Practice and Experience*. Springer, 2009, pp. 384–395.

[82] ZALEWSKI, M. p0f v3 (version 3.06b). http://lcamtuf.coredump.cx/p0f3/, Jen 2012.

[83] ZHU, S., SETIA, S., AND JAJODIA, S. Leap+: Efficient security mechanisms for large-scale distributed sensor networks. *ACM Trans. Sen. Netw. 2*, 4 (Nov. 2006), 500–528.

List of Acronyms

Abbreviations

ACD	Adaptive Cyber Defense
ADC	Analog to Digital Converter
AT	Adaptation Techniques
CPS	Cyber–physical systems
DevC	Device Configuration
ECC	Elliptic Curve Cryptography
FPGA	Field-Programmable Gate Array
FSBL	First Stage Boot Loader
MA	Markovian Agent
MAM	Markovian Agents Models
MCU	Microcontroller
MSS	Maximum Segment Size
MTD	Moving Target Defense
OCM	On-Chip Memory
OS	Operating System
PCR	Platform Configuration Register
PDR	Packet Delivery Ratio
PL	Programmable Logic (FPGA)
PS	Processing System (FPGA)
SAN	Stochastic Activity Network
SIREN	SecurIty Reconfiguration for Embedded Nodes
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TTL	Time To Live
WSN	Wireless Sensor Network