Duarte Miguel Garcia Raposo

MONITORING INDUSTRIAL WIRELESS SENSOR NETWORKS:
A MODEL TO ENHANCE SECURITY AND RELIABILITY

VOLUME 1

Tese no âmbito do Programa de Doutoramento em Ciências e Tecnologias da Informação, orientada pelo Professor Doutor Jorge Sá Silva, pelo Professor Doutor André Rodrigues, e pelo Professor Doutor Fernando Boavida, e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

julho de 2019

Faculdade de Ciências e Tecnologia
da Universidade de Coimbra

Monitoring Industrial Wireless Sensor Networks: A model to enhance Security and Reliability

Duarte Miguel Garcia Raposo

VOLUME 1

Tese no âmbito do Programa de Doutoramento em Ciências e Tecnologias da Informação, orientada pelo Professor Doutor Jorge Sá Silva, pelo Professor Doutor André Rodrigues, e pelo Professor Doutor Fernando Boavida, e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

julho de 2019

1 2 9 0

UNIVERSIDADE Ð
COIMBRA

This work was partially supported by the projects I-LOCATOR and e-STEAM, Portuguese QREN IDT projects Nr. 2482/2012 and 38650/2013 (COMPETE, European Union); and by the SOCIALITE (PTDC/EEI-SCR/2072/2014) and the MOBIWISE (P2020 SAICTPAC/0011/2015) projects, both co-financed by Portuguese COMPETE 2020.

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR

QREN QUADRO DE REFERÊNCIA ESTRATÉGICO NACIONAL

Cofinanciado por:

COMPETE 2020

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Europeu
de Desenvolvimento Regional

”When I was a newspaper man, I remember I hated having to write an article while there were still questions that I wanted to ask...”

Robert A. Caro

Acknowledgements

Saying thank you to everyone that help me in these last years is special, and can only be done in my native language. For me, it's important to choose the right words in this moment, to acknowledge the time and dedication that my kind and gentle advisers, colleagues, friends and family spent during this journey.

A todos/as, o meu obrigado. Não foi muito fácil chegar até aqui. Como em tudo existiram "algumas" dificuldades, mas penso que o balanço final é bastante positivo. Refiro-me, particularmente, às pessoas que fizeram parte desta fase da minha vida – as de sempre e as que foram chegando. De algum modo, adicionam notoriedade à tese aqui apresentada.

Em primeiro lugar, quero agradecer ao Professor Jorge, ao Professor Boavida, e ao André, de igual modo – pelo incentivo, rigor e disponibilidade demonstrada durante este longo, longo percurso. De modo particular, agradecer ao Professor Jorge a iniciativa de me propor fazer este Doutoramento, e de não ter desistido nos períodos mais difíceis. Ao Professor Boavida um agradecimento muito especial pelo rigor nas múltiplas revisões de textos (nem sempre entregues com a antecedência necessária). Ao André, agradeço a partilha de toda esta jornada. Agradeço todas as nossas conversas, que me ajudaram a superar muitos desafios, a persistência, o interesse e a amizade.

Em segundo lugar, agradeço a todos/as os/as meus/minhas colegas dos laboratórios do G6.1, G6.2 e no geral à família do LCT. Foram bons parceiros ao longo destes anos. Agradeço ao David Nunes e ao Dien que estiveram presentes numa primeira etapa. Ao Marcelo, à Soraya, ao Ngombo, ao Oswaldo e à Inês por toda a ajuda ao longo destes últimos 3 anos no projeto Socialite. Marcelo, Soraya e Ngombo o quadro do laboratório não vai ter a mesma graça/piada sem os nossos diagramas. Jorge Proença, Tiago Cruz, David e Karima obrigado pela vossa presença.

Em terceiro lugar, mostrar a minha gratidão aos meus ex-colegas da eneida, ao Flávio, ao José e ao Luís. Ao Flávio, por ser um amigo que esteve sempre disponível, pelas longas conversas sobre os mais variados tópicos, e por todo o conhecimento que sempre partilha. Ao José, por ter tentado, desde o início, encontrar um caminho. Ao Luís pela ajuda na construção da testbed.

E sim, a ti Andreia. Que neste momento estás a corrigir este texto, a descomplicá-lo como sempre fazes. Como sempre consigo-te roubar um sorriso. Obrigado pela tua dedicação, entrega e ajuda em momentos difíceis. Por estares em todos os momentos especiais, desde a primeira apresentação, até à última. Pelo conforto quando a situação mostrava o contrário. Pela tua alegria. Adorei partilhar esta etapa contigo.

Com um especial e enorme carinho a toda a nossa família, aos meus pais, à minha avó, e aos que já não estão presentes e que gostariam de hoje estar aqui. Obrigado pelo que têm feito ao longo da vossa vida.

Gostaria de agradecer também à Cristina, por mostrar mais do mundo, e por desde muito cedo me ter incentivado a fazer este percurso. Por fim, gostaria também de agradecer à professora, pelos conselhos que me deu e não segui, e à sua contagiante boa disposição.

Abstract

A new generation of industrial systems are growing, in a new industrial evolution that connects wireless technologies with powerful devices, capable to make their own decisions. In the Industry 4.0 paradigm, industrial systems are becoming more powerful and complex in order to keep with the requirements needed to build Cyber Physical Systems (CPSs). To achieve such paradigm, Industrial Wireless Sensor Networks (IWSNs) are a key-technology capable to achieve micro-intelligence, with low-cost, and mobility, reducing even further today's already short production cycles, and at the same time allowing new industrial applications. Specifically, in the last decade, more reliable and deterministic standards were proposed, all of them sharing the same base technology, the IEEE802.15.4 standard. At the same time, until now, Industrial Control Systems (ICSs) have remained disconnected from the Internet, relying in the airgap principle to ensure security. Nevertheless, there is a lack of post-deployment tools to monitor technologies like the WirelessHART, ISA100.11a, WIA-PA and the ZigBee standards, contrary to what happens with most common wired technologies. The lack of these tools can be explained by several characteristics present in current Internet of Things (IoT) devices like the fragmentation of the operating systems, the need to develop specific firmware for each application, different hardware architectures; etc.

Thus, in this thesis, and looking for the current challenges of Industrial IoT (IIoT) technologies, a monitoring model is proposed, capable not only to monitor current industrial networks based on the IEEE802.15.4 standard, but also the in-node components of sensor nodes, in several hardware and firmware architectures. The proposed architecture explores several techniques to obtain free monitoring metrics; agents in charge of processing these metrics; and relies in management standards to share all the monitoring information. To prove the performance of this proposal, a WirelessHART testbed was built, as well as the different components presented in the architectural model. Additionally, using representative anomalies, injected in a WirelessHART testbed, an Anomaly Detection system capable to detect network anomalies and security attacks was built, proving the effectiveness of the presented model in the network perspective. In the same way, in order to prove the effectiveness in the detection of firmware and hardware anomalies, an Anomaly Detection system for in-node components was also built. The two Anomaly Detection systems were able to detect with high recall and low false positive ratio the anomalies inserted in the systems, proving that the proposed model can be used as a post-deployment tool in real industrial scenarios.

Keywords: Monitoring; Industrial IoT; Wireless Sensor Networks; Anomaly Detection; WirelessHART; ISA100.11a; WIA-PA; ZigBee; Industry 4.0; Fault Taxonomy; Attack tools; Anomaly Injection.

Resumo

Atualmente assiste-se a uma nova geração de sistemas industriais, numa evolução que junta tecnologias sem fios com dispositivos embebidos, cada vez mais inteligentes e capazes. No âmbito da Indústria 4.0, os sistemas industriais tornaram-se mais potentes e complexos, em resposta aos requisitos impostos pelos novos Sistemas Ciber-Físicos. No panorama atual, as Redes de Sensores Sem Fios Industriais são uma tecnologia-chave, capaz de fornecer micro-inteligência, e mobilidade, a um baixo-custo, reduzindo cada vez mais os ciclos de produção industrial, e permitindo novos tipos de aplicações. Por esta razão, durante a última década, várias tecnologias baseadas na norma IEEE802.15.4 foram desenvolvidas e propostas, oferecendo técnicas de transmissão mais fiáveis e determinísticas. Ainda, no domínio da segurança, assistimos também a uma mudança de paradigma neste tipo de sistemas. O paradigma utilizado até então, regia-se através de políticas de segurança que privilegiavam o isolamento. Porém, a conexão destes sistemas à Internet origina um novo conjunto de ameaças externas, que tem crescido progressivamente. De modo a manter a fiabilidade, as ferramentas de monitorização em ambiente de produção permitem uma constante monitorização dos sistemas, prevenindo eventuais falhas. Contudo, existe uma ausência de ferramentas para normas como o WirelessHART, ISA100.11a, WIA-PA e ZigBee, ao contrário do que acontece no caso das tecnologias legadas. Esta lacuna pode ser explicada pelas diferentes características presentes nos dispositivos IoT, como por exemplo, a fragmentação dos sistemas operativos, a necessidade de desenvolver *firmware* específico para cada aplicação, e os diferentes tipos de arquitecturas de *hardware* existentes.

O trabalho desenvolvido nesta tese, apresenta um novo modelo de arquitetura de monitorização, não só capaz de monitorizar as tecnologias industriais baseadas na norma IEEE802.15.4, como também os próprios componentes internos dos nós-sensores (em diferentes arquiteturas de *firmware* e *hardware*). O modelo de arquitetura proposto apresenta técnicas que permitem obter métricas de estado sem custos, partilhadas através de protocolos de gestão, por agentes responsáveis pela respetiva aquisição. Para confirmar o baixo impacto da arquitetura proposta foi criada uma *testbed* utilizando a norma WirelessHART, com todos os agentes. Adicionalmente, para provar a eficácia e utilidade da arquitetura foram desenvolvidos dois sistemas de deteção de anomalias: o primeiro permite a deteção de anomalias de rede; e o segundo possibilita a deteção de anomalias no *firmware* e *hardware* nos nós-sensores. Estes sistemas foram avaliados, através da injeção de anomalias de rede, *firmware* e *hardware*. Os dois sistemas de deteção propostos conseguiram identificar os comportamentos anómalos com alto *recall* e baixo *false positive ratio*, provando assim, que o modelo proposto poderá ser utilizado como ferramenta de diagnóstico em redes de sensores sem fios industriais.

Palavras-chave: Monitorização; IoT Industrial; Redes de Sensores Sem

Fios; Detecção de Anomalias; WirelessHART; ISA100.11a; WIA-PA; ZigBee; Indústria 4.0; Taxonomia de Falhas; Ferramentas de Ataque; Injeção de Anomalias.

Foreword

During this PhD program, the practical knowledge acquired working in industrial projects, and the theoretical and deep concepts acquired in the academic projects, contributed to the development of this work. Working in the industrial field, in a company like eneida® Wireless & Sensors, helped me to have a clear and broader vision in the development of industrial products, starting with the first phases (requirements and hardware design), to the deployment of the products in fields like oil&gas, mines and electric. At the same time, working as a researcher in different projects of the Communication and Telematics (CT) group of the Centre for Informatics and Systems of the University of Coimbra (CISUC) helped me to explore in deep new technologies, tools and concepts presented by the academic community, giving me a clear vision of the future of these technologies. Thus, in this section some of the academic and industrial projects are presented, as well as the contribution of each of the topics addressed in this thesis.

I-LOCATOR Project : Eneida Precise Real-Time Industrial Location (I-LOCATOR), co-financed by QREN (24842/2012). The aim of this R&D project was the development of a real time location system of goods and people in highly demanding industrial environments, like refineries and mines, using WSN and industrial wired networks. The activities performed in the scope of this project was the study of the maximum theoretical throughput of the networks, in order to guide the deployment of the network in industrial environments. To this thesis, the project contributed by being the first contact with wired base standards like CAN, and the study of the different MAC layer approaches in wired and wireless standards, presented in the section 2. The results of the project were two national patents, and a co-author publication in the IECON conference.

iCIS Project : Intelligent Computing in the Internet of Services (iCIS) project (CENTRO-07-ST24-FEDER-002003), co-financed by QREN, in the scope of the "Mais Centro" Program. The goal of this project was the study of the current state of the art in WSN faults. In this project a new taxonomy of faults for WSN was proposed and developed. It is presented in section 3. The achieved results were published in the Journal of Network and Systems Management (JNSM).

e-STEAM Project : Eneida Sensing Transmitters With Energy Harvesting For Assets Monitoring (E-STEAM), co-financed by QREN (38650/2013). The aim of this R&D project was the development of two smart sensors that monitor the condition of steam valves and steam traps, using a low-power mesh wireless network standard, the WirelessHART. The activities performed in this project were the development of the firmware to interact with the WirelessHART radio, as well as all the network configurations needed to the scenario. The testbed developed in this project was used to test and evaluate the architectural model proposed in this thesis. The work performed in this project contributed to section 4, 5 and 6, and to the publications in the LCN, NCA and WoWMoM conferences, and in the MDPI Sensor journal.

Socialite Project : Social-Oriented IoT Architecture, Solutions and Environment project (PTDC/EEI-SCR/2072/2014), co-financed by COMPETE 2020 Program. The aim of this project was to explore the developed middleware and services in people-centric contexts, with the aim of demonstrating their use in enhancing the autonomy and quality of life of citizens. The activities performed in the context of this project were the development and configuration of all middleware services responsible to: represent and store the data context, secure and make data anonymous; and interact with IoT devices like smartphones, smartwatches and WSN. The work performed in this project contributed to this thesis in the management and data representation domain, presented in section 2 and 4, and in some co-author publications in SOCIALSENS, INFOCOM, RecPad, IEEE SENSORS, WoWMoM and PAMS conferences.

This work was funded by the following grants:

Project grant Intelligent Computing in the Internet of Services (iCIS) project (CENTRO-07-ST24-FEDER-002003), from September 2014 to June 2015.

Project grant Social-Oriented Internet of Things Architecture, Solutions and Environment (SOCIALITE), June 2016 to June 2019.

The outcome of the design, experiments, and assessments of several mechanisms on the course of this thesis resulted in the following publications:

Journal papers:

- D. Raposo, A. Rodrigues, S. Sinche, J. Sá Silva, and F. Boavida, **"Industrial IoT Monitoring: Technologies and Architecture Proposal"** Sensors, vol. 18, no. 10, 2018. Impact factor: 2.475
- D. Raposo, A. Rodrigues, J. S. Silva, and F. Boavida, **"A Taxonomy of Faults for Wireless Sensor Networks"**, Journal of Network and Systems Management, pp. 1-21, 2017. Impact factor: 1.75

Conference papers:

- D. Raposo, A. Rodrigues, S. Sinche, J. S. Silva, and F. Boavida, **"Security and Fault Detection in In-node Components of IIoT Constrained Devices"** in 2019 IEEE 44th Conference on Local Computer Networks (LCN), 2019.
- D. Raposo, A. Rodrigues, S. Sinche, J. S. Silva, and F. Boavida, **"Securing WirelessHART: Monitoring, Exploring and Detecting New**

Vulnerabilities" in 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), 2018, pp. 1–9.

- D. Raposo, A. Rodrigues, J. S. Silva, F. Boavida, J. Oliveira, and C. Herrera, **"An autonomous diagnostic tool for the WirelessHART industrial standard"**, in 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016

Cooperation papers:

- J. Fernandes, **D. Raposo**, S. Sinche, N. Armando, J. S. Silva, A. Rodrigues, L. Macedo, H. Oliveira, and F. Boavida, **"A Human-in-the-Loop Cyber-Physical Approach for Students' Performance Assessment"**, in SOCIALSENS 2019, the Fourth International Workshop on Social Sensing, Montreal, QC, Canada 15th April, 2019
- J. Fernandes, **D. Raposo**, N. Armando, S. Sinche, J. S. Silva, A. Rodrigues, V. Pereira, F. Boavida, **"An Integrated Approach to Human-in-the-Loop Systems and Online Social Sensing"**, in IEEE CAOS19,The First IEEE INFOCOM Workshop on the Communications and Networking Aspects of Online Social Networks, Paris, France 29th April, 2019
- R. Sharma, B. Ribeiro, A. M. Pinto, A. Cardoso, **D. Raposo**, A. Ngombo, A. Rodrigues, J. S. Silva, H. Oliveira, L. Macedo and F. Boavida,**"Unveiling Markers of Stress Via Smartphone Usage"**, in Pattern Recognition (RecPad), 2018
- S. Sinche, J. S. Silva, **D. Raposo**, A. Rodrigues, V. Pereira and F. Boavida, **"Towards Effective IoT Management"**, in IEEE Sensors 2018 international conference, New Delhi, India, 28-31 October 2018., 2018
- S. Sinche, O. Polo, **D. Raposo**, M. Fernandes, F. Boavida, A. Rodrigues, V. Pereira, and J.S. Silva, **"Assessing Redundancy Models for IoT Reliability"**. In IEEE 19th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2018.
- N. Armando, **D. Raposo**, M. Fernandes, A. Rodrigues, J. S. Silva, F. Boavida, **"WSNs in FIWARE – Towards the Development of People-centric Applications"**, Proceedings of PAAMS 2017 - 15th International Conference on Practical Appl", in PAAMS 2017 - 15th International Conference on Practical Applications of Agents and Multi-Agent Systems, 2017 [10.1007/978-3-319-60285-1_38]
- A. Reis, D. S. Nunes, H. M. Aguiar, H.B.P.D. Dias, R. Barbosa, A. Figueira, S. Sinche, **D. Raposo**, V. Pereira, J. S. Silva, F. Boavida, A. Rodrigues and C. Herrera, **"Tech4SocialChange: crowd-sourcing to bring migrants experiences to the academics"**, in Global Humanitarian Technology Conference (GHTC), 2016
- A. Figueira, D.S. Nunes, R. Barbosa, A. Reis, H.M. Aguiar, S. Synche, A. Rodrigues, V. Pereira, H.B.P.D. Dias, C. Herrera, **D. Raposo**, J.S. Silva,

F. Boavida, **“WeDoCare: A Humanitarian People-centric Cyber-Physical System for the benefit of Refugees”**, in Global Humanitarian Technology Conference (GHTC), 2016

- A. Reis, D.S. Nunes, H.M. Aguiar, H.B.P.D. Dias, R. Barbosa, A. Figueira, A. Rodrigues, S. Synche, **D. Raposo**, V. Pereira, J.S. Silva, F. Boavida, C. Herrera and C. Egas, **“Tech4SocialChange - Technology for All”**, in International Conference on Innovations for Community Services, 2016
- D.S. Nunes, **D. Raposo**, D. Silva, P. Carmona, and J.S. Silva, **“Achieving Human-Aware Seamless Handoff”**, in 7th International Workshop on Performance Control in Wireless Sensor Networks (PWSN15). Workshop of DCOSS 2015, 2015
- P. Carmona, D.S. Nunes, **D. Raposo**, D. Silva, C. Herrera, and J.S. Silva , **“Happy Hour - Improving Mood With An Emotionally Aware Application”**, in 15th International Conference on Innovations for Community Services (I4CS), 2015
- J. Oliveira, S. Semedo, **D. Raposo** and F. Cardoso, **“Place&Play industrial router addressing potential explosive atmospheres”**, in IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society, 2014
- D. Nunes, T.-D. Tran, **D. Raposo**, A. Pinto, A. Gomes, and J. S. Silva, **“A web service-based framework model for people-centric sensing applications applied to social networking,”** Sensors, vol. 12, no. 2, 2012.

Patents:

- J. Oliveira, F. Cordeiro, N. Sousa, T. Dien, **D. Raposo**; PT 108763 A – **“System and Method for Energy Saving in Wireless Sensors Networks”**; patent filled in 2015, August 10
- J. Oliveira, F. Cordeiro, N. Sousa, T. Dien, **D. Raposo**; PT 107579 A – **“System and Method for Real-time, Three-dimensional Accurate Location”**; patent filled in 2014, April 10.

Contents

List of Figures

List of Tables

Acronyms

ACLK	Auxiliary Clock
AES	Advanced Encryption Standard
AIC	Availability, Integrity and Confidentiality
AODV	Ad hoc On-Demand Distance Vector
APP	Application
APS	Application Sub-Layer
ARM	Advanced RISC Machine
ASN	Absolut Slot Number
BK	Broadcast Key
BSP	Board Support Package
CAP	Contention Access Period
CAPEX	Capital Expenditure
CBC-MAC	Chipher Block Chaining Message Authentication Code
CCA	Clear Channel Assessment
CCS	Code Composer Studio
CFP	Contention Free Period
CIL	C Intermediate Language
CLK	Clock
COAP	Constrained Application Protocol
COMI	CoAP Management Interface
CPS	Cyber Physical System
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CT	Communication and Telematics
DLL	Data Link Layer
DLPDU	Data Link Layer Protocol Data Unit
DWT	Data Watch Point

DoS	Denial of Service
ECU	Electronic Control Unit
ENISA	European Network and Information Security Agency
ETM	Embedded Trace Macrocell
FF	Foundation Fieldbus
FIFO	First In First Out
HIL	Hardware in Loop
HTTP	Hypertext Transfer Protocol
ICS	Industrial Control Systems
ICT	Information Communication Technologies
IDE	Integrated Development Environments
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IIOT	Industrial Internet of Things
IOT	Internet of Things
IP	Internet Protocol
ISA	International Society of Automation
ISM	Industrial, Scientific and Medical
IWSN	Industrial Wireless Sensor Network
IoT	Internet of Things
JK	Join Key
JTAG	Joint Test Action Group
LNMP	LoWPAN Network Management Protocol
LPM	Low Power Mode
LwM2M	Lightweight M2M
M2M	Machine-to-Machine
MAC	Media Access Control
MCLK	Master Clock
MCU	Microcontroller Unit
MIB	Management Information Base
MIC	Message Integrity Check

MISO	Master Input Slave Output
MOSI	Master Output Slave Input
MQTT	Message Queuing Telemetry Transport
MTTR	Mean Time to Repair
MTU	Maximum Transmission Unit
NETCONF	Network Configuration Protocol
NK	Network Key
NLPDU	Network Layer Protocol Data Unit
NM	Network Manager
NWK	Network
OCSVM	One Class Support Vector Machine
OEM	Original Equipment Manufacturer
OID	Object Identifier
OPEX	Operational Expenditure
OS	Operating System
OSI	Open System Interconnection
PAN	Personal Area Network
PDR	Packet Delivery Ratio
PHY	Physical
PL	Packet Loss
PLC	Programmable Logic Controller
QoS	Quality of Service
RAM	Random Access Memory
RFC	Request for Comments
RSL	Receive Signal Level
SA	Security Administrator
SCADA	Supervisory Control and Data Acquisition
SK	Session Key
SMCLK	Sub-main Clock
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SPI	Serial Peripheral Interface

SRA	Safety, Reliability and Availability
TAI	International Atomic Time
TDMA	Time Division Multiple Access
TP	Transport
TSCH	Time Synchronized Channel Hopping
UAO	User Application Object
UART	Universal Asynchronous Receiver-Transmitter
UK	Unicast Key
UTC	Universal Time Coordinated
VCR	Virtual Communication Relationship
WDP	Watch Dog Processor
WK	Well-known Key
WSN	Wireless Sensor Network
XML	Extensible Markup Language
YANG	Yet Another Next Generation
kNN	k Nearest Neighbours

Chapter 1

Introduction

"Success consists of going from failure to failure without loss of enthusiasm"

(Winston Churchill)

Contents

IoT wireless based solutions are growing and being deployed in the industrial field. However, at the same time, there is a lack of post-deployment tools to monitor these technologies. Such tools are needed as an answer to the security challenges bought by the connection of the industry to the Internet, and by the increase of hardware and firmware complexity. The work presented in this thesis aims the proposing of a monitoring architecture that can be used to improve the reliability and security of current industrial IEEE802.15.4 based technologies by supporting monitoring at all domains (network itself, node firmware, and node hardware). The proposed solution proves that it is possible to build new post-deployment tools that monitor the network and the in-node components of sensor nodes, with low impact on sensor nodes resources and without the need of extra hardware. The motivation and problem statement of this thesis are presented below, followed by the main objectives and contributions. At the end of the chapter the outline of this thesis is presented.

1.1 Motivation and Problem Statement

The Internet of Things (IoT) currently makes it possible to have a world sensed by and connected to all kinds of devices. Wireless Sensor Network (WSN) technology is the key for connecting physical and virtual environments. This technology is growing up so rapidly that in 2011 Cisco-IBSG estimated that globally there would be 50 billion interconnected "things" by 2020 [Evans, 2011]. The IoT paradigm leads to an extremely large number of new opportunities and technical challenges in several fields, and in the industrial field in particular. In industry, wired technologies continue to be prevalent [Chang et al., 2016; Petersen and Carlsen, 2011]. Digital technologies like ModBus, ProfiBus, CanBus, HART [Nobre et al., 2015], and even analogue technologies like 4-20mA [Kim et al., 2008], are used to monitor and control most processes. Despite the high reliability of such technologies, proven over many years, wired technologies are expensive, difficult to install, time consuming, and unable to cope with the requirements of Cyber Physical Systems (CPSs) and Industry 4.0. In Industry 4.0, CPSs will confer micro intelligence (namely processing and networking capabilities) to industrial objects, reducing even further today's already short production cycles [Trappey et al., 2016]. Thus, WSNs or, more specifically, Industrial Wireless Sensor Networks (IWSNs), are fundamental for meeting the requirements of Industry 4.0.

IWSNs characteristics like low operating costs, self-organization, self-configuration, flexibility, rapid-deployment, and easy upgrading, make them ideal to industrial scenarios. However, despite all of these favourable characteristics, the adoption of WSNs in industry requires standards, dependability, ease of use, network security, extended battery lifetime, low cost, and IP connectivity [Wang and Jiang, 2016]. In recent years, considerable effort was made in order to design technologies that meet these requirements, and to standardize IWSN solutions. Standards like IEEE 802.15.4[IEEE, 2006] and IEEE 802.15.1[IEEE, 2002] are the technology foundation of many industrial applications for process and factory automation. IEEE 802.15.4 is the base technology for standards such as ZigBee[Alliance, 2015], WirelessHART[IEC, 2010], ISA100.11a[ISA, 2009], and WIA-PA[IEC, 2015]. These are widely used in process automation applications in the areas of chemical manufacturing, pulp & paper, oil & gas, and glass & mineral. On the other hand, IEEE 802.15.1 is the base technology for standards such as WISA[Scheible et al., 2007] and WSAN-FA[Wang and Jiang, 2016], widely used in factory automation applications in the areas of assembly process for automotive, consumer products and electronics [Zand et al., 2012a].

Nevertheless, although standards compliance is necessary, it is not enough to guarantee IWSN reliability per se. Sensor node components, either at hardware or firmware levels, and the network itself, can be at the root of a variety of faults (see chapter 3). Sensor nodes are inherently resource-constrained devices in terms of energy, processing power and memory capacity. In this respect, the Internet Engineering Task Force (IETF) recently defined three classes of devices [Bormann et al., 2014]: Class 0 for devices with less than 10KB of RAM and 100KB of flash memory; Class 1 for devices with around 10KB of RAM and 100KB of flash; and Class 2 for devices that have more resources but are still quite constrained when compared to high-end devices. In addition to the mentioned device constraints, sensor, network, and application heterogeneity lead to extremely complex IWSN solutions and, consequently, to fault-proneness. These characteristics impose adequate, carefully-designed strategies in the development of sensor nodes firmware, hardware architectures, and operating systems (OSs) kernel (e.g., choosing between exokernel, microkernel, monolithic approach or hybrid approach) [Hahm et al., 2016]. As an way of example, OEM manufacturers can build their products based on a single chip (comprising wireless communications and processing capabilities) [Instruments, 2017a], or using a microcontroller and a separate radio (connected by Serial Peripheral Interface (SPI) or Universal Asynchronous Receiver-Transmitter (UART)) [Instruments, 2017b]. Furthermore, applications may be developed on "bare metal" (which makes them very hardware-specific), or using one of the available OSs (e.g., Contiki[Dunkels et al., 2004], RIOT[Baccelli et al., 2013], FreeRTOS[Barry, 2018]). Another important component is the network. For instance, despite the inclusion of security mechanisms in all of the referred standards, there are known attacks on WirelessHART, ZigBee ISA100.11a, and WIA-PA [Alcaraz and Lopez, 2010; Raza et al., 2009; Islam et al., 2012; Qi et al., 2014]. Additionally, some of these technologies, namely WirelessHART, ZigBee, and WIA-PA, are not immune to interference from equipment complying with other standards, such as IEEE 802.11, when operating in the ISM 2.4GHz frequency band. Such problems may lead to early sensor node energy depletion, and subsequent replacement, increasing the costs of network operation.

Vulnerabilities in industrial systems have also been explored by attackers in cyberwar. Examples of attacks to wired-based technologies are the Stuxnet and the Slammer worms [Do et al., 2017], each one infecting the Supervisory Control and Data Acquisition (SCADA) software and causing significant damage in industrial assets and, consequently, economic losses. Furthermore, with the addition of micro-intelligence to these systems, the hardware and software components are becoming more complex. Thus, is inevitable that such systems contain more software vulnerabilities and, at the same time, become vulnerable to hardware failures [Chaturvedi, 2016]. The Taum Sauk incident in 2005 is an example of such failure. The incident report showed that sensors failed to indicate that the reservoir was full, and, consequently, the water overflowed, resulting in the collapse of the reservoir. In the cyber-security topic, IWSNs have been neglected by the scientific and industrial community in the last years, by proposing solutions that only address wired-based technologies. As a consequence of IWSN faulty nature, post-deployment tools are needed in order to adequately monitor IWSNs, thus contributing to the global system reliability and security.

In the last decade, a wide range of WSN post-deployment tools [Rodrigues et al., 2013] were developed. Some of them can improve the reliability of WSNs by detecting network, firmware, and/or hardware problems. These tools help developers in both deployment and post-deployment environments by making several firmware- and hardware-related metrics accessible, and by detecting problems, using for instance, sniffers or sink nodes. However, despite the effort to build such tools, most of them were designed for specific applications, require specific or dedicated hardware, consume non-negligible amounts of energy, do not implement security mechanisms, are complex to configure and/or to use, and do not allow the centralized management of multiple industrial standards like ZigBee, WirelessHART, ISA100.11a, and WIA-PA.

1.2 Objectives and Contributions

The aim of this thesis is to improve the security and reliability of IWSNs using monitoring techniques that continuously monitor the condition of current IWSN systems in post-deployment environments. This is done by a new architecture capable to monitor the condition of the sensor nodes components and the network of IEEE802.15.4 based technologies. To perform such tasks, the architecture relies in free-metrics already available in the technologies used in this field.

In order to be successfully adopted in real-deployments, the architecture is designed in order to: 1) have low impact on the sensor node resources; 2) have low impact on the network traffic; 3) introduce minimal delay in the firmware of the sensor nodes; and 4) not have a major impact on the development phase. Furthermore, the architecture is designed to be modular and compatible with different hardware and firmware architectures, and to be supported by the ZigBee, WirelessHART, ISA100.11a, and WIA-PA standards.

The development of such architecture brings attention to more specific activities, in order to design all the components and evaluate them in terms of performance and effectiveness. Specifically, an extensive review of the state-of-the-art in terms of current diagnostic tools approaches, network metrics, industrial standards techniques, and management protocols is presented, and current open issues and opportunities identified. To measure the performance and effectiveness of the architecture a WirelessHART testbed was developed. The proposed monitoring components were also developed and tested in this testbed, by measuring their impact on the system. Lastly, to measure the effectiveness of the proposed solution, several anomalies were injected in the network, hardware and firmware of the WirelessHART testbed. The data was collected by the monitoring system and then anomaly detection algorithms were used to identify the anomalies.

As result of all this work, this thesis has succeeded in producing the follow contributions:

Contribution 1, A Review of the Network Metrics and Management Protocols for IWSN

By reviewing current diagnostic tools, an opportunity was identified. Most of the proposed network tools rely on extra hardware to collect metrics. These types of solutions are expensive, when used in large IWSN deployments. Thus, a review of free available metrics in the ZigBee, WirelessHART, ISA100.11a and WIA-PA standards was made, and several metrics described in section 2. These metrics were identified in a deep analysis made to each standard. Additionally, another gap in the state of the art of management protocols was identified. Thus, section 2 also presents a review of the current management protocols in the IoT field.

Contribution 2, A Review of Hardware and Firmware Monitoring Techniques

To propose a broad architecture capable of monitoring so different aspects of the IWSN systems, a review of current monitoring techniques used to monitor the condition of hardware and the firmware of sensor nodes was done. This review presents in my point-of-view a contribution that cannot be found in current state of the art and gives important contributions to the definition of future architectures in the field (see section 2.4).

Contribution 3, A Taxonomy of Faults for WSN

Monitoring systems are intrinsically related with fault identification, fault detection, fault recover and fault prevention mechanisms. Additionally, to measure the effectiveness of anomaly detection mechanisms in the field, a comprehensive fault characterization is needed, in order to identify the characteristics, behaviour, and the impact of faults in the WSN systems. Thus, the third contribution of this thesis is a taxonomy of faults in WSNs presented in section 3. The taxonomy extends and complements existing taxonomies, but with the specifics of WSNs.

Contribution 4, A Monitoring Architecture for IEEE802.15.4 IWSN based standards

As the main contribution, section 4 presents the monitoring architecture proposed in this thesis. Relying in free metrics already available on the network standards and hardware platforms, this architecture is unique in such approach. To support the integration of the architecture in the different firmware, network and hardware architectures, an extensive analysis to the standards and technologies is made, and solutions are presented to the different layers (e.g., network transport, security, time synchronization). For instance, as a monitoring tool that sends monitoring data over networks that rely on low bandwidth, the appropriate transport services available in each standard (WirelessHART, ISA100.11a, WIA-PA, ZigBee) are selected.

Contribution 5, A new Advertisement based attack to the WirelessHART Standard

To show the effectiveness of the proposed architecture, a deep analysis to the WirelessHART security scheme was made in order to generate representative network attacks. By monitoring the network with the proposed architecture, a new attack vector was identified and presented. This new attack allows network outsiders to perform an exhaustion attack to nodes that pretend to join to the network, by forging advertisement packets. The attack is described in section 5.

Contribution 6, An Anomaly Detection System for Network Anomalies

The effectiveness of the monitoring architecture was proved in the three monitoring domains. In the network domain, several attacks were conducted over the WirelessHART testbed, and the monitoring data collected using the monitoring architecture. Here, it was proved that using the in-node metrics and the network metrics available in the WirelessHART testbed it was possible to detect the injected anomalies, using outlier classifiers like One Class Support Vector Machine (OCSVM). This contribution is presented in section 5.

Contribution 7, An Anomaly Detection System for Hardware and Firmware Anomalies

In the same way as the last contribution, the firmware and the hardware domains of the architecture were also tested, by the injection of different anomalies in these two domains. Here, a representative firmware attack was conducted, and several hardware anomalies injected in the testbed. The monitoring data collected was used to compare three different types of anomaly classifier approaches: a linear model based, a proximity based, and a neural network based. The results show that the proposed architecture can also be used with good results, in the detection of firmware and hardware anomalies.

1.3 Outline of the Thesis

The remainder of this thesis is organized in 7 chapters. Chapter 2 presents the state of the art on industrial technologies, metrics and current diagnostic tools approaches from the hardware, firmware and network perspectives. Chapter 3 presents a novel taxonomy of faults for WSNs, that allows a fault characterization using 11 viewpoints. Chapter 4 presents the monitoring architecture with all the proposed components, and the recommended approaches to be used in the different standards(ZigBee, WirelessHART, ISA100.11a and WIA-PA). At the end of this chapter, a testbed using the WirelessHART standard is presented, and the impact of the monitoring architecture evaluated. Chapter 5 evaluates the effectiveness of the architecture in the detection of attacks made to the WirelessHART network. This chapter also presents a novel exhaustion attack that can be directed to systems complying with this standard. Chapter 6 presents several attacks and anomalies conducted to the firmware and hardware of the WirelessHART testbed. This chapter presents the effectiveness of the architecture in the detection of new and unknown attacks to three different anomaly detection classifiers. Lastly, Chapter 7 sums up this thesis and its contributions, and provide some insights on future work that can be conducted in the monitoring field.

Chapter 2

General Background

"The greater our knowledge increases, the greater our ignorance unfolds."

(John F. Kennedy)

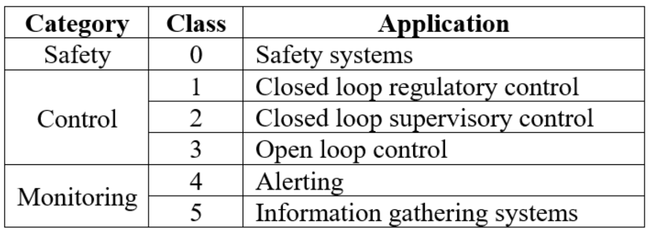
Contents

Industrial sensor networks are evolving every day, in an industry now open to the Internet, where technology is accessible everywhere. The new industrial revolution, the Industry 4.0 along with CPSs, bring new challenges for these technologies, in terms of security (the ability of the system to protect itself against accidental or deliberate intrusion), availability (the ability of the system to deliver services when requested) and reliability (the ability of the system to deliver services as specified) [Sommerville, 2010]. In this chapter, the state of the art is presented concerning the technologies and techniques used in the industrial field. This chapter starts by presenting the industrial application requirements and categories, followed by a deep analysis of the current industrial wired and wireless standards. To propose a monitoring architecture that can be supported by the four main industrial standards (ZigBee, WirelessHART, ISA100.11a and WIA-PA), hardware, firmware and network monitoring techniques are revised in order to present current approaches, and identify open issues and opportunities. Furthermore, recent management protocols are also presented and discussed. At the end of the chapter, current efforts made by the industry and academic community in the fields of security and reliability are presented in a broader perspective, identifying the lack of solutions for wireless based technologies. This chapter sets up the foundation for the monitoring architecture proposed in the next chapter.

2.1 IWSN Applications and Requirements

Over the past years, computing systems kept increasing their processing capability according to the Moore's law (either by reducing basic components size or by adding complex multi-core approaches to parallel processing). In a different path, WSN focus on the creation of smaller, more efficient and cheaper devices due to its application-dependent nature. Contrary to computer networks, WSNs follow a more single-purpose design that usually serves only one specific application-domain. So, in the design of WSN it is important the identification of the application domain and requirements in order to choose the best technology and standards.

IWSN applications have distinct requirements in comparison with common WSN applications. To identify the specific requirements for each application type, the Internet Society of Automation (ISA) created six classes [Kumar S. et al., 2014],



Category	Class	Application
Safety	0	Safety systems
Control	1	Closed loop regulatory control
	2	Closed loop supervisory control
	3	Open loop control
Monitoring	4	Alerting
	5	Information gathering systems

Table 2.1: Industrial applications defined by ISA

based on the criticality and the importance of the applications (table 2.1). Each class is classified according the message response time, the Quality of Service (QoS) requirements, and belongs to a specific industrial category such as monitoring, control and safety. The first type, the safety systems, represent all systems where actions and events are required in order of *ms* or *s* (e.g., fire alarms and safeguard systems). The second one, the closed loop regulatory control systems represent control systems where the information acquired by IWSN is used to control the industrial systems, using sometimes time requirements stricter than the safety systems. Thirdly, closed loop supervisory systems represent control systems where a reaction only occurs when a certain trend is observed. On the other hand, the open loop control represents control systems where an operator is inside the control loop. In these systems an operator analyses the data and undertakes the control of the system if a trend is observed. The alerting systems group represents monitoring systems based on regular/event-based alerts (e.g. a system that monitors the water temperature). Lastly but not least, the less strict system in time constraints is the information gathering systems. These systems are only used for data collecting and forwarding.

ISA defines industrial applications based on time and QoS requirements. However, there are other important requirements, that should also be mentioned, and taking into account when working with IWSNs. The authors of the following papers [Kumar S. et al., 2014; Zand et al., 2012a; Gungor and Hancke, 2009; Islam et al., 2012] identify some of these requirements and design goals:

- *Data Aggregation*: Different applications require different levels of data accuracy. To cope with the overhead and redundant data being forward in the network some applications require that the network support data aggregation of the sensed data in each node or in cluster-heads;
- *Energy Consumption*: In IWSN it is essential the proper management of the energy consumed by each node, in order to maximize the node/network lifetime and reduce the cost of the system (e.g. node lifetime can be improved with the minimization of radio duty cycle; network lifetime can be improved by using load balancing techniques);
- *Interoperability*: In this domain, it is essential the compatibility with existing legacy systems and others wireless solutions. For instance, some IWSN standards like WirelessHART and ISA100.11a (see section 2.2.2) already support old legacy systems like HART, but a more flexible solutions is required (e.g. by using Lightweight M2M (LwM2M) protocol);
- *Fault Tolerance, Reliability and Robustness*: In industrial critical applications, IWSNs need to be fault-tolerant and robust against failures. Using robust routing protocols, IWSNs can support topology changes and prevent network failure due to faulty nodes. Moreover, the delivery of the data between nodes must use data verification and correction methods, to prevent wireless communication errors;

- *Low-Delay*: The response time of safety systems and closed loop regulatory control systems use latencies that rounds the *ms* or *s*. In these applications, IWSNs have to insure real-time guarantees, similar to analogue wired communications (e.g. ISA50 technology);
- *Minimal Cost and Compactness*: IWSNs intend to reduce CAPEX and OPEX with the utilization of wireless communications instead of wired communications. Furthermore, sensor nodes must be small to make it easier to deploy them in large industrial networks (e.g. in some industrial applications like oil&gas, refineries and smart grid the installation time and cost can be higher because of safety and health requirements [Kelava et al., 2008]);
- *Predictable Behaviour*: In large networks, it is crucial the prediction of the network behaviour. Complex solutions tend to be more difficult to analyse, more favourable to faults and involve high costs in implementation, testing and deployment. IWSN solutions need to be simple with predictable behaviour, and proper monitoring and management systems should be used in post-deployment scenarios;
- *Quality of Service (QoS)*: To guarantee low latency in industrial systems it is essential that IWSNs have QoS mechanisms to prevent outdated data in control systems. QoS requirements can be divided in application specific QoS and network specific QoS. Application specific requirements are represented with a higher level of abstraction (e.g. the coverage of the network, the maximum number of nodes, etc). On the other hand, network specific requirements are represented with lower level of abstraction, consequently with much more detail (e.g. latency, reliability and availability requirements). As presented in the ISA industrial application classification, safety and control applications have more demanding Quality of Service (QoS) requirements;
- *Resistance to Noise and Co-Existence*: Industrial environments have many noise sources like machinery, engine vibrations, metallic frictions, humidity, temperature fluctuation and other wireless networks that communicate in ISM radio bands [Islam et al., 2012]. Moreover, IWSNs communicate with low-power signals that are more susceptible to noise. IWSNs must have specific radio techniques that prevent the disruption of the communication and guarantee the co-existence of several networks operating in the same radio band;
- *Scalability and Self Organization*: Scalability can be supported in different ways in IWSNs. Protocols and standards should be modular to allow the integration of new applications and network requirements, before and after the network deployment. Furthermore, to support large networks with a lot of nodes, IWSNs must support automatic mechanisms that configure the network keeping their QoS requirements;

- *Service Differentiation*: As a consequence of the different combination of sensor types, service differentiation is required in IWSN. Using this technique, IWSNs can assign different priorities in different ways (e.g. by node, packet, time, etc). For instance, in an IWSN that monitor and control some industrial assets, the control traffic will have higher priority than the monitoring traffic;
- *Secure Design*: Security mechanisms should be implemented in IWSNs, in order to protect the network from common attack types such as eavesdrop, interference and jamming. Moreover, without security, the network cannot assure some of the QoS requirements. The design of security mechanisms should be present in every detail of network design and should consider the IWSNs resource limitations.

This topic highlights the main industrial applications categories presented by ISA, as well as, the requirements addressed by the academic community. The identification of these requirements is important to the monitoring architecture definition. Additionally, it is important to notice that industrial networks have two main application types: control and monitoring applications. Control applications have highly restrictive QoS requirements since operator actions should be immediately executed. In a different way, monitoring application requirements are less demanding, depending only on the application requirements.

2.2 Technologies for Industrial IoT

Reliable, controlled operation, and the use of standardized technologies are key aspects to the adoption of WSNs in industrial applications [Palattella et al., 2014]. In best-effort type networks, all devices in the network obtain an unspecified data rate and delivery time, depending on the traffic load [Kobayashi, 2015]. As a result, data is delivered without any quality of service guarantees. However, in general, industrial process automation applications have stringent latency requirements. For instance, monitoring applications should guarantee an average latency of 100ms; control applications should guarantee 10ms to 100ms latency; and, lastly, safety applications should guarantee a maximum latency of 10ms [ISA, 2009]. To meet these requirements (see previous section), IWSNs may have to pre-allocate network bandwidth and physical resources, thus avoiding statistical effects that lead to insufficient bandwidth, uncontrolled jitter, and congestion losses. With these constraints in mind, this section surveys the main industrial technologies and standards (namely, HART, ModBus, IEEE 802.15.4, ZigBeePRO, WirelessHART, ISA100.11a and WIA-PA), highlighting their main features, monitoring functionalities, and applicable management protocols.

2.2.1 Industrial Wired Standards

In order to overcome the need for an industrial-level network standard, the ISA50 was proposed in 1972 [ISA, 1972] as an analogue communication standard. Despite having been replaced by more recent digital protocols (e.g., Foundation Fieldbus), it is still found in many instrumentation systems due to its use by the HART protocol. The fundamental principle of the protocol is related to the minimum current of the measuring device. A sensor that measures temperature between 0ºC and 20ºC converts this signal to an electrical current ranging from 4 to 20mA, where 20mA corresponds to a 20ºC, and the 0ºC to 4mA. By staying above a minimal level of electrical current, the system can distinguish between a zero value measurement (4mA) and a network failure corresponding to a lack of energy. The first 4mA is used to power the measurement device, while the remaining 16mA is used for control. The advantage of using 4-20mA, compared to other approaches such as 0-20mA is the capability of detecting open circuits (in which the current is zero) and the possibility of powering devices through the communication cable.

To cope with the great diversity of equipment that uses ISA50 standard, the HART protocol was created in order to enable analogue circuits to support digital communications. Developed in 1980 by Rosenmount, the HART protocol allows for an analog-to-digital signal conversion for devices that abide to the 4-20mA standard. Later on, the protocol was opened and it is now maintained by the HART Communication Foundation [FieldComm, 2014]. Sending messages on a 4-20mA network is made possible by making use of FSK modulation technique. Thus, the "1" bit is represented by a frequency of 1200Hz and the "0" bit by a 2200Hz one. This modulation does not affect the analogue communications since these are done on the 10Hz range. Regarding the transmission rate, the HART protocol communicates at 1200bps, which is quite slow by today's standards. The communication paradigm in HART adopts a Master-Slave philosophy. Communication is always initiated by the master, which sends a command message and waits for a response. On their side, the slave waits for this command and sends a response in return. Commands may belong to three types: universal, common and proprietary. The first two types of commands are specified in the protocol while the latter may be modified depending on the application. The most used network topology in HART is point-to-point. Message size may vary between 10 and 30 bytes and is composed by the following fields: preamble, start byte, address, command, number of data bytes, status, data, and checksum. Thanks to a popularity that remains even in today's world, the HART protocol is still widely used and actually has been bestowed with a wireless version known as WirelessHART (see section 2.2.2), which can be integrated with traditional HART in industrial networks.

Modbus was developed by Modicon [Fovino et al., 2009] in 1979 as a protocol independent of the link and physical layers. The protocol can operate over different physical layers, such as Ethernet and Serial. While being initially conceived to work with point-to-point topology, the protocol can be easily applied to multi-drop and peer-to-peer networks, as well as work with TCP/IP. Similarly to HART, Modbus applies a Master-Slave philosophy which allows a master device to command up to 247 slave devices. Master devices are usually computers or Programmable Logic Controllers(PLCs) while slaves tend to be simple units scattered through the terrain that acquire sensory data. Communication begins with the master device querying the PLCs about the desired data and waiting for a response. A message in the Modbus protocol is composed by the device address, function code, data bytes and error check fields. Commands sent by the master contain different function codes, which identify the operation to be executed by the slave and terminate with an error check for insuring the integrity of the transmitted data.

Lastly, the CAN protocol is a synchronous protocol introduced in 1986 by Robert Bosch GmbH [Bosch, 1991], and is the main protocol of the automotive industry. As many other network protocols, the CAN protocol follows the OSI layer model but restricts it to only three layers: Physical (PHY) layer, Media Access Control (MAC) layer and Application (APP) layer. On the PHY layer, the protocol uses NRZ-5 modulation and synchronizes the communication between the different Electronic Control Unit(ECUs) through two different types of synchronization mechanisms: hard synchronization and soft synchronization. At the MAC layer, the protocol uses the CSMA in order to avoid collisions. Every CAN message is identified by a unique ID which defines the priority of the message on the bus. In this way, when an ECU wants to transmit a message, it has to wait until to the other ECUs with minor IDs. Since these units are capable of hearing every message on the bus, the application layer includes a filter which blocks all messages except those with a specific ID. In CAN, the network maximum throughput is influenced by the length of the bus, which also determines the signal propagation time.

2.2.2 IWSN Standards

This subsection analyses the characteristic features of each of the main IWSN standards, namely ZigBee, WirelessHART, ISA100.11a, and WIA-PA. The analysis is done on a per-OSI-layer basis, starting with the physical layer and working up to the application layer. Table 2.2, below, presents a summary of the referred features, and may be used as guidance by the reader.

IEEE 802.15.4 [IEEE, 2006] is a standard for low-rate wireless personal area networks (LR-WPANs) that specifies the PHY and MAC layers. This layers design was optimized for very low-power consumption, high reliability (using mesh networks), low-data rates, and low-cost. ZigBee, WirelessHART, ISA100.11a, and WIA-PA have, all of them, adopted IEEE 802.15.4 at the PHY layer. However, because WirelessHART, ISA100.11a, and WIA-PA target worldwide adoption, they have chosen to use the 2.4 GHz frequency band only, as opposed to ZigBee, which can use the 868MHz and 915MHz bands as well [Zand et al., 2012a].

At the MAC layer, there are significant differences in the adoption of the IEEE 802.15.4 by each of the four standards. At this layer, networks can work in beacon or non-beacon modes of operation. When using the beacon mode, sensor nodes receive a specific message that is used to synchronize the network devices and, at the same time, to identify the network, and to describe the structure of the frame. Beacon networks are capable of detecting other beacon networks, and, for this reason, beacon networks can coexist in the same geographical area. Additionally, IEEE 802.15.4 uses a superframe structure in order to manage nodes channel access. The superframe is formed by three different parts: Contention Access Period (CAP), Contention Free Period (CFP), and inactive period. This superframe structure is used in ZigBee and WIA-PA, because these two standards operate in beacon mode. On the other hand, WirelessHART and ISA100.11a do not operate in beacon-mode, as their developers considered that this mode is not good enough for industrial applications (Note: WIA-PA also shares the same view, however, its authors opted for maintaining full compatibility with IEEE 802.15.4-based networks, and implement additional features at Data Link (DLL) layer). As a result, WirelessHART and ISA100.11a implemented their own superframes [Kumar S. et al., 2014]. WirelessHART superframe is composed of 10ms timeslots, while ISA100.11a can operate in any of three superframe modes (short, long, and hybrid). Finally, ZigBee can use a slotted or unslotted CSMA/CA mechanism for managing the access to the wireless medium, while the remaining standards (WirelessHART, ISA100.11a and WIA-PA) use Time Division Multiple Access (TDMA) and Carrier Sense Multiple Access (CSMA), providing a high and medium level of latency determinism, respectively.

Neither WirelessHART or ISA100.11a implement the full IEEE 802.15.4 MAC

Table 2.2: Summary of main features, adapted from [Wang and Jiang, 2016; Zand et al., 2012a]

Layer	ZigBee	WirelessHART	ISA100.11a	WIA-PA
APP	Object-oriented; Profile Defined Protocol;	Command-oriented; HART protocol;	Object-oriented; Native Protocol; Multi-wired field bus protocols;	Object-oriented; Profibus/FF/HART Protocol; Virtual Device
APS	Discovery of New Device; Binding; Fragmentation /reassembly;	-	Basic Tunnelling; Smart Tunnelling;	Data Communication and Management Services;
TP	-	Block Data Transfer; Reliable Stream Transport; Convergence;	Optional Security Features; Connectionless Services;	-
NWK	Tree Routing/ AODV Routing; Address Assignment; Network Joining/disjoining	Graph/Source/Superframe Routing	Addressing (6LowPAN); Routing Address Translation; Fragmentation/reassembly;	Addressing; Static Routing; Fragmentation/reassembly;
DL	-	Slot Timing Communication; Time Synched TDMA/CSMA; Channel Hopping;	Grapgh/Source/Superframe Routing; Slot Timing Communication; Duocast Transaction; Time Synched TDMA/CSMA; Channel Hopping;	Frequency Hopping; Aggregation and Disaggregation; Time Synchronization
MAC	IEEE 802.15.4 Mac Layer	IEEE 802.15.4 Mac Layer (partially implemented)	IEEE 802.15.4 Mac Layer (partially implemented)	IEEE 802.15.4 Mac Layer
PHY	IEEE 802.15.4 PHY 868M/915M/2.4GHz Radio Data rate: 20Kb/; 40Kb/s; 250Kb/s	IEEE 802.15.4 PHY 2.4 GHz Radio Data rate 250Kb/s	IEEE 802.15.4 PHY 2.4 GHz Radio Data rate 250Kb/s	IEEE 802.15.4 PHY 2.4 GHz Radio Data rate 250Kb/s

layer, as they consider that the MAC layer of IEEE 802.15.4 is not capable of delivering the deterministic latency needed by industrial applications. As such, these standards extend and complement medium access mechanisms with functionality at the DLL [Wang and Jiang, 2016], namely, Time Synchronized Channel Hopping (TSCH), which then evolved to the new IEEE 802.15.4.e standard. The TSCH mechanism offers two significant improvements: the possibility to have deterministic latency (communication resources are pre-allocated); and a mechanism of channel hopping that minimizes interference with nearby devices that operate in the same frequency, such as IEEE 802.11 devices. On the other hand, the WIA-PA follows the IEEE 802.15.4 standard at the DLL, including functionalities like time synchronization and frequency hopping techniques. Lastly, ZigBee does not implement any mechanism at this layer. It is also worthwhile mentioning that, in addition to extending/complementing MAC layer functionality, the DLL is also used by ISA100.11a for implementing some network-related functions, specifically in what concerns routing. In fact, this standard implements two types of routing: one at the DL layer, that handles all the IEEE 802.15.4 traffic, and another one at Network (NWK) layer, responsible for handling the IPv6 backbone traffic, as we will see below.

At the NWK layer, the choice of supported functionality and routing protocols is often influenced by the network architectures [Zand et al., 2012a]. For instance, ZigBee offers the possibility of having star, tree, and mesh topologies, and defines several field devices: coordinator, routers, and end-devices. Tree routing and Z-AODV protocols are used when the network operates in tree or mesh topology, respectively. Despite the fact that mesh networks can be considered more reliable, in ZigBee the utilization of this topology is not suitable for industrial applications, due to the overhead and non-deterministic latency of on-demand protocols like Z-AODV [Wang and Jiang, 2016]. In the case of WirelessHART, the basic network devices are: field devices, gateway, access points, and network and security manager. Typically, WirelessHART gateways support the role of security and network manager, and access point.

WirelessHART networks may operate in star or mesh topologies. However, the mesh topology is the most used one, due to its flexibility, inherent fault-tolerance, and ease of deployment and configuration. With this topology, routing can be done by using graph routing or source routing. When graph routing is used, the network manager needs to compute all graphs in the network and share them with sensor nodes. All communications using graph routing always consider two different paths to ensure reliability. In contrast, when source routing is used, network packets are forward between intermediate devices without the need for prior route information (the path of the packet is specified in the packet itself)[Petersen and Carlsen, 2011]. Differently from the other standards, ISA100.11a specifies two types of networks: the IEEE802.15.4 based network, and backbone network. In the WSN, ISA100.11a defines three device types: routing devices, field devices and handheld devices. The routing mechanisms available in this network are the same as the ones in WirelessHART. Additionally, in the infrastructure (i.e., backbone) side, ISA100.11a uses 6LowPAN, thus allowing for direct communication between external IP devices and ISA100.11a devices. Lastly, WIA-PA supports a hierarchical topology that uses star and mesh, or star-only topology. In the case of the mesh topology, the network operates using routers and gateways, while in the star topology the network is composed of routers and field/handheld devices. At this level, field devices are cluster members that acquire sensor information and send it to the cluster heads (routers). Then, the cluster-heads form the mesh network. Each routing device in the network shares its neighbour information with the network manager, and then the network manager computes and shares the static routes. For each pair of devices that want to communicate, at least two routing paths are assigned.

The Transport (TP) layer is responsible for providing host-to-host communication services between applications. As can be seen in table 2.2, only WirelessHART and ISA100.11a implement data transport functions at this layer, supporting different service level agreements. Optionally, ISA100.11a offers end-to-end security at this layer. In contrast to ISA100.11a and WirelessHART, WIA-PA provides different service level agreements at the Application Sub-Layer (APS), and not at the transport layer. The service-level agreements available in each standard are presented in section IV, sub-section B. Last but not least, ZigBee does not support any transport layer functionality nor does it support traffic differentiation at the APS. Moreover, in ZigBee, fragmentation, reassembly, and device discovery are implemented at the APS.

The application (APP) layer is the layer at which the connection between legacy systems and IEEE 802.15.4-based systems takes place [Zand et al., 2012a; Wang and Jiang, 2016]. At this layer, there are two important options that can be identified. WirelessHART uses a command-oriented approach; alternatively, ISA100.11a, ZigBee, and WIA-PA use a more flexible object-oriented approach. Object-oriented approaches are more flexible than command-oriented approaches because they allow for protocol translation by mapping attributes from one protocol to the other. As for native applications, ZigBee supports the ZigBee profiles; WirelessHART supports the HART protocol; WIA-PA supports native protocols like Profibus, Foundation Fieldbus(FF), and HART; last but not least, ISA100.11a supports the ISA100.11 application protocol.

2.2.3 IWSN Reports

IWSN standard technologies make device and network state information available to a variety of entities, e.g., network neighbours or a central management device. In general, this information is shared between nodes to compute routes, allocate bandwidth, calculate link costs between network devices, or generate alarms when critical events occur (e.g., link failure, route failure, low battery level, etc). This subsection presents the network and data link layer reports available in each of the standards being considered in this thesis [Alliance, 2015; IEC, 2010; ISA, 2009; IEC, 2015], and describes the context in which the reports are used. Table 2.3 presents a summary of the referred reports, and may be used as guidance for the discussion.

Table 2.3: IWSN Network metrics

Standard	**Network reports**	**Metrics**	**Report type**	**Visibility**
ZigBeePRO	Network Status	No route available; Tree link failure; Non-tree link failure; low battery level; No routing capacity; No indirect capacity; Indirect transaction expiry; Target device unavailable; Target address unallocated; Parent link failure; Validate route; Source route failure; Many-to-one route failure; Address conflict; PAN identifier failure; Network address update; Bad frame counter; Bad key sequence number;	Event	Coordinator or routers
	Link status	Neighbour network address; Incoming cost; Outgoing cost	Periodic	
	Network report	Radio channel condition; PAN ID conflict	Event/Periodic	
WirelessHART	Device Health	Packets generated by device; Packets terminated by device; DL mic failures; NWK mic failures; Power status; CRC errors;	Periodic	Network manager
	Neighbour Health List	Total number of neighbours; Nickname of Neighbour; Mean RSL; Packets transmitted to the neighbour; Packets received from the neighbour; Failed transmissions	Periodic	
	Neighbour Signal Levels	Total number of neighbours; Nickname of neighbour; RSL of neighbour in DB;	Periodic	
	Alarms	Path Down (Nickname of the neighbour); Source Route Failed (Nickname of the neighbour and NWK MIC from the NPDU failed routing); Graph Route Failed (Graph ID of the failed route);	Event	
ISA100.11a	Connectivity Alert per Channel	Attempted unicast transitions for all channels; Percentage of time transmissions on channel x did not receive an ACK); Percentage of time transmissions on channel x aborted due to CCA;	Periodic	System Manager
	Connectivity Alert per Neighbour	RSSI level; RSQI level; Valid packets received by the neighbour; Successful unicast transmissions to the neighbour; Unsuccessful unicast transmission; Number of unicast transmissions aborted (by CCA); Number of NACKS received; Standard deviation clock correction;	Periodic	
	Neighbour discovery Alert	Total number of neighbours; Neighbour address; Neighbour RSSI; Neighbour RSQI;	Periodic	
WIA-PA	Path Failure report	Route ID	Periodic	Network Gateway
	Device Status report	Number of sent packets; Number of received packets; Number of MAC layer mic failures detected; Battery level; Number of restarts; Uptime since last restart	Periodic	
	Channel Status report	Channel Id; Neighbour device address; LQI; Channel packet loss rate; Channel retransmission count	Periodic	
	Neighbour Reports	Neighbour address; Backoff counter; BackoffExponent; Last time communicated; Average RSL; Packets transmitted; Ack packets; Packets received; Broadcast packets;	Periodic	

The ZigBee standard comprises three report types that are used for sharing network- and node-related information: 1) link status report; 2) network status report, and 3) network report. Link status reports share sensor node's neighbours incoming and outgoing link costs. The report is broadcasted by ZigBee coordinators and routers in one-hop fashion. This report is useful during network discovery, to find neighbour devices, and in the operation phase for updating the devices' neighbour table. Network status reports are sent by devices in order to report errors and other events that arise at the network layer. The report can be sent in unicast or broadcast modes over the network, and can only pertain to an event at a time. The list of the possible reported events is presented in the Table 2.3. Lastly, network reports allow a device to report network events, like Personal Area Network (PAN) conflict, or the radio channel condition to the coordinator. As limitation, in the tree topology, some of these packets may not be received by the ZigBee coordinator, due to the presence of routing devices, that make their own routing decisions. On the other hand, when using a star topology, all the packets will be received by the ZigBee coordinator.

Contrary to what happens in ZigBee, in WirelessHART the network manager controls all communication in the network, and only authorizes new services if resources are available. Consequently, field devices (here with full network capabilities) must share node state and network based information with the network manager. In WirelessHART, all network reports are sent to the network manager by using the maintenance service. WirelessHART specifies four types of reports: 1) device health; 2) neighbour health list; 3) neighbour signal levels; and 4) alarm report. Device health reports summarize all the communication statistics of a unique field device and are periodically sent to the network manager. The statistics include generated packets by device, terminated packets by device, power status, and others.

Neighbour health list reports include statistics about the communication with all neighbours linked to a field device. These reports include the total number of linked neighbours, the mean Received Signal Level (RSL) to the neighbour, and packets and errors statistics. Neighbour signal level reports include statistics of discovered but not linked neighbour devices detected by a field device. When a device wants to connect to the network it usually sends a join request and a neighbour signal level report. Lastly, WirelessHART defines several alarm types: path down alarm; source route failed alarm; and graph route failed alarm.

Similarly to what happens in WirelessHART, in ISA100.11a networks, the system manager controls all communication in the network. However, in this standard, WSN routing takes place at the DLL, due to the use of 6LoWPAN at the NWK layer. Network metrics in ISA100.11a are shared at the MAC layer, instead of being shared at the NWK layer. ISA100.11a defines two groups of network reports: 1) connectivity alerts, and 2) neighbour discovery alert. Connectivity alerts comprise two types of reports: per-neighbour report, and per-channel report. Per-neighbour reports contain neighbours' connection statistics. On the other hand, per-channel reports contain per-channel all-neighbours statistics, and convey them to the system manager. Finally, neighbour discovery alerts are sent periodically to the system manager with a list of overheard neighbours. The system manager makes new routing decisions based on these reports. Per-neighbour reports, per-channel reports, and neighbour discovery alert, are similar to WirelessHART's neighbour health list, device health, and neighbour signal level, respectively.

In WIA-PA, route computation is also performed by the network gateway, similarly to WirelessHART and ISA100.11a. WIA-PA defines four types of reports: 1) device status report; 2) channel condition report, 3) neighbour report, and 4) path failure report. Device status reports include statistics about the condition of field devices and routers, such as the number of packets exchanged with neighbours, number of restarts, and uptime. Channel condition reports send statistics grouped by channel and neighbour to the network gateway. These statistics include link quality, packet loss rate, and number of transmission retries. Neighbour reports, also received by the network gateway, group neighbours statistics and neighbours scheduling details, such as backoff counter and exponent, transmitted and received packets, and number of acknowledgments. Lastly, path failure reports are generated when a route path failure occurs. These reports are sent by a routing device to the network gateway, whenever the retransmission counter of a specific path exceeds a given threshold.

2.3 Management of Constrained Devices

Network management emerged in traditional networks with the need to control and monitor networks as they grew up in size and complexity. ISO/IEC 7498-4 [ISO/IEC 7498-4, 1989] was one of the first initiatives to establish a management framework, by defining a set of functional areas, namely: fault, configuration, accounting, performance, and security management. Furthermore, ISO/IEC 7498-4 proposed the use of managed objects as abstractions for network resources, which, in turn, contain several attributes. In current protocols, managed objects can be defined using several syntaxes, like Structure of Management Information (SMI)v2 [McCloghrie et al., 1999], Yet Another Next Generation(YANG) [Bjorklund, 2010], and Extensible Markup Language(XML) [OMA, 2017]. Managed objects data are stored in management databases and accessed by management protocols. In this sub-section, we identify two types of management protocols: 1) protocols designed for traditional networks; and 2) protocols designed for networks of resource-constrained devices, that can also be used in some traditional networks.

As one of most used protocols for the management of IP devices, Simple Network Management Protocol(SNMP)[Case et al., 1990] provides most of the basic functionality defined in ISO/IEC 7498-4. The standard specifies an architecture based on agents and managers. Devices being monitored run SNMP agents that share the device state information with an SNMP manager, by using query/response interactions and trap notifications. Each SNMP agent has a collection of managed objects whose data is stored in a Management Information Base (MIB). Each object is identified using a specific Object Identifier (OID). Despite the success of SNMP for monitoring purposes, the protocol failed to provide an effective and reliable way to configure devices. Thus, IETF started a working group that wrote an informational RFC [Schoenwaelder, 2003] with several requirements that coming network management standards should implemented. This work was the basis for the NETCONF [Enns et al., 2011] protocol. Differently from SNMP's manager-agent architecture, Network Configuration (NETCONF) uses a client-server architecture. The server runs on a management device (SNMP agent) and shares the monitoring information (also by query/response and notification messages) with the client (SNMP manager). Additionally, when devices need to be configured, the client may send several configuration commands in one or several transactions. The transactions supported in NETCONF give operators the capability of sending out-of-order commands, and of performing rollback and commit operations.

The management protocols mentioned up to now were designed for traditional networks. In [Sehgal et al., 2012], the authors implemented traditional network management protocols in networks of constrained devices, and concluded that SNMP and NETCONF are not suitable for this type of networks. For instance, the use of TLS encryption in NETCONF adds significant overhead in terms of session time (i.e. in the order of seconds). In later years, with the growth of IoT, new solutions that address resource-constrained devices and 6LowPAN networks were proposed. These solutions also take advantage of transport protocols specially designed for resource-constrained devices, like Constrained Application Protocol (CoAP) [Shelby et al., 2014].

One of the first solutions developed with a focus on the management of 6LowPAN networks was the LoWPAN Network Management (LNMP) protocol [Mukhtar et al., 2008]. The LNMP protocol implements a solution based on SNMP that targets 6LowPAN networks. In this solution, 6LowPAN gateways convert the information from 6LowPAN networks to SNMP MIBs and make it available over IP. On the other hand, other management architectures were evaluated, and new research directions that take advantage of new technologies like HTTP appeared. In [Marotta et al., 2014], the authors evaluated the use of different architectures like SNMP, Resource-Oriented Architecture (ROA), and Service-Oriented Architecture (SOA), and concluded that ROA architectures are more suitable for resource-constrained devices in terms of response time and power consumption, and are less sensitive to changes in timeout. As a result, a RESTful version of NETCONF was created, named RESTCONF [Bierman et al., 2017]. Distinct from NETCONF, that uses SSH and TCP, RESTCONF allows the communication between server and client using HTTP operations. Using HTTP at the application layer enables clients to receive notifications without maintaining a permanent connection with the server, as it is the case of NETCONF (one of its major drawbacks). The syntax used in RESTCONF is YANG, the same

Table 2.4: Network management protocols revision

Standard	Modeling Language	Supported Operations	Resource Constrained Devices Support	Notifications Support	Used Protocols
SNMP	SMIv2	Monitoring/Configuration	No	Yes	UDP
NETCONF	YANG	Monitoring/Configuration	No	No	SSH/SSL/HTTP
RESTCONF	YANG	Monitoring/Configuration	Yes	Yes	HTTP/TLS/TCP
COMI	YANG/SMIv2	Monitoring/Configuration	Yes	Yes	CoAP/DTLS/UDP
LWM2M	XML/YANG	Monitoring/Configuration/Application management	Yes	Yes	CoAP/DTLS/UDP and SMS

syntax used in NETCONF. Also using an ROA-based architecture but with a different application protocol, the CoAP Management Interface [Bierman et al., 2017] (COMI) is an internet draft that intends to provide access to resources specified in YANG or SMIv2, using CoAP. The draft defines, as in the cases of NETCONF and RESTCONF, a separation between operational and configuration data store, the use of DTLS, and a conversion of YANG string identifiers to numeric identifiers that contributes to reducing the payload size.

LwM2M [OMA, 2017] is also a protocol designed for resource-constrained devices. This protocol provides device management and application management, an aspect that differs from the previously mentioned network management protocols. As COMI, LwM2M also supports the use of CoAP at the application layer. Being a REST-based protocol, LwM2M uses GET/PUT/POST operations to perform read/write/execute operations over the managed objects resources. In this protocol, the definition of managed objects is done using XML. An extensive list of managed objects is available in OMA [OMA, 2018]. In addition to these, OMA allows the creation of specific objects by individuals, organizations, and companies.

Summing up, in this sub-section a set of network management protocols were presented. By analysing the protocols presented here, we identified protocols suitable to managing traditional networks (switches, routers, computers), and protocols designed for networks of resource-constrained devices (e.g., sensor nodes). An important trend is that new protocols like LwM2M use general-purpose languages to describe the managed objects (e.g., XML), instead of YANG and SMIv2. All of these management protocols can be incorporated into the different gateways identified in the industrial standards presented before (i.e., ZigBee coordinator, WirelessHART network manager, ISA100.11a system manager, and WIA-PA network gateway), because these roles are performed by unconstrained devices. Finally, protocols like SNMP, NETCONF, RESTCONF, COMI and LwM2M cannot be directly used in sensor nodes, because current standard industrial technologies do not allow running application protocols other than their own. Thus, the management of these kinds of networks can only be done in a standardized way at gateway level. Only new standards like 6tisch

[Thubert et al., 2016] support the management of sensor node devices using COMI or LwM2M, because 6tisch uses the CoAP protocol at the application level. Nevertheless, due to the fact that no 6tisch-based products are available and, consequently, it is not yet used in industrial settings, 6tisch will not be addressed in this thesis.

2.4 Survey of Current Diagnostic Tools

The design of a diagnostic tool applicable to low-end IoT devices, like WSN nodes, is a tough task due to their characteristics and diversity. WSN nodes have limited resources, support a variety of application architectures, and rely on complex network mechanisms. In addition, WSN applications can be developed for a specific operating system, or even almost from scratch. These characteristics make it difficult, or even impossible, to develop a common diagnostic tool for all kinds of scenarios, and, at the same time, compatible with several operating systems. Despite this, in the last decade several diagnostic tools were proposed in order to provide inside views on WSNs' and nodes' behaviour. In [Rodrigues et al., 2013], the authors analyse an extensive set of diagnostic tools. In this section, some of the tools described in [Rodrigues et al., 2013], as well as more recent tools like [Bhadriraju et al., 2012; Dong et al., 2014; Schuster et al., 2014; Dong et al., 2013; Rodenas-Herráiz et al., 2017], will be presented from a diagnostic target perspective, organizing their presentation into network-based, firmware-based, and hardware-based tools.

2.4.1 Network Tools

The nature of wireless communications makes this technology unreliable and failure-prone. In this context, diagnostic tools that are able to collect network metrics and evaluate the state of the network are essential. Three types of approaches can be used to collect network information: 1) passive, using extra hardware to collect network information without interference; 2) active, using on-node available resources; and 3) hybrid, using a mix of active and passive approaches.

When passive approaches are used, the acquisition of network-related information is made by installing extra sniffer nodes or sniffer capabilities in the existing sink nodes. These diagnostic tools may differ in the type of sniffer nodes, in the storage approach, and in the way the gathered information is transmitted. Specifically, some of the existing solutions use sink sniffer nodes to collect traffic data from the network; others use extra networks of sniffers deployed with the main WSN; others use sniffer nodes that collect data and store it in memory; and lastly, the most expensive in terms of network installation, send network-related information by wired technologies (e.g. SNIF[Ringwald et al., 2006], SNTS[Khan et al., 2007], PDA[Romer and Ma, 2009], LiveNet[Chen et al., 2008] and L-SNMS[Yuan et al., 2015]).

Differently from passive tools that rely on extra hardware to monitor the network, active approaches use on-node metrics already available in sensors nodes. Active tools are easy to install, do not need extra hardware and, consequently, are less expensive when compared to passive approaches. However, active tools may have impact on node resources, as memory, energy, processing, and network throughput are needed to collect, store, and transport network-related information. Some of the tools gather traffic metrics from the sensor nodes' operating system, others from the network layer, and others directly from sink nodes. Also, there is a specific set of tools that use software scripts deployed in sensor nodes to collect statistics about the network traffic seen by the node. Subsequently to data acquisition, the transport of network-related information can be made using the main application channel or a secondary channel. To minimize the impact of data transport, some tools use compression and aggregation techniques so as to reduce the traffic overhead. Examples of this type of tools are Marionete [Whitehouse et al., 2006], Megs [Lodder et al., 2008], Memento[Rost and Balakrishnan, 2006], Wringer[Tavakoli et al., 2008], 6PANview[Bhadriraju et al., 2012], and D2[Dong et al., 2013].

Lastly, hybrid approaches use a mix of methods described in the cases of the active and passive approaches. Examples of this type of tools are Sympathy [Ramanathan et al., 2005] and Dustminer[Khan et al., 2008].

2.4.2 Firmware Tools

Another source of faults commonly addressed by diagnostic tools is the sensor nodes' firmware. After firmware development, sensor nodes are deployed in the field and, in some cases, faults may stay in a dormant state until an input activates them. If the error is not properly handled, a firmware failure may occur and sensor node data may not be delivered or may be corrupted. In an extreme although not uncommon situation, a firmware fault may even prevent a sensor node from entering sleep mode and, eventually, lead to battery exhaustion. With the aim of promptly detecting firmware faults, there are diagnostic tools that help developers in either the development phase, or in the WSN deployment phase.

Tools that are used during the development phase usually require debugging interfaces, specific hardware implemented in the microcontrollers architecture, and Integrated Development Environments (IDEs) that allow accessing the available functionality. Examples of these technologies are: the old Joint Test Action Group (JTAG) interface used to program microcontrollers and to have access to special internal registers (e.g., hardware and software breakpoints); UART ports used for outputting log messages that are useful for debugging purposes; the EnergyTrace[Instruments, 2017d] technology, from Texas, that allows developers to measure the impact of firmware on nodes energy consumption; and the CoreSight[Arm, 2017] technology, implemented by Advanced Risc Machine (ARM) in their microcontrollers, which allows performance profiling, memory access, real-time tracing, and software debugging through a new type of interfaces, namely the Serial Wire Debug and the Serial Wire Output Pin (examples of variables that can be access using this type of technology are cycles per instructions, sleep cycles, and exceptions). On top of that, IDEs like Code Compose Studio (CCS)[Instruments, 2017c], among others, enable access to these tools and help developers to detect code faults.

None of the development phase diagnostic technologies are suitable for post-deployment, because they require a wired connection between nodes and the tool's hardware. Consequently, in the last decade, several deployment phase tools were proposed that are able to provide monitoring information, although with considerable limitations when compared to the functionality delivered by development phase tools. Some examples are [Schuster et al., 2014], Nucleos[Tolle and Culler, 2005], Enverilog[Luo et al., 2006], Marionete[Whitehouse et al., 2006], Clairvoyant[Yang et al., 2007], NodeMD[Krunic et al., 2007], L-SNMS[Yuan et al., 2008], LIS[Shea et al., 2009], Memento[Rost and Balakrishnan, 2006], Dustminer[Khan et al., 2008], DT[Cao et al., 2008], Tracealyzer[Holenderski et al., 2010], and Dylog[Dong et al., 2014]. Despite their limitations, these tools allow WSN operators/managers to collect operating systems variables (e.g., task queue state, number of reboots), main application variables, and application events and states (like function call traces). This information is usually stored in flash, RAM, or sent via the application main channel using logs. In order to deliver all of these without requiring an extra effort during the main application development, code instrumentation techniques are used, which make it possible to automate the process of adding firmware metrics collection and transmission functionality. Firmware data is usually sent using one of two paradigms: event-driven or query-driven. Additionally, some of the tools implement extra functionality, such as: source-level debugging, offering commands similar to hardware debugging (break, stop, watch, backtrace, etc); remote call specific functions; remote reprogramming; and specification of trace log events triggering conditions. Tools that usually read and write large amounts of data from/to flash memory (e.g., Tracealyzer) are not, in general, appropriated for industrial WSN technologies due to the energy wasted in the process.

2.4.3 Hardware Tools

Finally, a source of faults that diagnostic tools also commonly address is hardware faults. Hardware faults may occur before and/or after WSN deployment, and are due to one or more of several reasons, such as: bad hardware design, external environment phenomena, aging, and extreme temperatures. Hardware diagnostic tools can be divided in two groups: 1) external-physical-tools that are used by developers and operators; and 2) on-node-tools that infer hardware faults using available hardware resources.

The first group of tools consists of physical tools/equipment that operators and developers use to check hardware condition and the occurrence of faults. Tools like oscilloscopes, multimeters, and logic analysers are connected to the target system in order to check the condition of the electronics and the existence of faults. These tools are frequently used during the development phase, where developers search for electronics defects and for communication problems between hardware modules (e.g., using logic analysers [Saleae, 2018]).

On-node-tools use a different approach. This type of tools is designed during the hardware and/or firmware development phase, with the specific purpose of collecting state information from the different modules using on-node components. In [Scherer and Horváth, 2012], the authors use the ARM CoreSight technology to create a WDP (Watch Dog Processor), by polling the main processor memory variables and setting conditions on these variables. Also using the same technology, in [Scherer and Horvath, 2014] the authors extend Hardware in Loop (HIL) tests, by incorporating some metrics provided by the CoreSight technology. Moreover, in [Dutta et al., 2008] the authors propose a technique to read the energy wasted in boards that use switching regulators. By collecting these metrics, operators and developers are able to detect hardware faults that occur during the deployment phase without using external tools.

2.5 Solutions and Approaches: Security and Reliability

Information Communication Technologies (ICTs) are changing the industrial mindset. Nevertheless, Industrial Control Systems(ICSs) significantly differ from common ICT systems, when system requirements and priorities are considered. In ICS, availability is one of the most important requirements, followed by integrity and confidentiality (AIC). Nevertheless, the European Network and Information Security Agency (ENISA) [Knowles et al., 2015] are providing a new alternative definition for the field. According to them, current ICSs should privilege safety, reliability and availability (SRA), instead of AIC. Moreover, SRA is recognised as being intrinsically related with security.

Regarding the increase of cyber security attacks to ICSs, the security of ICSs have been scrutinized from different perspectives. Governments, industry, and standardization bodies have been proposing new standards, guidelines, and best practices for the various industrial domains (e.g., oil & gas, chemical, nuclear) [Knowles et al., 2015]. At the same time, in the academic research community, a deluge of different solutions that focus on ICS technologies and standards have been proposed. In general, some of these solutions try to incorporate already known techniques used in ICT systems. Examples of such solutions are: software layer solutions; firewalls that prevent specific attacks to industrial protocols like DNP3; Intrusion Detection Systems (IDSs), using open-source technologies like

Snort; honeypots, that try to prevent attacks to real systems and, at the same time, collect attack vectors; hardware monitoring solutions (Shadow Security Units), that are directly attached to control devices for monitoring their behaviour [Graveto et al., 2019]; and, lastly, machine-learning-based IDSs. Nevertheless, part of these efforts only address problems related with the introduction of ICT networking technologies, such as Ethernet and IP in industrial systems, and/or with network components, and do not address critical security issues that can also be found in hardware and firmware components, as well [Robertson and Riley, 2018].

Despite the fact that ICT-based solutions are increasingly being used in critical infrastructures and application areas, of which the nuclear field [Kim et al., 2018] is an example, little effort is being done in order to develop efficient and effective post-deployment monitoring solutions with emphasis on security and reliability. Some architectural proposals, nevertheless, target specific standards, like [Kim et al., 2019]. However, these solutions usually address the network part only, and do not propose solutions for hardware and firmware security.

On the other hand, reliability is usually achieved with fault-tolerance techniques and diagnostic tools. In the WSN domain, fault-tolerance techniques are a well-known field of research. There are several proposals that address network faults, as well as several works that try to prevent faults in sensor readings [Chouikhi et al., 2015]. The next chapter presents some of these techniques and tools, that can be used in WSN-based solutions and applied to industrial wireless standards.

2.6 Summary of the Chapter

Security and reliability are, nowadays, two important requirements in industrial systems, that keep evolving in order to deal with some of the challenges presented by the Industry 4.0. In terms of security, a special attention has been devoted to the development of new monitoring systems, that rely on techniques and solutions used on the ICT field. However, part of this effort is only made for legacy systems, based on wired standards and technologies.

As presented in section 2.1, IWSN differ from other technologies, providing great benefits for industrial applications but with some constrains. At the same time, the technology is fragmented. There are several hardware architectures, several firmware approaches, and different network protocols, hampering the development of monitoring systems like the ones that exist for wired technologies. At the same time, in the research and industry communities there are some proposals of monitoring techniques, specific for each field. Some of these techniques can be used for free to monitor IWSN systems, and at the same time, increase their reliability and security. For instance, current industrial network standards share network metrics to perform routing operations, however, the use of these metrics in monitoring systems could not be found in any similar proposal. Lastly, with the increase of IoT solutions, and the increase of Machine-to-Machine (M2M) communications, new management protocols appeared in the last years, improving the interoperability between different solutions. All these technologies together allow the proposal of a new monitoring architecture, that is presented in chapter 4.

Chapter 3

WSN Faults

"Simplicity is prerequisite for reliability."

(Edsger Dijkstra)

Contents

Over the last decade, WSNs went from being a promising technology to the main enabler of countless IoT applications in all types of areas. In industry, WSNs are now used for monitoring and controlling industrial processes, with the benefits of low installation costs, self-organization, self-configuration, and added functionality. Nevertheless, despite the fact that base WSN technologies are quite stable and subject to standardization, they have kept one of their main characteristics: fault-proneness. As presented in section 2.4, in recent years considerable effort has been made in order to provide mechanisms that increase the availability, reliability and maintainability of this type of networks. In this context, a whole range of techniques such as fault detection, fault identification and fault diagnosis used in other research fields are now being applied to WSNs. Unfortunately, this has not led to a consistent, comprehensive WSN fault taxonomy that can be used to characterize and/or classify faults. Neglecting the importance of WSN fault characterization (e.g., when using supervised and semi-supervised algorithms for anomaly detection) may lead to bad classifiers and, consequently, bad fault handling procedures and/or tools. In this chapter, we start by reviewing base fault management concepts and techniques that can be applied to WSN. We then proceed to propose and present a comprehensive WSN fault taxonomy that can be used not only in general purpose WSNs but also in IWSN. Finally, the proposed taxonomy is validated by applying it to an extensive set of faults described in the literature. Additionally, it will be used in chapters 5 and 6 that focus on injecting network, hardware, and firmware anomalies in a WirelessHART testbed.

3.1 Introduction

Despite the benefits of using WSN in industrial applications, there are also some drawbacks that are delaying the deployment of IWSNs, specifically their fault-prone nature, as proved by many deployment experiences [Warriach et al., 2012]. In general, sensor nodes are resource-constrained and fragile, sensor readings can be incorrect, network links are failure prone, network congestion is common, packets may be corrupted or lost [Paradis and Han, 2007] and, lastly, sensor node components may also fail [Mahapatro and Khilar, 2013]. This faulty nature of WSNs not only make the maintenance of sensor nodes and networks difficult, but may also result in severe consequences for human lives, the environment, and the economy, when used in critical WSN applications. Moreover, WSNs may serve different types of applications with distinct requirements, as potentiated by the widespread use of WSN technology in daily life, making fault detection extremely difficult. In this IoT scenario, sensor nodes must have mechanisms that monitor and detect their own faults as well as neighbors' faults. Additionally, mechanisms for detecting network faults and global application faults must be also in place.

Increasing the quality of hardware and software in production and development processes may be one possible solution for decreasing fault occurrence in WSNs.

However, this alone will not prevent the occurrence of faults and it will increase the cost of WSNs. Thus, in recent years, significant attention has been given to using several techniques such as fault detection, fault identification, fault tolerance, fault diagnosis, fault injection and anomaly detection. Using these techniques, some authors proposed solutions that: detect faults in sensors, thus increasing the quality of sensor data [Warriach et al., 2012]; detect faults in the routing layer, using a fault tolerant routing mechanism [Ma et al., 2006]; inject faults at assembly level, thus detecting hardware and software faults at design phase [Cinque et al., 2009]; and increase the performance of the network, detecting network and global faults.

Despite the growing concern with WSN fault management and the existence of several surveys on fault tolerance applied to WSN [Mahapatro and Khilar, 2013; de Souza et al., 2007], there is no full, comprehensive, consistent, generally adopted WSN fault taxonomy, as existing work either concentrates on describing fault causes or on presenting techniques and algorithms for dealing with specific faults, without trying to characterize and understand the nature of faults in WSN. In our view, this partial analysis of WSN faults has considerable impact on existing fault handling techniques and tools and is the main reason for the non-adoption of these techniques in real scenarios, as shown in [Rodrigues et al., 2013], which also demonstrates that current WSN diagnostic tools have low maturity, stability and support, and most of them are not available for real world deployments.

In this context, the contributions of this chapter are the following: i) a review of fault management concepts and techniques and their application to wireless sensor networks; ii) a proposal for a comprehensive, consistent WSN/IWSN fault taxonomy that can be used as reference by WSN researchers, developers, integrators and users, as well as by developers and users of tools for fault detection, fault diagnosis and fault handling; iii) validation of the proposed taxonomy using an extensive set of faults described in the literature.

The remainder of this chapter is organized as follows. The motivation for this work is presented in section 3.2. Base concepts and techniques, especially those pertaining to fault tolerance, are presented in section 3.3. The proposed WSN fault taxonomy is presented in 3.4. This section also includes the application of the proposed taxonomy to an extensive set of faults described in the literature. In section 3.5 we compare our work with related work. Finally, section 3.6 sums up this chapter with some conclusions and guidelines for future research.

3.2 Motivation

As the number of WSN deployments increases, researchers and practitioners are now realizing the need for post-deployment diagnostic tools, like the one presented in this thesis. The fragile, complex and failure-prone nature of WSN led to the use of redundancy and other fault tolerance techniques in this type of networks, but this alone is not enough to make them reliable. Failures do happen and, thus, diagnostic tools must be in place in order to identify their cause, nature, and help identifying ways of dealing with them. In this line, the offer of diagnostic tools has been increasing since 2005. However, despite the increasing number of currently available tools, none of them can be considered "ready to use" in real-world WSN applications such as, for instance, the application addressed in [Tran et al., 2015]. With this in mind, section 2.4 presents and compares some of the available WSN diagnostic tools, in order to understand the main limitations of current post-deployment diagnostic tools. This allowed us to clearly identify the main reasons for the low adoption of these tools in real deployments. Actually, the majority of analyzed WSN diagnostic tools have low maturity, stability and support. Most of them were tested in laboratory testbeds only. In general, they do not support security mechanisms (encryption and authentication) and are not portable because they were only designed for a specific operating system.

The analysis of the existing literature, made us realize that, contrary to what happens in order fields, there is a very important piece missing in the area of WSN fault management: a fault taxonomy that can be used to understand and classify the different types of faults in WSN in a comprehensive manner. This is, thus, the main contribution of this chapter.

3.3 Concepts and Techniques

Before proceeding to the presentation of the proposed taxonomy, it is important to define and understand the basic concepts and terminology that it will be used throughout the chapter and thesis. In view of this, this section starts by adapting the concepts of systems, components and services introduced in [Avižienis et al., 2004] to the WSN domain. We then proceed to revise the definitions of fault, error and failure presented in [de Souza et al., 2007; Avižienis et al., 2004; IEEE Std, 1994] and to illustrate their use in WSN through an example. Last but not least, common fault management techniques are put into perspective.

3.3.1 Systems, Components and Services in WSN

Computing systems are formed by *systems* and *components* that deliver *services*, as defined in [Avižienis et al., 2004] (see Figures 3.1 and 3.2). A *system* can be described as an abstract entity that interacts with other entities, such as other systems like hardware, software, humans and physical world. Physically or logically, each system has a *system boundary*, which establishes the border between the system and the external *environment*. The system boundary provides one or more *service access points* that collectively form the *service interface*, through which services are delivered according to a specified system *behavior*.

The system *behavior* is determined by an internal sequence of states, as specified in the *requirements specification*, where functional and non-functional requirements are described. Furthermore, each system generates its own behavior using its *structure*, which determines how its set of *components* are organized and interact with each other (see Figure 3.2). On the other hand, each *component* can be regarded as another system recursively, until a component is considered an *atomic component*. Using this terminology, a WSN can be described as a system with its *components* and *structure* that delivers *services* to other *systems* (see Figure 3.3). The *components* of the WSN system are the sensor nodes, which, in turn, are made up of other systems, recursively, according to a given structure. Specifically, each sensor node *system* has its own *components* (other systems) that can be logical, such as processing, communications and data acquisition, or physical, such as radio, sensors and batteries. All of these components/systems cooperate in a sensor node system in order to deliver sensor nodes services. In a broader perspective, all sensor nodes in a WSN cooperate in order to deliver WSN services to other external systems.

3.3.2 Faults, Errors, Failures and Anomalies

As seen in the previous subsection, systems deliver services to external systems, according to a given behavior. In the service lifecycle, any state in which the system behavior is in accordance with the system specification is called *correct service state*. *Faults* or *errors* may happen while the system is in a correct service state. If they are properly handled, these faults or errors will not have impact

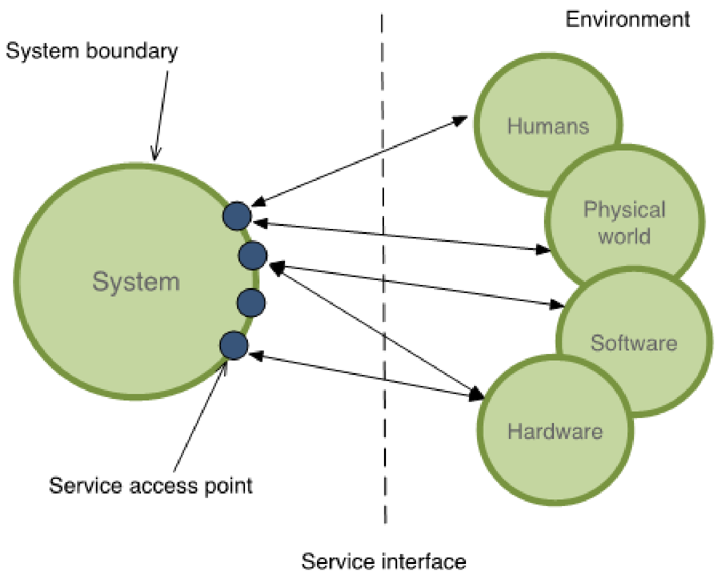


Figure 3.1: Systems, services and environment

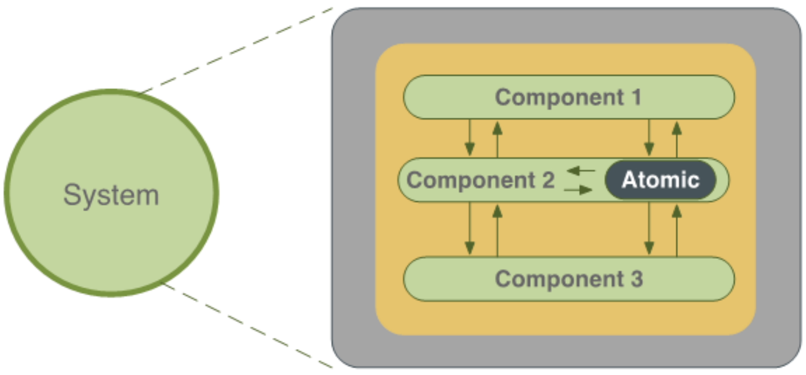


Figure 3.2: System components

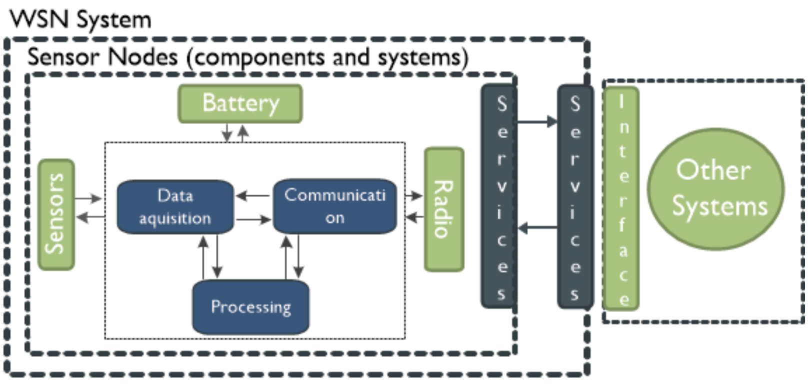


Figure 3.3: WSN system, components and services

on the system state. On the other hand, when some of these faults or errors are not properly handled, the service deviates from the correct service, generating a service failure or simply *failure*, thus delivering an incorrect service. Several definitions for fault, error, failure and anomaly can be found in the literature [de Souza et al., 2007; Avižienis et al., 2004; IEEE Std, 1994; Birolini, 1999]. However, in this thesis we use the definitions presented in [de Souza et al., 2007; Avižienis et al., 2004; IEEE Std, 1994] as they are commonly accepted in the area of dependable systems:

- **Fault**: any kind of defect that can lead to an error, and consequently the cause of an error [de Souza et al., 2007; Avižienis et al., 2004];
- **Error**: an incorrect system state. Such state may lead to a failure if it is not handled, i.e., to a deviation from the correct service [de Souza et al., 2007; Avižienis et al., 2004];
- **Failure**: the deviation of a system from its specification, i.e., deviation of the delivered service from the correct service [de Souza et al., 2007; Avižienis et al., 2004];
- **Anomaly**: any abnormality, irregularity, inconsistency, or variance from expectations (term used with a broad meaning), in other words, the deviation of a system from its normal behavior leading or not to a system

failure.

In order to illustrate the use of the above definitions in the context of WSNs, Figure 3.4 presents an example of a sensor node system *anomaly*. In this scenario, a sensor node periodically sends sensed data to other sensor nodes in the system. However, the battery voltage level of the sensor node is low and its microprocessor powers off several times, performing several reboots. As a result, the sensor node cannot send the data to the other sensor nodes, thus deviating from the correct service.

Analysing this scenario, the *fault* (1) is the low battery voltage. The *error* (2) is the state of the microprocessor, which powers off several times, and the *failure* (3) occurs when the sensor node cannot delivery its services to the WSN. As described in [Warriach et al., 2012], this type of fault may produce other types of faults (4) in the sensor node system, such as a stuck-at-fault fault, where sensor sample readings experience zero or roughly zero difference over a period greater than expected.

3.3.3 Fault Management Techniques

Being a technology used in several types of applications, where some of them are critical, with strict requirements in terms of availability, reliability, safety, integrity and maintainability, WSNs must resort to techniques that overcome or work around their faulty nature because it cannot be assumed that all sources of errors can and/or will be eliminated. In light of this, in latter years several fault management techniques have been proposed, with the aim of preventing the interruption of services delivered by WSNs, thus increasing the availability of WSN systems by decreasing both the number of failures and the Mean Time To Repair (MTTR) . In general, in the context of fault management techniques, fault tolerance techniques prevent the occurrence of failures in WSNs, and recover services when failures cannot be avoided or circumvented, using specific techniques such as hardware replication. In this respect, the following fault management techniques are relevant and will be presented next:

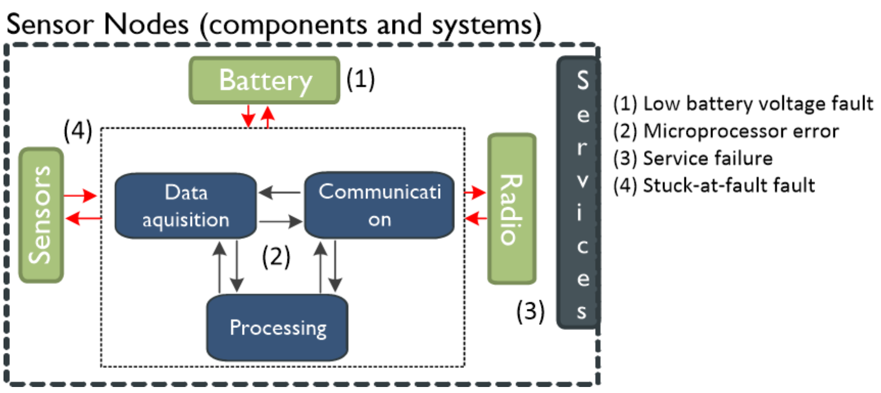


Figure 3.4: Sensor node system anomaly example

1. **Fault prevention**: Usually, the first fault prevention measures in any system are taken at design phase. In this phase, assumptions and requirements are specified and the system is built in order to meet these requirements. In WSNd, fault prevention [Paradis and Han, 2007] is used both at design and deployment phases, e.g., by ensuring adequate network coverage or by setting up the network and configuring network parameters in order to achieve communication redundancy. Additionally, complementary techniques may also be used in fault prevention, such as dependability benchmarking and fault injection [Cinque et al., 2009; Ali and Tixeuil, 2010; Coronato and Testa, 2013; Sailhan et al., 2010; Fairbairn et al., 2013]. In general, in WSN dependability benchmarking, the impact of external environment conditions like time, location, weather or natural hazards are evaluated in order to understand the behavior of WSN under extreme conditions. Concurrently, in order to create these extreme conditions, fault injection is used for generating faults in these scenarios. So, dependability benchmarking and fault injection can be seen as complementary fault prevention techniques;

2. **Fault detection**: After WSN design and deployment, it is crucial to detect faults when they occur during the WSN operational phase [Warriach et al., 2012; de Souza et al., 2007; Koushanfar et al., 2003]. Ideally, fault detection should be performed at all system levels, e.g., sensor nodes, network, and application level. This requires monitoring the system components, and classifying faults in a binary mode as true or false. In order to perform the classification task, several types of parameters (named features in the pattern recognition field) are collected and analyzed (e.g., packet loss, energy, CPU cycles). In [Mahapatro and Khilar, 2013] the authors present the concept of WSN fault diagnosis that corresponds, in fact, to network-wide WSN fault detection. Instead of detecting faults in each sensor node system, faults are detected at network level, requiring each sensor node to have a global view of the network;

3. **Fault isolation and identification**: Subsequently, after detecting the existence of one or more faults, fault identification must be performed [Warriach et al., 2012; Paradis and Han, 2007]. This requires fault isolation [Paradis and Han, 2007], through which correlated faults are identified and several fault hypotheses are proposed. After that, fault hypotheses are tested in order to unambiguously identify the fault type;

4. **Fault recovery**: The final stage of fault management is fault recovery. After the detection and identification of faults, recovery techniques are applied to services, components and systems in order to maintain them working as specified. Some of the recovery techniques may include the replication of vital components, the creation of alternate routing paths, or the adjustment of the sending rate of sensors if congestion in the network is found.

3.4 WSN Fault Taxonomy

This section presents the proposed WSN fault taxonomy. The purpose of this taxonomy is for it to be used as reference by researchers, developers, integrators and users, eliminating ambiguity and, ultimately, enabling the use of a common and coherent perception of WSN faults.

One outstanding paper on basic concepts and taxonomy in dependable and secure computing is presented in [Avižienis et al., 2004] and, in fact, it served as the main basis for the taxonomy presented in the current report. It should be noted that the authors of [Avižienis et al., 2004] explicitly mention that their proposed taxonomy is not closed and should be completed as new technological fields arise. This is the case with the current thesis, that extends this taxonomy to the WSN field.

The taxonomy proposed in the current chapter was developed as the result of an extensive analysis of the literature and state of art on WSN faults, fault tolerance, dependability and secure computing. Through this analysis, we were able to identify a comprehensive set of WSN faults, which were subsequently organized into different viewpoints and types, as presented in Figure 3.5

In this section, in addition to presenting and explaining each of the WSN fault types, we will provide examples of each fault type and their respective references, for further reading. The description of the various faults will be done according to the identified viewpoints (Figure 3.5), namely, phase of creation or occurrence, system boundaries, phenomenological cause, dimension, objective, intent, capability, persistence, state, reproducibility and source system.

3.4.1 Phase of Creation or Occurrence

WSNs go through different stages in their lifecycle and consequently different types of faults can occur in these stages. In the taxonomy proposed in [Avižienis et al., 2004] the authors only consider two phases, namely the development phase and the operational phase. Nevertheless, our experience in WSN deployment [Tran et al., 2015] in industrial environments clearly showed us that the characteristics of WSNs justify the need for two more fault types: requirements phase faults, and deployment phase faults. Thus, from the viewpoint of phase of creation or occurrence, the following four fault types should be considered:

1. **Requirement faults**: WSN design starts with the requirements stage where functional and non-functional requirements are specified. In this stage, some requirements may be ambiguous and specification errors may occur, leading to requirement faults. For instance, routing protocols may have been chosen on the assumption of a specific deployment scenario or specific sensor node mobility patterns. Thus, if some of these assumptions are not met, the used routing protocol may not be able to cope with

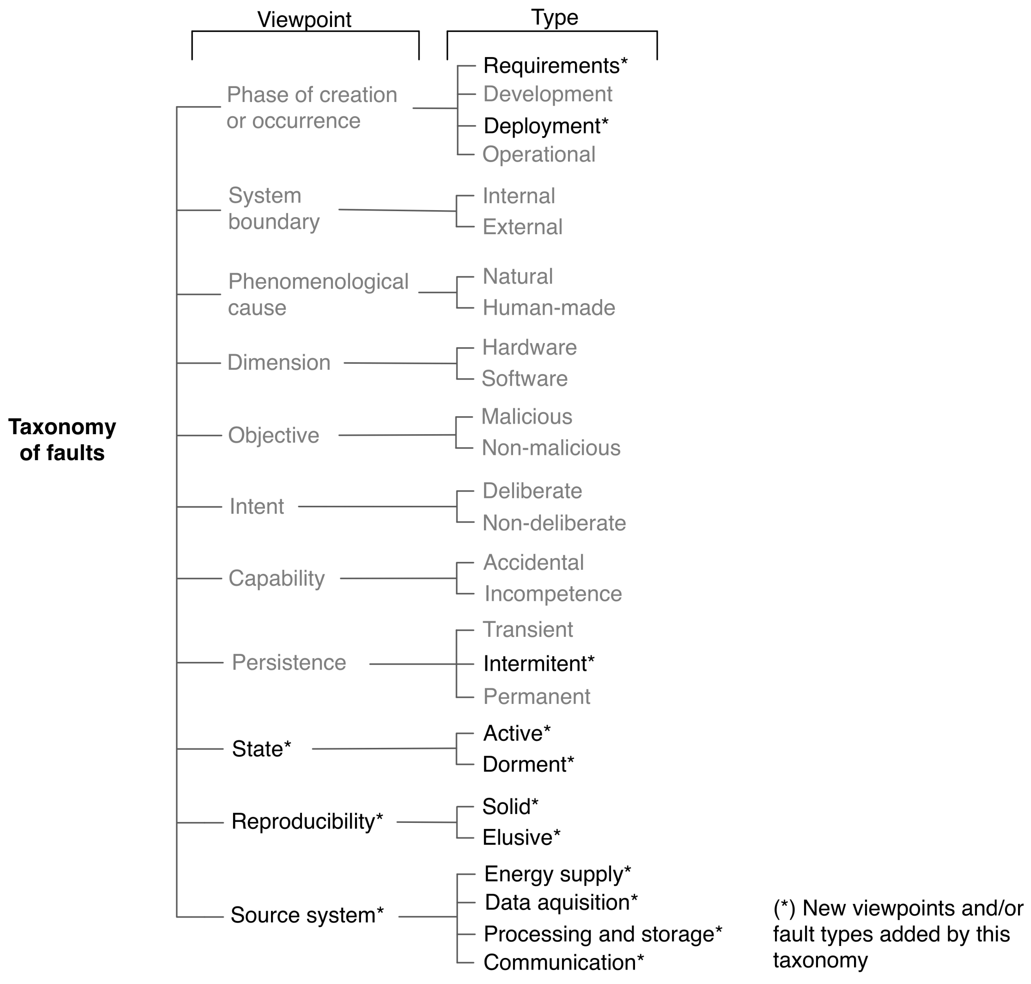


Figure 3.5: Taxonomy of faults in WSNs

network changes in time, leading to dropped messages and other types of performance degradation;

2. **Development faults**: In the development phase, software, firmware, and hardware are produced according to the requirements specified in the previous phase. Nevertheless, some faults may occur due to a variety of reasons. For instance, during software and/or firmware development, vulnerabilities/flaws in security policies might be introduced without awareness. In addition to software and firmware faults, hardware production faults may also be introduced in sensor nodes as, for instance, physical bridging during welding and assembly processes, thus leading to hardware malfunction;

3. **Deployment faults**: After software/firmware development and hardware production, the next phase is the deployment of the network. WSNs can be used in an extremely large variety of application scenarios, and each application can have its specific deployment characteristics with its specific deployment faults potential. Based on our field knowledge, we identified two broad categories of WSN deployments: unplanned deployment,

and planned deployment. In unplanned deployment, sensor nodes are randomly placed and the network configures itself. In this type of deployment, traffic congestion faults and degraded or route failure faults usually occur due to sensor nodes positioning or unforeseen interference. On the other hand, in planned deployment, used for instance in industrial scenarios where sensor nodes need to be placed near specific machinery to monitor and control industrial processes, unforeseen ambient noise faults and channel noise faults are two examples of faults that can happen;

4. **Operational faults**: lastly, after WSN deployment, when the network is operating, faults may also occur. For instance, in this phase an operator may wrongly configure the system parameters, thus leading to failures.

Table 3.1 provides examples and references to all of the mentioned types of phase of creation or occurrence faults.

Table 3.1: Phase of creation or occurrence fault examples

Requirements	Poor design, architectural faults, hardware and software design faults, incorrect algorithms [Ma et al., 2006]; wrong requirements (e.g., a sensor not provisioned for snow) [de Souza et al., 2007]; wrong routing protocol and wrong topology [Ali and Tixeuil, 2010].
Development	Software coding mistakes vulnerabilities/flaws in security mechanisms, manufacturing imperfections, poor component selection, poor construction [Ma et al., 2006].
Deployment	Route misconfiguration during deployment; physical faults due to equipment damage during deployment; wrong topology [Ali and Tixeuil, 2010].
Operational	Battery depletion [Paradis and Han, 2007]; water infiltrations [Sailhan et al., 2010]; wrong configuration or reconfiguration parameters [Avižienis et al., 2004].

3.4.2 System Boundary

As mentioned in section 3.3, each system has its own system boundary, which marks the border between the system and the external environment. In this respect, faults can occur inside the WSN system or in external systems (i.e., in the environment) and propagate into the system by interaction between external systems and the WSN system. Thus, in terms of system boundary, two types of faults should be considered:

1. **Internal faults**: These are faults that originate inside the WSN system, i.e., in one or more of the components that make up the WSN system. For instance, firmware or hardware component faults are internal faults, as their source is a component inside the WSN system;

2. **External faults**: These are faults that originate outside the system or component domain [Ma et al., 2006] and that propagate into the WSN system. Channel noise, radiation, electromagnetic interference, operator mistakes, and environmental extremes are examples of external faults that may produce several types of errors and, consequently, system and component failures.

Table 3.2 provides examples and references to the mentioned types of faults, from the system boundary point of view.

Table 3.2: Fault examples from the system boundary point of view

Internal faults	Bit-flip faults in memory or special registers, logical bridging, physical bridging [Cinque et al., 2009]; memory error fault, registers error fault [Ali and Tixeuil, 2010].
External faults	Battle damages, channel noise, environmental extremes (earthquakes, floods, fire, hurricanes), operator mistakes, radiation and electromagnetic interference [Ma et al., 2006]; overheating [Avižienis et al., 2004].

3.4.3 Phenomenological Cause

Systems are subject to all sorts of phenomenological effects that may cause errors and, subsequently, lead to failures. From a phenomenological cause point of view, faults can be classified into two different types:

1. **Natural faults [Sailhan et al., 2010]**: These are faults caused by natural phenomena that may cause errors in the WSN system. Water infiltration and battery degradation are two examples of phenomena that can originate natural faults. Water infiltration may lead to hardware degradation, and battery degradation may be triggered by an increase in ambient temperature. On other hand, natural faults like power transients cause physical deterioration in sensor nodes hardware;

2. **Human-made faults [Sailhan et al., 2010]**: Humans may also cause faults, either intentionally or unintentionally, in all of the stages of the WSN lifecycle. For instance, a wrong pointer initialization (a fault that can be originated in development phase due to a mistake or bad decision) may result in pointer-initiated memory violation faults and, consequently, may lead to a segmentation fault.

Table 3.3 provides examples and references to the mentioned types of faults, from the phenomenological cause point of view.

Table 3.3: Fault examples from the phenomenological cause point of view

Natural faults	Battery degradation, direct sunlight that swamps infrared signals, short circuit caused by water infiltration resulting in high or low sensor readings [Sailhan et al., 2010].
Human-made faults	Memory corruption, memory leaks and pointer-initiated memory violation [Sailhan et al., 2010]; configuration and reconfiguration faults, errata faults, error in maintenance or operating manuals, faulty human-made tools, logic bombs, unreleased file-locks, and unterminated threads [Avižienis et al., 2004].

3.4.4 Dimension

WSN system components can be looked at from one of two perspectives or dimensions: on one side, they consist of pieces of hardware that physically support their operation and, on the other hand, they comprise software/firmware modules that logically determine their functionality. Either of these perspectives is also applicable when we are dealing with faults and their classification.

1. **Software faults [Koushanfar et al., 2003]**: These are all faults directly or indirectly originated by firmware or software. These include all software faults originated in development process and all software faults resulting from the interaction with other systems. For instance, a trap door is a bypass access control accidentally or intentionally inserted in software (fault) that produces a security flaw (error), and consequently makes the system vulnerable to malicious actions (failure);

2. **Hardware faults [Koushanfar et al., 2003]**: These are faults from electronic and/or mechanical WSN components and systems. In hardware production, an imperfection (fault) created in the welding process of electronic components can lead to intermittent system or component delivery (error), leading to a failure. On the other hand, physical faults are a specific kind of hardware faults that occur in hardware not by design errors but by physical phenomena that damage or compromise the electronic components. For instance, ageing may cause sensors to return inaccurate readings, eventually leading them to stop working.

Table 3.4 provides examples and references to the mentioned types of faults.

3.4.5 Objective

As seen in section 3.4.3, several faults are human-made. These can be classified according to the objective with which they were caused as malicious or non-malicious.

Table 3.4: Software and hardware dimension fault examples

Software faults	Authenticated byzantine faults, byzantine faults, fail-stop fault, timing fault [Mahapatro and Khilar, 2013]; deadlock fault, live lock fault and stack overflow [Rodrigues et al., 2013]; asynchronous fault, registry fault [Ali and Tixeuil, 2010]; missing "AND EXPR" in expression used as branch condition, missing "If (cond) statement(s)", missing small and localized part of the algorithm, missing variable initialization/assignment, wrong arithmetic expression used in parameter of function call, wrong logical expression used as branch condition, wrong value assigned to a variable, wrong variable used in parameter of function call; software aging.[Koushanfar et al., 2003].
Hardware faults	Deterioration of sensor (aging), frozen sensor, sensor unreported value [Warriach et al., 2012]; manufacturing imperfections [Ma et al., 2006]; flipping bits in code/data memory and flipping bits in processor register [Sailhan et al., 2010]. Physical faults: environmental influence fault [Ali and Tixeuil, 2010], radiation and electromagnetic interference fault [Ma et al., 2006], atmospheric perturbation fault and electromagnetic interference [Mahapatro and Khilar, 2013].

1. **Malicious faults [Fairbairn et al., 2013]**: These are faults whose objective is to harm the system. A peak in system service (fault) can be deliberately caused in WSN (generally known as a denial of service attack), leading to network congestion (error), and consequently to local resources exhaustion (failure). Moreover, security breaches [Ma et al., 2006], when introduced with intention, can be seen as malicious faults generated with the aim of compromising the integrity of the network, or of exposing the network to malicious attacks;

2. **Non-malicious faults [Fairbairn et al., 2013]**: These are human-made faults generated in an unintentional way. For instance, an operator can accidentally misconfigure a system or component leading to a network error and, consequently, to a failure. This type of faults also includes faults that result from omission of an action that should have been performed, and also faults where humans unintentionally perform wrong acts [Avižienis et al., 2004], as will be seen in the next sub-section.

Table 3.5 provides examples and references to the mentioned types of faults, from the objective point of view.

Table 3.5: Fault examples from the objective point of view

Malicious faults	Peak in service, physical damage (corrosion, strokes, fires) [Ali and Tixeuil, 2010]; Trojan horses, viruses, worms, zombies [Avižienis et al., 2004].
Non-malicious faults	External or object interference; errors in maintenance or in operating manual[Avižienis et al., 2004]; poor component selection[Ma et al., 2006]; omissions; wrong acts.

3.4.6 Intent

From the perspective of the intent with which they were caused, faults can be subdivided into deliberate and non-deliberate faults, as described below.

1. **Deliberate faults [Avižienis et al., 2004]**: these are faults caused by bad decisions that may or may not have a malicious objective. For instance, a defective or wrongly planned network installation is a deliberate decision that can lead to low radio coverage faults and other environmental faults;

2. **Non-deliberate faults [Avižienis et al., 2004]**: these pertain to all faults that derive from mistakes, i.e., actions for which the causing human (developer, operator, or other) is not aware of. For instance, a missing "AND" expression used as branch condition is a non-deliberate non-malicious fault if unintentionally caused by a developer.

Table 3.6 provides examples and references to the mentioned types of faults, from the intent point of view.

Table 3.6: Fault examples from the intent point of view

Deliberate faults	All malicious faults introduced in table 5; wrong logical expression used as branch condition, wrong value assigned to a variable [Koushanfar et al., 2003]; use of off-the-shelf components with unknown faults and bugs[Avižienis et al., 2004].
Non-deliberate faults	Missing "AND" in expression used as branch condition, missing "If (cond) statement(s)", missing small and localized part of the algorithm, missing variable initialization/assignment [Koushanfar et al., 2003]; software coding mistakes [Ma et al., 2006].

3.4.7 Capability

Until now we identified two important viewpoints that characterize human faults by objective and intent. Furthermore, we also highlighted that humans may cause faults in WSN with non-malicious purposes due to bad decisions or mistakes. Another important viewpoint is that of the capability of the person or persons that cause the fault. From this perspective, faults can be classified into accidental faults or incompetence faults [Avižienis et al., 2004], as described below.

1. **Accidental faults [Ali and Tixeuil, 2010]**: these are faults that occur by accident, not because of the lack of capability of the human agent. For instance, replacing the battery of the wrong sensor node because during installation the node was wrongly tagged may disable an entire network region if the node in question is a node route key;
2. **Incompetence faults**: these result from a wrong action performed because of lack of professional expertise. For instance, in the development phase of a WSN, the developer may generate bugs when developing firmware using a technology or programming language he/she is not proficient in.

Table 3.7 provides examples and references to the mentioned types of faults, from the capability point of view.

Table 3.7: Fault examples from the capability point of view

Accidental faults	Ambient noise (when introduced by human action) [Ali and Tixeuil, 2010], software coding mistakes, insertion of a wrong sensor calibration parameter by mistake [Ma et al., 2006].
Incompetence faults	Wrong sensor model [Koushanfar et al., 2003], vulnerabilities/flaws in security policies and mechanisms [Ma et al., 2006]; wrong specification [Avižienis et al., 2004]; wrong arithmetic expression used in parameter of function call [Koushanfar et al., 2003].

3.4.8 Persistence

Another important viewpoint for faults classification is persistence, according to which we can identify three types of faults:

1. **Transient faults [Mahapatro and Khilar, 2013]**: These represent temporary faults caused by a specific event (usually by interaction with an external system) that occurs in very specific conditions and in a short period of time. Very often, if the same conditions appear though on a different instant in time, the fault may not appear. For instance, if a sensor node A is using a service delivered by sensor node B, a failure in service delivered by B may induce a fault in sensor node A. However, if sensor node A tries to access the service delivered by sensor node B at another point in time, the system may deliver the correct service and the fault in sensor node A will not occur;

2. **Intermittent faults [Mahapatro and Khilar, 2013]**: These are a type of temporary faults that occur randomly and repeatedly. This type of fault may occur in logic components (software) or physical components (hardware). Low battery voltage may lead to intermittent hardware faults;
3. **Permanent faults [Mahapatro and Khilar, 2013]**: These are faults that always produce errors. A bug in the main firmware loop of sensor node will always generate an error when it is called.

Table 3.8 provides examples and references to the mentioned types of faults, from the persistence point of view.

Table 3.8: Fault examples from the persistence point of view

Transient faults	Timing fault [Mahapatro and Khilar, 2013]; configuration and reconfiguration faults ; node mobility faults [Avižienis et al., 2004].
Intermittent faults	Atmospheric perturbation fault, authenticated byzantine fault, byzantine fault, electromagnetic interference fault: fail-stop fault; hardware noise fault; incorrect computation fault [Mahapatro and Khilar, 2013], faults caused by electrode sensors used in soil deployments.
Permanent faults	Crash fault [Mahapatro and Khilar, 2013]; physical damage faults (corrosion, strokes, fires) [Ali and Tixeuil, 2010].

3.4.9 State

Some faults occur when certain conditions are met. Others manifest themselves independently of the input conditions. Examples of the former are undetected bugs introduced in specific software/firmware routines during the development phase. If the testing phase is not exhaustive enough, some of these bugs may go undetected and cause faults under rare circumstances only. So, according to their state, faults can be classified into two categories:

1. **Dormant faults [Avižienis et al., 2004]**: These are faults that do not cause errors until some specific conditions are met in the system, e.g., a specific input, or the execution of certain lines of code, or even both. Dormant faults may stay inactive in the system for long periods of time. For instance, physical bridging can cause a fault that does not manifest itself until a specific input or activation pattern occurs;
2. **Active faults [Avižienis et al., 2004]**: Contrary to dormant faults, these faults stay active in the system, causing errors regardless activation input or other conditions. For instance, offset data faults are active faults that remain so until sensor calibration takes place.

Table 3.9 provides further examples and references to the mentioned types of faults, from the perspective of their state.

Table 3.9: Fault examples from the state perspective

Active faults	Direct sunlight that swamps the infrared signal [Sailhan et al., 2010]; offset sensor bias that always changes the sensor reading, stuck at fault (sensor sends the same value over a period of time).
Dormant faults	Errors in maintenance or operating manual, logic bomb [Avižienis et al., 2004]; trapdoors in code; unterminated threads; undetected software bugs.

3.4.10 Reproducibility

When dealing with faults, namely in the development, training and execution of fault handling algorithms used to identify, classify, inject and/or handle faults, the ideal situation is that faults are reproducible. Unfortunately, this is not always the case, making fault handling extremely difficult. In what reproducibility is concerned, there are two types of faults:

1. **Solid faults [Avižienis et al., 2004]**: these are faults whose activation is reproducible, due to their predictable nature. One example of such fault is a pointer-initiated memory violation;
2. **Elusive faults [Avižienis et al., 2004]**: this comprises faults whose activation is not systematically reproducible, such as communication faults originated by signal noise and/or RF interference.

Table 3.10, below, provides further examples and references concerning solid and elusive faults.

Table 3.10: Fault examples from the reproducibility perspective

Solid faults	Calibration faults (gain data fault and offset data fault), low battery fault; deadlock fault, live-lock fault; pointer-initiated memory violation fault; limited communication bandwidth, limited memory, limited power and low battery faults, limited computational capability.
Elusive faults	Ambient noise [Ali and Tixeuil, 2010]; pattern sensitive faults [Avižienis et al., 2004]; signal fluctuation, signal attenuation; RF interferences, multipath fading and multipath interference [Alena et al., 2011].

3.4.11 Source System

As mentioned in section 3.3, WSN systems comprise several components or sub-systems, of which we highlight the energy supply sub-system, the data acquisition sub-system, the processing and storage sub-system, and the communication sub-system. All of them may be sources of faults and should be taken into account for fault characterization. Thus, from the perspective of the source systems, faults can be characterized as:

1. **Energy supply faults [Ma et al., 2006]**: these pertain to faults that originate in the sensor node power source. WSNs can be deployed in all types of indoor or outdoor scenarios, in which they are subject to all kinds of conditions that can affect the performance of the power source, usually batteries or super-capacitors. For instance, abnormally high battery temperature causes battery degradation, leading to battery low voltage which, in turn, may lead to data faults, network faults, etc.;

2. **Data acquisition faults [Warriach et al., 2012]**: these are faults that occur due to biased or faulty sensor readings. Hardware malfunction (fault) may occur in the sensors' data acquisition sub-system, leading to a series of incorrect sample readings (error) as, for instance, when zero difference samples occur over a period of time. If this error is not properly detected and handled, the sensor node data will be useless (failure);

3. **Processing and storage faults**: these may occur in WSN hardware and/or software/firmware, affecting either the quality and consistency of the stored data or the operations that are performed on them. For instance, bit flips in memory or special registers can corrupt the stored data. Another example is an abnormal increase in the processor temperature, which may cause the drifting of the clock and desynchronize the sensor node from the network, originating several network faults, especially when using TDMA based mechanisms;

4. **Communication faults [Sailhan et al., 2010]**: due to the inherently distributed and dynamic nature of WSNs, communication is one of the major sources of faults in sensor networks. Wireless communications are usually subject to considerable interference (e.g., ambient noise, channel noise, multipath fading, RF interference, etc.) that affects the communication between sensor nodes. Moreover, at networking level, sensor nodes may also be subject to several types of faults. Routing faults can be caused by errors in routing algorithms and/or protocols, which can lead packets to be caught in network loops (error) and never to arrive at their destination (failure). Additionally, sensor nodes may also suffer from faults related to message processing. For instance, when sink sensor nodes cannot dispatch messages faster than they receive them, there may be local congestion in the buffers between the various protocol layers and some messages may have to be dropped;

Table 3.11, below, provides examples and references to the mentioned types of faults, from the source system point of view.

Table 3.11: Fault examples from the source system perspective

Energy supply faults	Battery degradation fault [Sailhan et al., 2010]; limited power fault[Ma et al., 2006]; low battery voltage faults [Mahapatro and Khilar, 2013].
Data acquisition faults	Gain data fault, offset data fault [Warriach et al., 2012]; software aggregation bug [de Souza et al., 2007].
Processing and storage faults	Limited memory fault, limited computation capability fault [Ma et al., 2006]; incorrect computation faults[Mahapatro and Khilar, 2013]; memory errors [Ali and Tixeuil, 2010]; memory corruption [Sailhan et al., 2010]; omission computation.
Communication faults	Message, packet and data manipulation modification/corruption/removal and addition fault [Sailhan et al., 2010]; misguided messages fault [de Souza et al., 2007]; corrupted routing maintenance packets fault [Rodrigues et al., 2013]; routing loops fault [Miao et al., 2013]; degraded route path fault [Ma et al., 2006]; bad network installation [Sailhan et al., 2010].

3.5 Related Work

Fault tolerance is a long-established field, in which there are numerous studies and taxonomy proposals. Nevertheless, WSNs have specific characteristics that require extending and adapting existing fault taxonomies to the reality of this type of networks. To the best of our knowledge, this proposal is the first attempt to specify a taxonomy of faults for WSNs.

Concepts and taxonomy in dependable and secure computing are presented in several pieces of work as, for instance, in [Avižienis et al., 2004] and [Jalote, 1994]. The taxonomy presented in [Avižienis et al., 2004] can be seen as a reference, well-accepted taxonomy for dependable and secure computing in general that, although applicable to WSNs, does no cover all of its needs.

Several other papers present surveys related to fault tolerance and diagnosis, although without the purpose of defining a full WSN fault taxonomy. In this category we highlight [Mahapatro and Khilar, 2013; Ma et al., 2006; de Souza et al., 2007]. In [Mahapatro and Khilar, 2013] the authors classify WSNs faults based on duration (permanent, intermittent, transient), underlying cause, and the behavior of the failure component (hard or soft). In this context, they identify several fault types (crash, omission, timing, fail-stop and byzantine). In [Ma et al., 2006] the authors identify several WSN faults and proposed three new types: specification mistakes, implementation mistakes and component mistakes. Lastly, in [de Souza et al., 2007] the authors survey fault tolerance techniques in WSN and address several types of faults (node faults, network faults, sink faults), classifying them into crash or omission, timing, value and arbitrary. Other papers focus on proposing new WSN fault tolerance techniques and, in doing so, describe several fault types addressed by their proposed technique, thus constituting an alternate source of fault classification. In this category we highlight the papers in [Warriach et al., 2012; Cinque et al., 2009; Ali and Tixeuil, 2010; Sailhan et al., 2010; Koushanfar et al., 2003], as they provide relevant contributions to WSN faults classification. In [Warriach et al., 2012] the authors present two types of faults that affect the performance of WSN: system faults and data faults. In [Cinque et al., 2009] the authors address a hardware fault injection tool and, based on this tool, present several examples of fault types, such as transient and permanent faults, arbitrary faults, hardware faults and data faults. In [Ali and Tixeuil, 2010] the authors describe faults using the time perspective (transient, intermittent and permanent) and using a hybrid model. In [Sailhan et al., 2010] the authors also present several examples of WSN faults and fault types, including active faults, dormant faults, data faults, exclusive faults and elusive faults. Lastly, in [Koushanfar et al., 2003] the authors address faults that are directly related to sensors and sensing, such as offset faults, frozen sensor faults, sensor aging faults and sensor function faults.

The analysis of the existing literature on WSN faults clearly shows that related work focuses the attention in specific fault tolerance algorithms and methods, rather than on proposing a comprehensive WSN fault taxonomy. This, in our view, leads to fault and diagnostic tools that are partial and/or too specific, with limited applicability in real world scenarios, as highlighted in our previous work [Rodrigues et al., 2013]. Moreover, most studies address faults based on a single dimension, e.g., the cause/origin dimension, which does not provide enough information for fault classification. By proposing a comprehensive WSN fault taxonomy, the current proposal addresses the shortcomings of existing related work.

3.6 Summary of the Chapter

A WSN fault taxonomy is needed to better understand and classify the various types of faults that can occur in WSN systems and, thus, helping in the development of fault management tools. In this chapter a review of fault management concepts and techniques and their application to wireless sensor networks was presented, as well as a proposal for a comprehensive, consistent WSN fault taxonomy. This taxonomy can be used by WSN researchers, tool developers, integrators and users. While presenting the taxonomy, examples of a large variety of faults were given and references to existing literature were provided. In the scope of this thesis, this chapter contributed to a better identification of fault characteristics, which is of key importance to the following chapters.

Chapter 4

The Proposed Monitoring Architecture

"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."

(Sir William Bragg)

Contents

Current WSN diagnostic tools have several drawbacks, as already pointed in section 2.4. In our opinion, some of these drawbacks are hampering the use of WSNs in industry because nowadays, to the best of our knowledge, there aren't multi-network, standard-compliant monitoring tools that support the IWSN technologies addressed in this proposal (ZigBee, ISA100.11a, WirelessHART, WIA-PA). In order to make this clear, in sections 2.2, 2.3, and 2.4, a review of the state-of-the-art in what concerns the main Industrial IoT standards, network management, and diagnostic tools was made. As a result, some questions arose: How can multiple networks, possibly comprising equipment compliant with different standards, be monitored in an integrated way? How to add management functionality at gateway level? How can firmware and hardware be monitored without extra costs in hardware, firmware, and network?

In what concerns network monitoring, three important aspects were identified in sections 2.2 and 2.4: the set of techniques used by current diagnostic tools to monitor the network; the available metrics, provided by almost all industrial standards, that can be collected and shared at gateway level; and the current management standards. In what concerns hardware and firmware monitoring, several approaches and technologies were also presented, as well as log techniques that can be used to convey hardware and firmware metrics inside the network. Thus, the review made in these sections identified several crucial points and research directions that led us to the proposal of this architecture and related properties. Implementations of the proposed architecture will lead to diagnostic tools that are totally compatible with industrial standards. In this respect, a proof-of-concept implementation will be presented in section 4.6.4, which also discusses evaluation results.

It should be highlighted that the current architectural proposal goes well beyond identifying components and respective abstract relations. More than defining the requirements, interactions, and roles to be performed by each architecture component, this section defines technologies, approaches, and protocols to be used in realistic, practical industrial scenarios. For instance, this section addresses the techniques and technologies for collecting hardware, firmware, and network metrics; defines the services used in each standard for conveying the monitoring information; and identifies the management protocols and technologies for dealing with the monitoring information.

The remainder of this chapter is organized as follows: initially, the proposed monitoring architecture is presented, providing the reader with a global view of its components and their main roles; subsequently, each component is described in detail, by specifying their role and requirements. Moreover, the technologies applicable to each component will be identified.

4.1 Architecture Overview

With the emergence of several IWSN standards and the increase in the number of IWSN deployments, it is crucial to define an architecture able to monitor multiple-network, standard-compliant technologies. The proposed architecture, presented in Figure 4.1, pretends to be flexible, scalable, energy-efficient, low-cost, and multi-standard solution in order to cover hardware, firmware, and network monitoring. In order to ease the adoption by IWSN vendors, OEM producers, and developers, the architecture was designed according to six main guidelines: i) it should support the monitoring of multiple IWSNs; ii) it should support multiple IWSN standards; iii) it should not lead to a significant increase in energy expenditure; iv) the collection of hardware metrics should not increase the cost of manufacturing the devices; v) the acquisition of metrics should not have a significant impact on the main application size and delay, nor should it lead to a large traffic overhead; vi) the network metrics defined by each IWSN standard should be used; and lastly, vii) the collection of sensor-node metrics (hardware/firmware) and network metrics should be independent. The proposed architecture has five base modules: 1) sensor node monitoring agent; 2) gateway monitoring agent; 3) monitoring logger; 4) management agents; 5) and management system.

The sensor node monitoring agent is responsible for the collection of node monitoring data (hardware and firmware metrics), and the sending of these metrics to the network gateway. The latter will forward the metrics to the monitoring logger, where they will be parsed and stored. The monitoring agent will use the most appropriate service in each industrial standard for forwarding the metrics. These metrics are encapsulated in a specific application format. During its operation (either when collecting information or when sending it), the monitoring agent should minimize the impact on the available resources.

Each industrial gateway has a pair of agents (one monitoring agent and one management agent). The gateway monitoring agent is the component that collects the network metrics and the gateway state (globally called management objects) and stores them in a local database (the datastore). These are then accessed in a standardized way by management systems, through the services delivered by the gateway management agent. In order to support the interoperability with management systems, the gateway also stores the representation of the management objects (i.e., IWSN standard metrics, and gateway state).

On the other hand, the handling of the collected sensor node monitoring data is carried out by a monitoring logger component, which parses the log messages and stores them in its datastore. The monitoring logger is a software component with two main sub-components: log parser and management agent. The log parser is the component that receives the log messages from the gateway, parses them, and stores them in the local datastore. Like the gateway, the monitoring logger locally stores a representation of the management objects. This

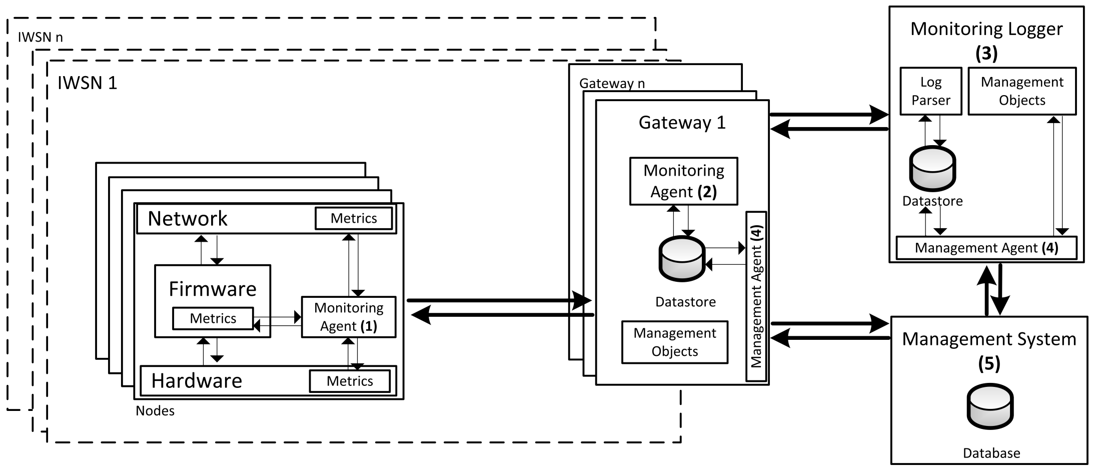


Figure 4.1: Proposed monitoring architecture

representation enables the management agent to share the sensor node metrics with the management system in a standardized way (i.e. by using description languages such as SMIv2, XML and YANG). It should be noted that the monitoring logger is a logical component that can be deployed either on the gateway (if the manufacturer supports it) or on the management system.

The management system receives network monitoring data from the gateway management agent, and sensor node data (hardware and firmware metrics) from the monitoring logger management agent. Besides the traditional functions of configuring the monitoring capabilities of IWSN devices, the management system can include, for instance, a diagnostic tool that alerts operators or developers of critical events in the network, hardware, or firmware. Thus, the management system is capable of monitoring sensor nodes and network behaviour of multiple IWSNs using the standards addressed in this thesis. This chapter outlined a proposal for a monitoring architecture for IWSNs that: does not require any modification to IWSN standardized technologies, benefits from the management information provided by each IWSN standard, and communicates with management systems in a standardized way. This addresses requirements i), ii), vi) and vii), identified in the beginning of this section. The next sub-sections detail each architecture component and show how the identified questions and requirements (especially, the ones more related with implementation strategies) are address by this proposal.

4.2 Sensor Node Monitoring Agent Overview

Sensor node metrics (hardware and firmware) are collected by the sensor node monitoring agent, as opposed to network metrics, which are collect by the gateway monitoring agent. Independence between the acquisition of network and sensor node metrics is one of the main features of the proposed architecture. This makes it possible to monitor the network independently from the acquisition of sensor node metrics. This separation also allows having closed modules in IWSN gateways (provided by IWSN vendors) and a more open monitoring logger module (that can be extended by developers if needed).

This section presents and discusses the main requirements for hardware and firmware metrics acquisition in the sensor node monitoring agent, and provides guidelines as to the technologies that can be used in this scope. Additionally, the approaches used in the transport of the logged data over the IWSN will be also presented.

4.2.1 Hardware Metrics Collection

In order to persuade IWSN vendors and OEM producers to incorporate real-time monitoring in sensor nodes' hardware, the monitoring techniques used by the sensor node monitoring agent should meet the following requirements: the added hardware components should not significantly increase the cost of sensor node manufacturing; the board space required by monitoring components should be minimal; the energy consumed by the hardware monitoring components should have a negligible impact on the battery life; processing and memory overhead should be minimum; and lastly, access to the hardware metrics should be seamless across all hardware platforms.

Regarding the latter requirement, considering the techniques presented in section 2.4, those who enable direct access to microcontroller metrics (by using internal hardware registers and hardware counters) have a higher level of portability when compared to external physical tools. From the sensor node monitoring agent perspective, seamless access to hardware metrics, regardless the underlying technology, is extremely important. By using different Board Support Packages (BSPs) developed by hardware manufactures (e.g., Drivelib [Instruments, 2019]), or by using interfaces available in the sensor nodes operating systems (e.g., Contiki, RIOT), it is possible for sensor node monitoring agents to gather hardware metrics directly from each hardware platform. Hardware monitoring techniques that collect vast quantities of data to infer the hardware conditions are not appropriate for IWSN application scenarios.

The work performed in [Scherer and Horváth, 2012; Scherer and Horvath, 2014; Dutta et al., 2008; Rodrigues et al., 2014] presents a set of metrics that can be collected with little cost and extra value to assess the health of hardware components. Specifically, in [Dutta et al., 2008] the authors present a low-cost technique that, using an extra wire connected to an Microcontroller Unit (MCU) hardware counter, enables the measurement of sensor nodes energy consumption. This technique was used in [Rodrigues et al., 2014], in an anomaly detection system, with good results, and proved its low footprint in terms of memory, cost, and processing load. Using the same MCU technology (the MSP430 family), in [Rodrigues et al., 2014] the authors present a technique that enables data acquisition on microcontroller cycles and execution time. In 16-bit architectures, the MSP430 family is one of the most used microcontrollers in WSN hardware due to its low power consumption. On the other hand, in 32-bit architectures, ARM microcontrollers are the most used ones. For these platforms, the work presented in [Scherer and Horváth, 2012; Scherer and Horvath, 2014] describes a set of metrics that can be collected from the CoreSight technology available in the ARM architecture. The Data Watch Point (DWT) block can provide several measurement statistics, such as interrupt overhead, and sleep cycles. Another block, the Embedded Trace Macrocell (ETM), can provide statistics concerning the number of executed or skipped instructions.

Summing up, the monitoring of hardware condition can be performed at hardware level in compliance with the defined requirements. Furthermore, the proposed methods allow data acquisition for different types of microcontroller architectures and technologies. A set of techniques already addressed and proven by the scientific community were presented, and are recommended for use in the proposed architecture

4.2.2 Firmware Metrics Collection

Development of firmware for sensor nodes can be made using one of two types of approaches: the "bare metal" approach or the OS-based approach. When the "bare metal" approach is used, the application is usually developed in C or C++, and access to the hardware is made by the BSP supplied by the manufacturer, resulting in a very hardware-dependent development. This type of development is less portable than OS-based development, in which hardware operations are managed by the OS. Developers only have to call OS generic functions that will, in turn, be translated into hardware-specific calls.

Firmware monitoring techniques used by the sensor node monitoring agent should support the monitoring of applications developed either using bare metal or OS-based approaches. Thus, as main requirements, firmware monitoring techniques should: be application independent; support, at least, C and C++ (according to [Hahm et al., 2016] most OSs use these languages); be OS-independent; when enabled, have a negligible impact on the execution of the main application (i.e. a minimal increase in the microcontroller's load); minimize the use of RAM, program memory, and external memory; be easy to integrate into the developers work flow; and not depend on physical access to the hardware. Similar to hardware monitoring techniques, firmware techniques that collect vast quantities of data and need physical access are not appropriate to this architecture.

From the set of techniques used in the tools presented in section 2.4, instrumentation techniques are the only ones that fulfil the mentioned requirements. Tools that use instrumentation techniques with scripting languages allow developers to easily integrate them in their daily workflow and, at the same time, provide code tracing, debugging, profiling, and performance counters. Supporting instrumentation in C and C++ languages allows the use of instrumentation techniques in almost all OSs. In the proposed architecture, the sensor node monitoring tool collects firmware metrics by using instrumentation techniques.

In [Dong et al., 2014; Schuster et al., 2014; Dong et al., 2013; Shea et al., 2009] the authors present some technologies, techniques, and metrics that use code instrumentation. For instance, in [Shea et al., 2009] the authors use the C Intermediate Language (CIL) to instrument the C source code. The instrumentation code is inserted by a parser before building the binary file. Using a similar approach, in [Schuster et al., 2014] the authors use the PYCParser tool to instrument the code to support logging. On the other hand, in [Dong et al., 2014; Bhadriraju et al., 2012] the authors use binary instrumentation to inject the monitoring code using the trampoline technique. Differently from the other instrumentation techniques, in binary instrumentation, developers do not need to have access to the source code to change the program flow. Using trampoline techniques, the displaced instructions are executed and then another jump is made back to the actual code. This method allows the use of instrumentation in running code. These techniques allow the collection of several firmware state and identification metrics like: function header, control flow (if, else, switch), function footer, function calls, variables values, and number of variable assignments.

In conclusion, from all the techniques presented in section 2.4, solutions that need physical access to hardware interfaces (e.g. JTAG) to collect firmware monitoring metrics cannot be used in real industrial deployments. For this reason, the sensor node monitoring agent proposed in the current architecture collects firmware metrics using techniques deployed in the firmware by code instrumentation.

4.2.3 Transport of Collected Data

IWSN standards were engineered to optimize four important sensor node resources: node battery, reliability, latency and network bandwidth. In this type of networks, sensor nodes should operate during several years, and network resources should be optimized for conveying the collected data with a variety of QoS requirements. Typically, the transport of sensor node monitoring information has lower priority than the main application traffic because the collection of this information cannot compromise the sensor node main application (nor its operating lifetime). Thus, the transport of sensor node monitoring data made by the sensor node monitoring agent should fulfil the following requirements: the monitoring information should be sent using appropriate network services and without compromising the main sensor application; the monitoring information storage should have a minor impact on the sensor node resources (for instance using a First In First Out (FIFO) buffer in RAM, and dropping data when the buffer is full); when available, data should have a network timestamp to make it possible to correlate several events in the network; when security is enabled for the main application, the monitoring data should also be secured; when log packets exceed the maximum payload size and no fragmentation and reassembly mechanisms are available in the underlying layers, the sensor node monitoring agent should support fragmentation and reassembly mechanisms; lastly, the impact of such operations on battery life should be minimal.

Regarding the timestamp requirement, from the set of the four analysed standards ZigBee is the only one that does not define a time standard in its specification. On the other hand, WirelessHART, ISA100.11a, and WIA-PA define their own time standard as Universal Time Coordinated (UTC), International Atomic Time (TAI), and UTC, respectively, and provide services to synchronize sensor nodes [Wang and Jiang, 2016]. Consequently, in ZigBee, sensor node monitoring data cannot be correlated without implementing additional mechanisms in the sensor nodes that enable to synchronize the network time between nodes.

As already mentioned, the Maximum Transmission Unit (MTU) in IEEE802.15.4 is 127 bytes. Thus, all of the four standards use fragmentation and reassembly mechanisms to overcome this limitation. WIA-PA and ISA100.11a implement this mechanism at the network layer, unlike ZigBee that implements it at the application support sub-layer. WirelessHART also supports the transport of large packets but, in this case, sensor nodes need to allocate a special type of transport service - the block data transfer. This service only allows the existence of a unique block transfer service for the whole network [Wang and Jiang, 2016; Alliance, 2015]. Thus, in WirelessHART, the transport of blocks of data that exceed the maximum MTU size requires fragmentation and reassembly to be implemented at application level, if the block data transfer service is not used.

Another important aspect is the security of sensor node monitoring data. All of the industrial standards within the scope of this architecture offer end-to-end encryption. In the cases of ZigBee, WirelessHART, and ISA100.11a, security is managed using a centralized approach. On the other hand, WIA-PA uses a distributed approach, where the security manager together with the routing devices configure the security measures and forward the keys to field devices. All the standards under consideration provide encryption using symmetric keys, although ISA100.11a can also use asymmetric encryption in device joining processes. Last but not least, the highest level of IEEE 802.15.4 security policy is AES-128-CCM. Consequently, most IEEE802.15.4 radio chips support AES-128-CCM operations at hardware level [Wang and Jiang, 2016].

In order to minimize the impact on the main application traffic, the approach to transporting sensor nodes monitoring data should be carefully selected for each standard, as presented below.

In WirelessHART, data transport can be done using one out of four types of services: block transfer service, maintenance service, periodic service, and event service. The block transfer service can only be used by one sensor node at a time. Thus, this service is not appropriate to transport sensor node monitoring data. The maintenance service provides the network with a minimal bandwidth for basic sensor nodes control and management operations. Because this is a low bandwidth service, it cannot handle the transport of sensor node monitoring data. The event service is used by applications to send data packets during unexpected events, such as alarms and warnings (when this service is requested, the radio needs to define the latency associated to the service)[Technology, 2017]. Finally, the publish service is used to periodically send data, like sensor readings (when this service is requested, the radio needs to define the interval for sending the data)[Zand et al., 2012b]. Taking into consideration the characteristics of each service, we conclude that the most appropriate service to transport the data is the publish service, since the event service is a service used by applications with latency constraints, and the monitoring data does not have such requirement.

ZigBee does not specify services for data transport, leaving this to the application layer. According to the ZigBee specification [Alliance, 2015], in order to guarantee compatibility among different manufactures, ZigBee devices need to implement application profiles (also called endpoints). An endpoint is a ZigBee application for a specific domain with a set of clusters (or application messages) that can be mandatory or not. In turn, clusters are composed by attributes that represent the exchanged data. ZigBee sensor nodes that implement the same ZigBee profile (endpoint) are able to communicate with each other. In the context of this architecture, the sensor node monitoring agent is an endpoint and a set of cluster messages that can be sent to the ZigBee coordinator (that must also support the same endpoint).

Compared to the other standards, ISA100.11a is by far the most complex and customizable standard. Instead of using the term services, used in WirelessHART, in ISA100.11a contracts establish the resources allocated by the system manager to devices. Before a device can send data to another device, a contract needs to be created. Contracts are identified by an ID, unique within the scope of the device (but not necessarily so in the scope of the network), and are unidirectional. The system manager is the device that has the authority to assign, modify and revoke contracts. Like WirelessHART services, there are several types of contracts and attributes that can be used for establishing service levels. Firstly, contracts may be of two types: periodic, that schedule network resources for the periodic sending of data; or, otherwise, non-periodic. Secondly, contracts can also be negotiable. The system manager may change or revoke the contract to make resources available to other high priority contracts. Lastly, contracts can have several levels of priorities: best effort queued, used in client-server communications; real time sequential, used in voice and video applications; real time buffer, used for periodic communications; and network control, used for managing network devices by the system manager. Message retransmission can, additionally, be enabled or disabled inside the contract. Thus, sensor node monitoring agents that run inside an ISA100.11a network should request

Table 4.1: Network transport revision

Standard	**Time Standard**	**Fragmentation and Reassembly**	**Security**	**Service Level Agreement (available options) Support**	**Service Level Agreement (recommended) Protocols**
ZigBee	-	Application Sub-Layer	Symmetric encryption available in all standards and supported by IEEE802.15.4 hardware	-	-
WirelessHART	UTC	Block transfer service		Publish, Event, Maintenance, and Block Transfer service	Publish service
ISA100.11a	TAI	Network Layer		Contract type: periodic and non-periodic Contract priority: best-effort queued, real-time sequential, real-time buffer and network control	Contract type: Non-periodic Contract priority: Best-effort
WIA-PA	UTC	Network Layer		Publish/subscribe VCRs, source/sink VCRs, and client/server VCRs	Source/sink VCR or Client/server VCR

a non-periodic contract type, that can be negotiated and revoked if needed. In this way, the system manager can revoke the contract of the sensor node monitoring agent, thus guaranteeing that the main sensor application can deliver sensor data without disruption. Additionally, the contract priority used by the sensor node monitoring agent should have the best effort queued type.

In the case of WIA-PA, networks may operate in two distinct modes: a hierarchical network topology that combines star and mesh, or a star-only topology. The star-only topology is a special case of the hierarchical network. For this reason, this topology uses the same services available in the hierarchical topology for data transport. In WIA-PA, the Virtual Communication Relationship (VCR) is the main standard block to access the objects specified in the User Application Objects (UAOs). VCRs distinguish the routing and communication resources allocated to each UAO. Each VCR has a VCR identifier, a source UAO ID, a destination UAO ID, address of source device/destination device and the VCR type. Similar to WirelessHART services and ISA100.11a contracts, in WIA-PA, VCRs can be classified according to the application scope: publish/subscriber (P/S) VCRs, used for publishing periodic data; report source/sink (R/S) VCRs, used for transferring aperiodic events and trend reports (alarms); and client/server (C/S) VCRs, used for transferring aperiodic and dynamic paired unicast messages (for getting and setting operations in UAO). Additionally, VCRs also provide aggregation methods. In this architecture, the sensor node monitoring agent data are UAOs capable of representing log messages. Thus, sensor node monitoring agents, available in each field device, as well as routing devices, need to select the appropriate VCR. P/S VCRs are appropriate for real-time operations, using exclusive timeslots in intra and inter-cluster communication. For this reason, this type of VCR should not be used for sending the monitoring data. On the other hand, R/S and C/S VCRs use the CAP period of the IEEE802.15.4 slot inside clusters, and use shared timeslots in inter-cluster operations. Thus, C/S and R/S VCRs are the most appropriate for transporting sensor node monitoring data in WIA-PA networks.

4.3 Gateway Monitoring Agent

The gateway monitoring agent is the component in charge of representing management objects data in a standardised way, and of sending network monitoring data to the management systems. This agent also deals with data from multiple standards. The gateway monitoring agent is independent from the sensor node monitoring agent, thus allowing the separation of sensor node data collection from network monitoring data coleection. In this context, gateway monitoring agents must meet the follow requirements: network monitoring should not significantly increase the cost of equipment nor the cost of installation; collecting network metrics should add low overhead to the gateway; monitoring should be energy-efficient; the monitoring solution should be easy to install and should be extensible; and, lastly, gateway monitoring agents should support widely used industrial standards, namely the ones addressed in this thesis.

Considering the three types of techniques identified in section 2.4.1 (active, passive, and hybrid), active type techniques are the only ones that are easy to install and do not rely on extra hardware to perform monitoring tasks. However, this type of techniques usually consumes sensor node resources, as pointed in section 2.4. On the other hand, in section 2.2.3, the revision made to the metrics shared between nodes showed us that the standards under consideration can support the sharing of critical information related to network and sensor node state, which can be used for resource allocation and routing tasks. Thus, by using these metrics, active tools will be capable of fulfilling the identified requirements without spending significant sensor node resources. Furthermore, some current industrial solutions available on the market already provide these metrics at the gateway, using proprietary libraries. The only disadvantage of active type techniques is the partial coverage in ZigBee networks because some metrics cannot be gathered by the coordinator. This arises from the fact that, in ZigBee, route computation is done in a distributed fashion for some topologies.

4.4 Monitoring Logger

As shown in table 2.2, the standards under consideration use different application approaches. Specifically, ZigBee, ISA100.11a, and WIA-PA use object-oriented representation, whereas WirelessHART uses a command-oriented representation. Sensor node monitoring data needs to comply with the representation defined in each standard, which leads to different representations for the same sensor node data. To solve this issue, the monitoring logger is the component that parses distinct standard representations into the same representation.

The monitoring logger should fulfil the following requirements: support the connection to the different standards "gateways"; support the parsing of the distinct application protocols using a sub-component (the log parser); support the representation of sensor node monitoring data in the languages supported by the management protocols; allow access to the monitoring data by management systems, using the management agent; and, lastly, store the sensor node monitoring data in a datastore. As a software component, the monitoring logger can be installed as an extra software module in industrial gateways or in another compatible equipment.

4.5 Management Agents and Management System

Management systems are one of the building blocks of current IP-based networks. By monitoring the networks and their equipment, operators and manufactures can maximize the network uptime, improving the delivered quality of service. When applied to IWSNs, network and sensor node monitoring can also provide similar benefits. It is unthinkable to have a large IWSN that uses one or more standards without a central management system to perform predictive maintenance of sensor nodes. As presented in the previous sections, hardware, firmware and network monitoring data can be delivered by the sensor node monitoring agent and by the gateway monitoring agent, respectively. However, a key to the puzzle of this architecture is missing. To improve interoperability between these agents, different manufactures, and diagnostic tools, appropriate management protocols and syntax languages are needed. In this context, management agents should fulfill the following requirements: agents should allow the representation of sensor node and network data as managed objects; when needed, managed objects should be able to be extended, supporting additional monitoring metrics; management agents should be able to share management object information with management systems, updating the model used in management systems when needed; the management protocol should be able to support security mechanisms; management agents should run in IWSN gateways, for which the minimum hardware requirements should be set to IETF Class 2 [Bormann et al., 2014] (e.g., gateways that can support, for instance, a simplified version of Linux-based operating system); management protocols should support notification messages (query-based architectures can be heavy for IWSN gateways and are not scalable); lastly, the management agent must operate in IP networks (IWSN gateways must be able to be connected to an IP network).

When network management technologies were previously analyzed in section 2.3, some protocols, syntaxes, and key features were identified. Current network management protocols can be divided into protocols designed for resource-constrained devices, and protocols for traditional networks. While traditional network management protocols are more mature and widespread, resource-constrained management protocols use more advanced, simpler, and modern technologies for representing management objects (e.g., YANG), and for data transport (e.g., CoAP, HTTP). Here, we highlight that protocols designed for resource-constrained devices can also be used in more powerful devices, like

IWSN gateways. However, there are other aspects that need to be addressed regarding the requirements defined in this section. From the set of syntaxes for representing management models, XML and YANG are the most flexible and most simple languages. The learning curve to describe management objects using SMIv2 is longer than using languages like YANG and XML. Thus, SMIv2 is not recommended in the scope of this architecture. Consequently, SNMP is also not recommended for this architecture. Moreover, when working with data models, one important requirement is the sharing of the model with management systems. From the remaining protocols proposed in the scope of this architecture, NETCONF [Enns et al., 2011], COMI[der Stok et al., 2017], RESTCONF[Bierman et al., 2017], and LwM2M[OMA, 2017] allow the discovery of the models and of the resources available in each management node, by using discovery commands or specific URIs that enable to retrieve the used models. Additionally, other important requirement is the use of notification messages. By creating subscriptions to different management topics, diagnostic tools will be able to have the monitoring information in real time and without extra effort, contrary to what happens in query-based protocols that need to actively look for the monitoring information. Considering the set of analysed management protocols, the only one that does not support notifications is the NETCONF protocol. Before a client and a server can exchange management messages, NETCONF has to establish a permanent connection using SSH and TCP. Thus, it is impossible to notify management clients in the case of a connection loss. RESTCONF, LWM2M, and COMI have support for notification messages.

Lastly, one of the main characteristics of protocols for resource-constrained devices is the capability to use the protocols in the node itself (something they were developed for). However, in IWSNs, for which application protocols are statically specified by the standard, it is not possible to use other protocols in sensor nodes, such as the management protocols we have been addressing. The only option is to represent managed objects in the gateway of each standard. The authors of [Chang and Lin, 2016] present a solution to monitor and manage legacy systems with LWM2M protocol, by using several LWM2M clients to represent each legacy device behind the gateway. Additionally, as presented in [Raposo, 2017], using the YANG language it is possible to represent the network metrics shared with a gateway in a WirelessHART network. In this way, COMI and RESTCONF can also be used in the gateway to represent legacy devices.

4.6 Building a Proof-of-Concept

In the previous section, the monitoring architecture was presented as well as its components, requirements, and available solutions for each component in light of the existing standards and protocols. Being a general monitoring architecture that integrates several network standards, hardware technologies, and firmware

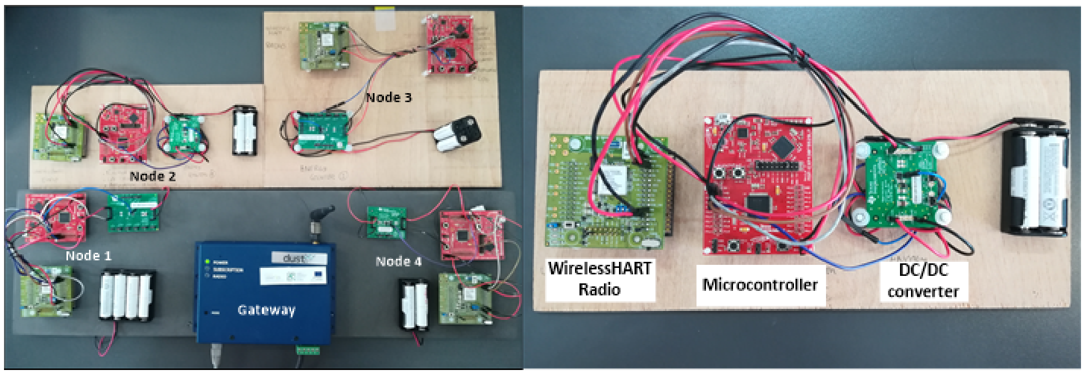


Figure 4.2: Validation of the monitoring architecture using a WirelessHART testbed

architectures, a proof-of-concept implementation of such architecture showing each supporting technology and monitoring technique is unfeasible. Thus, in this chapter, a proof-of-concept scenario is presented using a small testbed with a WirelessHART network. Using some of the technologies and solutions already presented in the sections 2.3 and 2.4, we set out to prove that the proposed architecture is able to monitor the network and the sensor nodes (hardware and firmware) operations, with low impact on IWSN resources.

This section starts by presenting the test scenario, comprising the industrial application, and the deployed hardware and firmware components. Secondly, the collected metrics and associated mechanisms are presented. Thirdly, data acquisition and processing of sensor node metrics are explained. Lastly, the impact on the network and sensor node resources is analysed and discussed.

4.6.1 Test Scenario

The proposed monitoring architecture was tested using a typical industrial application. The proof-of-concept scenario consisted of four sensor nodes and a WirelessHART gateway. On one hand, sensor nodes monitor the temperature of the industrial assets and send their readings to the gateway. On the other hand, the gateway controls all the traffic in the network, performing the role of a network and security manager (Figure 4.2, on the left). According to the proposed monitoring architecture, WirelessHART sensor nodes and the gateway are capable of sharing sensor node monitoring data and network metrics.

The firmware deployed in the sensor nodes can be divided into two distinct modules: the industrial application, and the sensor node monitoring agent. The industrial application is responsible for temperature sampling and for controlling the network operations (by sending control messages to the radio, such as network joining, network notifications, and service requests). On the other hand, the sensor node monitoring agent collects hardware and firmware metrics and sends them by using a specific network service.

As presented in the sensor node monitoring agent description, section 4.2.3, WirelessHART defines four types of service level agreements: block transfer service, maintenance service, periodic service and event service. The proof-of-concept application running at the microcontroller requests the following services to the gateway: a publish service, for sending temperature data every minute; an event service, for sending alarms if the temperature rises above a certain threshold (the verification is made every 30 seconds); an additional publish service that is requested and deleted each time the sensor node needs to send monitoring data (every 15 minutes); and, lastly, a maintenance service, directly allocated by the radio that handles all the control messages between the radio and the network manager.

In terms of hardware, the sensor nodes are formed by three components (Figure 4.2, on the right): radio, microcontroller and power supply. The radio (Linear DC9003A-C) handles the communication tasks with other network nodes and with the microcontroller. The microcontroller (Texas MSP430F5 launchpad) runs the industrial application and the sensor node monitoring agent. At the same time, the microontroller is connected to the WirelessHART radio over UART, and to temperature sensor (ADT7301) over SPI. Lastly, the power supply is formed by a pack of batteries and a DC/DC converter used to supply energy to the radio and the microcontroller. Two of the nodes use TPS62740EVM-186 buck converters, and the other two use TPS61291EVM-569 boost converters. In addition to assuring a predefined voltage level, by using these DC/DC converters, we can measure the energy expenditure of each sensor node almost for free and in real time.

The gateway (LTP5903CEN-WHR) controls all network operations, manages the network, performs security operations and, at the same time, connects the IWSN with the industrial network through an IP network. Additionally, an application running at the gateway allows the subscription of specific types of messages (e.g., application messages, sensor node monitoring data messages, and network report messages). This application performs the role of the gateway monitoring agent, collecting the reports identified in table 2.3.

Finally, the monitoring logger was developed as a python application that implements the log parser sub-component. When the monitoring logger application starts, a specific subscription is made to the network manager in order for it to receive the sensor node monitoring messages. After that, when the data arrives at the sub-component, the log parser converts the packets to JSON objects and stores them in a datastore.

In this proof-of-concept scenario, we only intend to measure the impact of the architecture on sensor node resources (memory, processing, energy) and on the network operation (overhead in relation to the sensor node traffic). Thus, we did not implement the management agents presented in the architecture (at the gateway and at the monitoring logger). Such agents were already presented and are freely available, as described in [Špírek, 2017; Eclipse, 2017], and, thus, their implementation in this proof-of-concept scenario would not provide relevant added-value. However, we should emphasize that the WirelessHART network metrics management models that would be required for such implementation were already created by us and are available for download in [Raposo, 2017].

4.6.2 Collected Metrics

The architecture proposed in this thesis allows the collection of hardware, firmware, and network metrics using agents deployed in sensor nodes and network gateways. In this chapter, some of the requirements were presented, as well as some monitoring techniques already explored by other researchers. Thus, to prove that this architecture has low impact on sensor nodes resources, our test scenario implements some of these techniques, enabling the collection of metrics from the hardware of the Texas MSP430F5 launchpad, from the industrial application, and from the WirelessHART network.

Collecting hardware metrics was done using techniques that directly access the registers and counters of the Texas MSP430F5 launchpad. As this is a 16-bit microcontroller, the implemented techniques were the ones presented in [Dutta et al., 2008; Rodrigues et al., 2014], comprising a processing metric, an energy metric, and a time metric. The processing metric gives the number of instructions executed by the microcontroller. Using switching regulators in combination with the technique presented in [Dutta et al., 2008], it was possible to have an energy metric that provides the energy spent by sensor nodes. Lastly, the time metric gives a time measurement in milliseconds.

Specifically, the metrics were implemented in the following way. Firstly, there are two possible approaches to implement the processing metric in the Texas MSP430F5529 launchpad: 1) by configuring pin P7.7 to output the MCLK clock and connecting it to a hardware counter; or 2) by using the same clock source and same frequency of the MCLK clock in SMCLK, and configuring a counter to count it. As the Texas MSP430F5 launchpad does not give us access to pin P7.7, the second approach was used, and TimerA2 was configured to be sourced by the SMCLK. Secondly, the energy metric was obtained by connecting the output of the inductor used in the switching regulator (used as a DC/DC converter, either the TPS62740EVM-186 or the TPS61291EVM-569) to the TimerA1 clock signal input (P1.6) of the microcontroller. Lastly, the time metric was obtained using the TimerB, configured to be sourced by the Auxiliary Clock (ACLK) (being sourced by ACLK allows to count time even during sleep periods such as Low Power Mode 3 (LPM-3)).

Apart from hardware metrics, the architecture also enables firmware monitoring by using instrumentation techniques applied to the main application code. In order to prove that it is possible to collect some metrics with low impact on the sensor node resources, manual instrumentation of the code was performed,

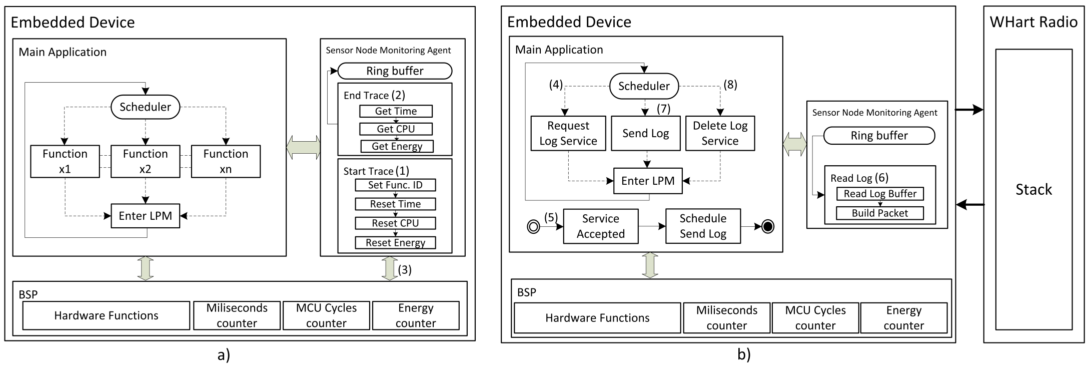


Figure 4.3: On the left, (a) the application architecture and the sensor node monitoring agent acquiring the state information. On the right, (b) the request of the WirelessHART publish service

with the objective of obtaining a trace of function calls. Specifically, the function calls trace was obtained by assigning a specific ID to each function in the code. Additionally, the instrumentation code enabled each executed function to collect the function identifier, the function duration, the spent energy, and the processing load (by using the metrics collected from the hardware). The details of the data processing and transport over the network will be presented in the following sub-section. Lastly, after collecting the metrics concerning the sensor node state (hardware and firmware), the gateway monitoring agent collects the reports identified in table 2.3, namely device health report, neighbour health list report, neighbour signal levels, and network alarms.

4.6.3 Sensor Node Instrumentation and Monitoring Information Processing

The proof-of-concept implementation presented here aims at demonstrating the applicability of the proposed architecture to several software development scenarios. Specifically, the architecture should support OS-based applications as well as “bare metal” applications, as in the current case. With this in mind, the application was developed based on a function-queue-scheduling architecture. Similar to what happens in OS-based approaches, the functions are added to a queue of function pointers, and called when appropriate. The scheduler is responsible for getting the next function in the queue, which will then be executed. On completion, the microcontroller is put in low power mode (LPM) to save energy.

The main topic of this section is, nevertheless, the sensor node monitoring agent and its interactions with the BSP, the main application, and the network (figure 4.3). The sensor node monitoring agent is a library developed in C/C++ that collects the sensor node state data, stores it, and triggers a periodic event to send the monitoring data over the network. By manually inserting a simple call to the sensor node monitoring agent at the beginning of each function to be monitored (see figure 4.3(a)(1)), it is possible to record the state of the sensor node's firmware and hardware. The monitoring information is then stored in a ring buffer (figure 4.3(a)(2)), until it is sent over the network. In addition, the hardware state data is collected directly by the sensor node monitoring agent library from the BSP (figure 4.3(a)(3)).

Last but not least, as illustrated in the figure 4.3 (b), when there is sufficient data to fill an IEEE 802.15.4 data packet payload, the sensor node monitoring agent requests a publish service to send the data available in the ring buffer (4). If the network has enough resources to send the data at the requested rate, an authorization is received (5), and the sending process starts (6-7). In case the network does not have enough resources, the microcontroller will receive a service denial, and the sending of the log will be postponed. When no more data is available in the ring buffer, the sensor node monitoring agent requests the deletion of the service in order to free the network resources (8).

4.6.4 Results

In order to show that the proposed architecture can be implemented in industrial scenarios with efficiency and low impact on resource consumption, some experiments were conducted, whose results are described in this sub-section.

The performed tests can be divided in two groups: tests that measure the impact on the sensor node resources, and tests that measure the network operation overhead introduced by the sensor node monitoring agent. It should be noted that, as presented in section 2.2.3, the industrial standards addressed in this paper already share network metrics to perform routing and other network related operations and, thus, these metrics do not add extra overhead to the network operation. Hence, the only network overhead is the one caused by sensor node monitoring agents.

For the first group of tests, the used hardware and software were, respectively, the Texas Instrument MSP-FET (with EnergyTrace technology support) and the Code Composer Studio (CCS) connected directly to the launchpad debug pins and bypassing the board debugger. In the second group of tests, the gateway monitoring agent was used to collect the impact of the traffic generated by the sensor node monitoring agent on the network and by collecting device health reports. The results presented in figure 4.4 show (a) the impact of the hardware techniques on the sensor node battery lifetime; (b) the overall impact of the sensor node monitoring agent on the sensor node lifetime; (c)(d) the impact of the sensor node monitoring agent on the microcontroller processing and memory, respectively; and (e) the overhead caused by the transport of the sensor node monitoring data over the network.

In order to measure the impact of the hardware monitoring metrics on sensor node resources, a simplified version of the industrial application was developed with the objective of guaranteeing the independence of each added metric. In this test, the energy expenditure was measured by the MSP-FET. Each test

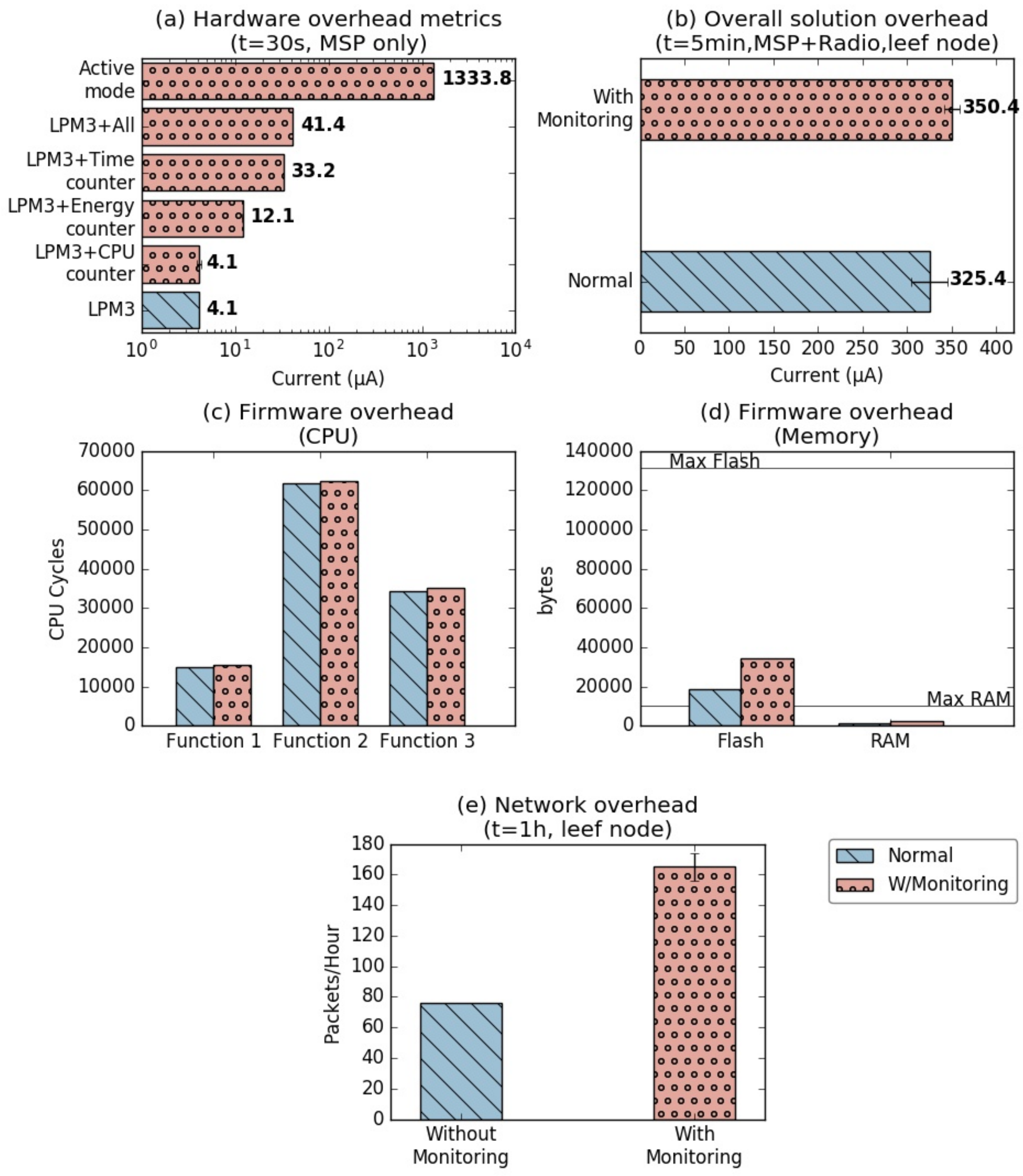


Figure 4.4: Obtained results

was performed during 30 seconds. Additionally, the radio was disconnected in order to allow the easy detection of the comparatively small increase of energy expenditure caused by each hardware metric.

As shown in the figure 4.4 (a), the energy spent by the MSP430 in LPM3 without any hardware metric enabled is around 4.1µA. Enabling the processing metric does not increase the energy spent by the microcontroller. In this LPM, the MCLK and the SMCLK are turned off and, consequently, the processing metric interrupt will not occur when the microcontroller is in LPM. On the other hand, the energy counter increases the microcontroller energy expenditure in LPM, because even in this LPM mode the system consumes energy and, as a result, the interrupt associated with the energy counter mechanism will be executed (this interrupt is tied with an hardware counter that is sourced by an external clock). In this mode, when enabled, the microcontroller counter consumes 12.1µA (σ=0.00). Lastly, the time counter, sourced by the ACLK consumes around 33.2µA (σ=0.05).

Contrary to the significant increase in energy consumption in LPM associated to each metric, the energy spent by the complete solution in active mode with the radio on (acquisition plus transport of the monitoring data) only corresponds to a 3.5% increase (figure 4.4 (b)) in relation to a non-monitoring situation. Using two AA NiMh batteries (1.2v, 1900mA), the lifetime of the sensor node is 658 days (σ=6) without monitoring and 635 days (σ=12) with the sensor node monitoring agent enabled.

The latency generated by adding the monitoring mechanisms to the sensor nodes firmware, as well as the microcontroller's memory taken by the monitoring mechanisms are also two relevant aspects in assessing a monitoring solution. The firmware graphs, in figure 4.4 (c) and (d), show the number of cycles executed by the microcontroller in three firmware functions, and the RAM and ROM requirements, respectively, with and without monitoring. In terms of processing overhead, the manual instrumentation performed in the code of each function inserts an average of 758 cycles (σ=3), adding 93µs of latency at 8MHz clock. As can be seen in (c), the overhead is quite small, when compared to the cycles spent to execute the functions without monitoring, representing 5.1%, 1.2%, and 2.2%, for functions 1, 2, and 3, respectively. An analysis in terms of memory was also made, measuring the increase of flash memory and RAM. In what concerns flash memory, the main application without monitoring occupies 18.590KB, as represented in figure 4.4 (d). Adding the sensor node monitoring agent and the instrumentation code, the application size increases to 34.659KB. This represents 12.26% of the total capacity of the flash memory in this microcontroller version (131.072KB). Lastly, as also presented in figure 4.4 (d), the main application without monitoring takes 1.631KB of RAM. Adding the sensor node monitoring agent to the application increased RAM usage to 2.506KB. When compared to the initial utilization, the sensor node monitoring agent only consumes an additional 8.54% of all the RAM available in the microcontroller (10.240KB).

Table 4.2: Evaluation results overview

Metric	**Value**
Node Lifetime	-3.5%
Overhead latency	93μs
Flash overhead (each instrumented function)	+13 bytes
RAM overhead (each instrumented function)	+14 bytes
Flash usage	*12.26%
RAM usage	*8.54%
Network traffic	+46%
*when compared with the max capacity available	

The network overhead caused by the sensor node monitoring agent was also measured. Using health reports collected every fifty minutes by the gateway monitoring agent, we were capable of measuring the traffic generated by a leaf node in normal operation and with the monitoring capabilities enabled. As can be seen in figure 4.4 (e), the main application sends 76 packets/hour (σ=0) on average. Most of this traffic is generated by the publish service that sends the temperate values every minute. Enabling the sensor node monitoring agent causes a traffic increase of 46%, sending an average of 165 packets per hour (σ=9), of which 89 packets are generated by the sensor node monitoring agent. In WirelessHART, the IEEE802.15.4 payload is limited to 94 bytes per packet. Thus, each packet sent by the sensor node monitoring agent is capable of carrying 6 log entries, each with log ID and function details (function id, duration metric, processing metric, and energy metric).

Summing up, the assessment of the proposed monitoring architecture, carried out through this proof-of-concept implementation, shows its low impact and high efficiency in what concerns sensor node resources and network. Table 4.2 summarises the obtained results. By causing a mere 3.5% reduction in sensor nodes' battery lifetime, introducing a 93µs latency overhead in each function, occupying 12.26% of flash memory, and using 8.54% of RAM, the implementation enabled the monitoring of hardware and firmware in sensor nodes. Despite the increase of 46% in network traffic generated by sensor nodes, the proposed solution, here implemented for WirelessHART, used network resources in a smart way. The monitoring data was only sent when the network had resources to send it, requesting the service and deleting it each time the sensor node monitoring agent needed to send monitoring data. Using this approach, the sensor node monitoring agent only requested free network resources, not used by the main industrial application.

4.7 Related Work

To the best of our knowledge, the architecture presented in this chapter is the first multi-domain architecture that proposes a solution to monitor the hardware, firmware and the network condition of IWSNs in the process-automation domain. Seeing that, in this section, a comparison is only possible within solutions presented in the context of Industry 4.0, specifically, solutions that improve the reliability of the network and integrate monitoring techniques. Additionally, only solutions proposed to the industrial standards addressed were considered.

From the first perspective, the authors of [Quan Wang, 2010; Grimaldi et al., 2016] propose techniques that improve the reliability of the network at routing schedule-level: by using a finite-state Markov model [Quan Wang, 2010], and by specify scheduling emergency and recovery mechanisms when a path-down alarm occurs[Grimaldi et al., 2016]. On the other hand, the work presented in [Kunzel et al., 2012; Neumann et al., 2017; Lampin and Barthel, 2018] proposes several techniques and platforms that can be used to monitor some IWSNs. In [Kunzel et al., 2012] the authors develop a passive monitoring tool for evaluation of the deployed WirelessHART networks using passive sniffers deployed in the network area. Additionally, in [Neumann et al., 2017] the authors propose an hybrid monitoring technique to monitor wireless and wired industrial technologies in the scope of the project HiFlecs. Lastly, in [Lampin and Barthel, 2018] the authors present a debugging tool that is able to collect in the SensorLab2 environment routing topology information and end-to-end performance, by debugging the information using UART ports and then convert it to TCP/IP messages. Finally, some authors also show some concerns in the lack of integration solutions between the industrial standards [Jose Da Cunha et al., 2017; Tanyakom et al., 2017; Teslya and Ryabchikov, 2018]. In [Jose Da Cunha et al., 2017] the authors presented how the Message Queuing Telemetry Transport (MQTT) protocol can be used together with ISA S5.1 and ISA95/88 standards to represent the monitoring and control in topics using the URI scheme. Lastly, in [Tanyakom et al., 2017] the authors present how the WirelessHART standard and the ISA100.11a standard can be integrated using the Modbus protocol and a proprietary software called Wonderware InTouch.

When compared the reliability solutions already proposed by other researchers with the architecture proposed in this chapter some differences appear. Most of the work performed in this field is not taking in consideration the main components that may produce faults in a IWSN (as described in chapter 3). Additionally, most part of the work focus on a specific problem like [Quan Wang, 2010; Grimaldi et al., 2016], and do not address the needs of monitoring several process automation networks at the same point. Additionally, some of the works that present solutions to monitor these standards [Kunzel et al., 2012; Neumann et al., 2017; Lampin and Barthel, 2018], keep proposing passive monitoring tools that need extra networks to be installed, adding extra costs to a technology designed to be low-cost. Lastly, most of the work [Jose Da Cunha et al., 2017; Tanyakom et al., 2017; Teslya and Ryabchikov, 2018] in the field of integration focus their effort in the representation of the sensor data, and any of them try to explore the use of management protocols like LWM2M and COMI (protocols specific for constrained devices) to represent and make available the monitoring data.

4.8 Summary of the Chapter

As a starting point for new management models and diagnostic tools, we strongly believe that the architecture presented in this chapter will open new research directions and proposals that will lead to better diagnostic tools for IWSN standards (ZigBee, WirelessHART, ISA100.11a, and WIA-PA). Specifically, the proposed architecture offers the possibility of having IWSN management systems able to monitor individual sensor nodes and the network as a whole, in both the development and deployments phases.

Being a comprehensive architecture that addresses the most important industrial standards in the process-automation domain, addressing several hardware technologies, and several firmware architectures, it was not practical to build a prototype that demonstrate all the technologies and models that can be supported. Thus, in this chapter, when the architectural components were described, some possible solutions were presented in order to provide some guidelines for further implementation. Nevertheless, as shown by our proof-of-concept implementation, the architecture can be used to perform the monitoring of a variety of sensor and network-wide parameters in a way that does not increase the cost of node manufacturing, does not have significant impact on sensor node resources, and does not steal network bandwidth from the main application.

In the next chapter, the proposed architecture will be explored in the three proposed dimensions (hardware, firmware and the network) in order to show its effectiveness.

Chapter 5

Attack, detect and explore new vulnerabilities in WirelessHART

"When Wireless is fully applied the earth will be converted into a huge brain, capable of response in every one of its parts."

(Nikola Tesla)

Contents

Industrial Control Systems are now exploring the use of Internet of Things technologies not only to make them fitter to their job but also to explore the advantages that come from connecting them to the Internet. Nevertheless, with this paradigm shift, new threats appear, of which the stuxnet worm is just an example, and Intrusion Detection Systems architectures and solutions were and still are being considered. However, most existing projects concentrate on high level system aspects and thus neglect security aspects at wireless communication standards level, such as WirelessHART (the standard with largest market share), choosing not to address security solutions to common, known attacks identified by the community. In this chapter, using the monitoring architecture proposed, we will monitor the WirelessHART testbed, and at the same time, conduct network attacks from an outsider perspective. As main contributions this chapter presents a new exhaustion attack for WirelessHART that, until now, to the best of our knowledge, has not been yet described. Additionally, the presented work proves that using classifiers like One Class Support Vector Machines (OCSVM), and our monitoring architecture we are capable to detect the new exhaustion attack and more common attacks like jamming and collision.

5.1 Introduction

ICS networks differ from networks in other common IT fields. Here, requirements like reliability, availability and security must take precedence over all the others. Standards like WirelessHART take into consideration these requirements in their design. Through a combination of techniques at the Data Link Layer (DLL), like TDMA and channel hopping (for more detail, see section 2.2.2), the standard enables deterministic communications and enhanced immunity to interferences. At the same time, by using the IEEE802.15.4 Physical (PHY) layer and part of its MAC layer, the standard is able to achieve low power consumption and low cost (namely, by using IEEE802.15.4-based radios). Additionally, at the application layer, the use of HART [Nobre et al., 2015] commands makes the technology backward compatible with existing Supervisory Control and Data Acquisition (SCADA) systems.

However, the introduction of new IoT technologies in ICS networks is shifting the connection paradigm of the critical infrastructures. Until now, critical infrastructures have been operating as separate networks, disconnected from public Internet infrastructures, but this paradigm is changing, exposing these systems to a new set of threats (e.g., stuxnet, slammer, mariposa [Pasqualetti et al., 2015]). In the case of stuxnet, the worm was used to launch cyber-attacks on important critical infrastructures, specifically against Iran's nuclear program. Consequently, in later years, several projects have emerged in this field proposing several architectural models to monitor these networks and to detect this type of threats [Maglaras et al., 2018]. However, most of these projects design their IDS for existing wired technologies like Modbus, CanBus and Profibus, neglecting the fact that wireless technologies like WirelessHART (as well as, ISA100.11, ZigBee, WIA-PA and the new 6TiSCH) are increasingly being used and present new and distinct risks, given their wireless nature and resource constrained characteristics. As a result, in this chapter, we will explore this perspective, in what concerns the WirelessHART standard, presenting the effectiveness of the presented architecture to detect such attacks.

So far, to the best of our knowledge, work performed on this topic has been made in a more theoretical perspective or/and by using simulators. In their research, the authors of [Raza et al., 2009; Alcaraz and Lopez, 2010; Bayou et al., 2016] present a set of threats to the WirelessHART standard but we were not able to identify papers that explore these threats either in real scenarios or in testbeds. Additionally, most of the proposed solutions focus on the ZigBee standard [Wang and Jiang, 2016]. Consequently, there is also a lack of solutions, specifically for the WirelessHART, as well as a more general approach to monitor the network of the four main standards (WirelessHART, ISA100.11a, ZigBee, WIA-PA). In this chapter, the monitoring architecture will be used to monitor and detect some of the network threats identified in [Raza et al., 2009; Alcaraz and Lopez, 2010; Bayou et al., 2016]. As main contributions of this chapter: 1) we prove that it is possible (by using the network metrics available in the standard) to identify and detect threats like jamming, exhaustion, and collision attacks conducted from an outsider prespective; 2) additionally, a new exhausting attack is presented, that explores vulnerabilities of the join phase of WirelessHART-based networks. To detect this new attack vector, we use not only the standard network metrics but also the in-node metrics related with the hardware and firmware (proposed in the architecture). Using a OCSVM, we show that it is possible to detect this type of attacks with good precision and recall. Lastly, this chapter also presents how these attacks were made and describes the tools and the low-cost hardware used to conduct them.

The remainder of this chapter is organized as follows, Section 5.2 presents the WirelessHART standard from a security perspective, and identifies some design flaws that can be explored. In section 5.3, the hardware tools used to conduct the attacks are presented. Section 5.4 details the conducted attacks and the best metrics that can be used for their identification. Section 5.5 presents the attacks detection results using the OCSVM classifier. Lastly, section 5.6 sums up the chapter.

5.2 WirelessHART from a Security Perspective

This section addresses two main topics. In the first topic a brief description of the security characteristics of the WirelessHART is made, complementing part of the discussion already started in section 2.2.2. In the second topic, an analysis to the already recognized security threats will be presented, using the threats identified in [Raza et al., 2009; Alcaraz and Lopez, 2010; Bayou et al., 2016]. The analysis will be conducted from an outside attacker perspective.

5.2.1 WirelessHART Security

As already mentioned (see section 2.1), one of the most important requirements of the industrial networks is the determinism of the communication processes. To guarantee low network latency, WirelessHART networks use a MAC scheme that combines TDMA techniques with frequency changes. On one hand, in the time domain, all nodes should synchronize with their neighbours to know the precise timing to transmit or to listen, according to the Absolut Slot Number (ASN). The network consists of a set of timeslots with 10 milliseconds duration, that are combined to create one or several superframes. Each timeslot represents a communication opportunity between two nodes and can be contention free or not. On the other hand, in the frequency domain, sensor nodes use a pseudo-random channel each time they need to listen or transmit, statistically avoiding possible channel interferences. The frame and timeslot structures are presented in figure 5.1. A timeslot guarantees the transmission of the information and its acknowledgement. Before the sender starts transmitting, the receiver node turns on the radio in RX mode, during a guard time. If the receiver node does not receive data in this period, then the node goes back to sleep mode and waits for the next timeslot. Otherwise, it waits the transmission to finish, validates it and sends the acknowledgement to the transmitter node. In [Wang and Jiang, 2016] the authors present an exhaustive analysis of the standard's architecture.

In terms of security [IEC, 2010; Raza et al., 2009; Alcaraz and Lopez, 2010; Bayou et al., 2016], the standard offers confidentiality, authenticity, and integrity, either in hop-by-hop and in end-to-end communication, using the Advanced Encryption Standard (AES) with a 128-bit key. Being a mesh network, nodes need to rely on their neighbours to forward the data to the network manager. In detail, the Data Link Layer (DLL) provides data authenticity and integrity on a hop-by-hop communication basis (at this level, data is not encrypted), protecting the network from outsiders (devices that are not part of the network and may compromise the network through physical or logic attacks[Alcaraz and Lopez, 2010]). Additionally, end-to-end security is provided by the Network Layer (NWK) (only the header of the NWK frame is not encrypted), protecting the end-to-end communication from malicious insiders.

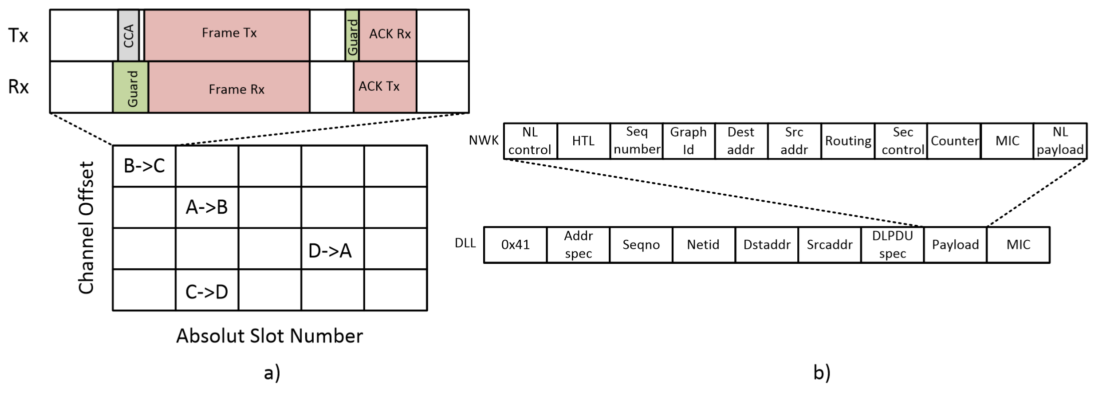


Figure 5.1: On the left (a) the timeslot structure in time and frequency. On the right, (b) the DLL frame and the NWK packet structures

As presented in figure 5.1 (b), the DLL and NWK layers have a Message Integrity Code (MIC). This field provides source integrity, by authenticating the sending device as a valid neighbour. The MIC is calculated using the AES-CCM mode with the following parameters: a 128-bit network key, a 13 byte-string that contains the ASN concatenated with the source address, and the Data Link Layer PDU (DLPDU) to be encrypted. On the other hand, the MIC of NWK layer is calculated over the entire Network Layer Protocol Data Unit (NLPDU) using: the NLPDU header, the NLPDU payload, a 13-byte nonce, and the 128-bit session key (SK), used in communications between devices and the gateway.

To achieve different levels of security, the standard defines a set of security keys that allow to secure the exchange of messages between the network devices and/or with the Network Manager (NM) hop-by-hop and end-to-end. The keys specified in the standard are: 1) the well-known key (WK), 2) the join key (JK), 3) the network key(NK), 4) the broadcast session key (BK) and a 5) unicast session key (UK). Firstly, the WK (1) is a special key that is already defined in the standard specification and used in the transmission of the advertisement messages between the joining device and the devices already connected to the network. Secondly, the JK (2) is manually distributed by the Security Administrator (SA) in the deployment phase. This key acts as a password that the device uses to authenticate with the NM in the join procedure. The key is used to receive the remaining keys (NK, UK and BK) in the security handshake. Thirdly, the NK (3) is distributed by the NM to the joining devices, in the security handshake, and used to compute the MIC of the DLPDUs to authenticate the communication between two neighbour devices. Lastly, the BK (4) and the UKs (5) are used to send unicast and broadcast data end-to-end between the network manager and the network devices.

After the analysis performed in this section is easy to understand that the security of the WirelessHART protocol depends of the secrecy of the JK. If an attacker was able to obtain this key, the attacker can initiate join procedures by itself and connect to the network in order to obtain all the remaining keys in security handshake (NK, UK and BK). Having access to this information, the attacker will be able to compute the MIC of messages, and as a result exchange valid messages with the neighbours and with the NM. Thus, the attacks made to a WirelessHART network can be divided in two types: outsider attacks, attacks made by nodes that do not have access to the join key; and insider attacks, nodes that have access to the JK and, consequently, to all the other keys.

5.2.2 Threat Analysis

Despite the specification of several security measures in wireless network standards, these networks are still the target of several attacks. WirelessHART is not an exception. In [Raza et al., 2009; Alcaraz and Lopez, 2010] the authors present a security analysis of the standard, describing some of the threats that can be explored by malicious attackers. In this analysis we will focus on the terminology and on the concepts used in [Alcaraz and Lopez, 2010], and, as already mentioned, from an outsider perspective. Only the relevant attacks and security concepts for this topic will be analysed.

In [Alcaraz and Lopez, 2010] the authors split their threat model into three main groups: confidentiality, integrity and availability attacks. Attacks on confidentiality are related with the capability of an attacker to obtain unauthorized access to any information exchanged in the network. On the other hand, integrity attacks are related to the capability of an attacker to manipulate information in exchanged messages (e.g., routing messages), and use it to change the behaviour of the network. Lastly, availability attacks try to disrupt the network, preventing it from providing its services. This type of attack in WSNs is usually performed with the aim of exhausting the sensor nodes resources, such as their energy.

In what concerns availability attacks, a WirelessHART network may be exposed to attacks like: jamming, interference, collision, exhaustion and network isolation. Using the Time Slotted Channel Hopping (TSCH), and its frequency hopping method, a WirelessHART network may prevent attacks like jamming, interference and collision, by blacklisting the channels where the attacks occurs. However, an attacker with sufficient resources may launch the attack in all the 16 channels. An attacker may also send data in these channels to generate collisions. The collisions can be detected by the Cyclic Redundancy Check (CRC) field available in the DLL frame. These attacks can also be a DoS attack [Raza et al., 2009]. The authors of [Raza et al., 2009] also defend that any device that supports WirelessHART and has knowledge of network parameters like network id, device id, and the well-known key can send a "valid" JK requests to neighbouring devices authenticated with the WK. The idea of faking a message like this is to force the neighbour nodes to consume resources in the transport of the packet and its analysis. At the end, the message will be discarded by the network manager because it contains an invalid JK. On the other hand, attackers that want access to network parameters like the network id, and the ASN used at the attack moment, need only to sniff the advertisement messages sent by the network manager and perform traffic analysis. As mentioned before, only the NLPDU payload is encrypted. Moreover, the authors of [Alcaraz and Lopez, 2010] also present a form of isolating a network by overloading the gateway with multiple TCP/Modbus commands and alarms. In the current chapter, a different type of isolation attack will be described in section 5.4.2.

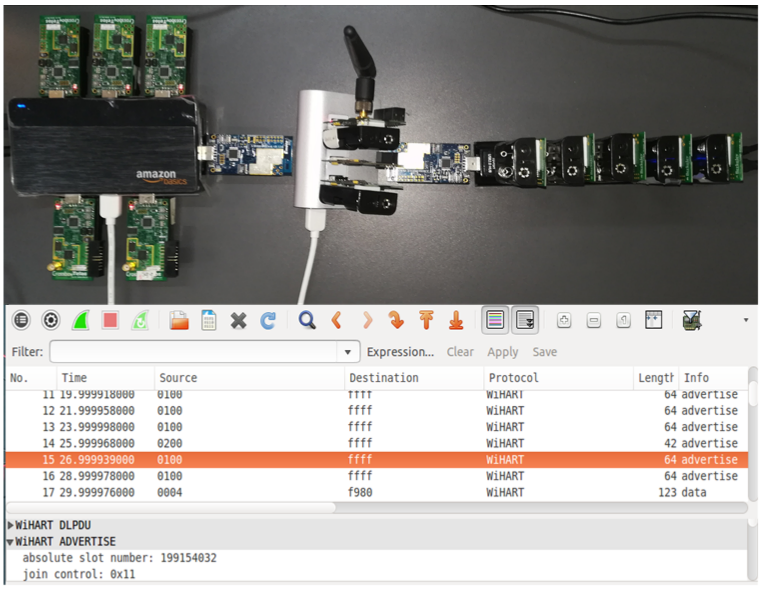
In what concerns integrity attacks, the authors of [Alcaraz and Lopez, 2010] argue that WirelessHART prevents information manipulation network attacks by supporting the use of different SK to generate the MIC between the NM and the network devices, the network key to encrypt the message, and by its nonce that can be used to validate the age of a packet. However, there is a situation explored in section 5.4.2 (the new attack explored in this chapter), where a sensor node may be the target of an information replay attack.

Lastly, as previously mentioned, confidentiality attacks can also be performed. An attacker with an IEEE802.15.4 radio can sniff a WirelessHART frame and have access to the contents of DLL packets and to the NLPDU header. This attack may help attackers to sniff packets like advertisement messages, periodically sent by the network manager, and obtain important data like the ASN, the link join slot, channel map size, etc.

5.3 Attack Tools

KillerBee [Charles, 2018] is an open-framework initially developed by Joshua Wright for attacking ZigBee and IEEE 802.15.4 networks. The script offers a set of tools written in Python that interact with the GoodFET interfaces, allowing the manipulation of IEEE802.15.4 radios like the CC2420. The framework offers tools like the zbwireshark, zbdump, and zbreplay. Zbwireshark is an application with sniffing capabilities that communicates with the Wireshark application, dumping the IEEE802.15.4 frames in Wireshark. Similarly, zbdump also performs sniffing, its output being dumped to a Pcap file. On the other hand, zbreplay allows to launch replay attacks in IEEE802.15.4 radios.

For analysing the WirelessHART network, we choose the TelosB sensor nodes, a well-known IEEE802.15.4 radio (CC2420), supported by the GoodFET firmware and the KillerBee framework. To sniff the WirelessHART network, an array with several TelosB capable of capturing the packets in all the 2.4GHz channels was used. After installing the GoodFET firmware in the TelosB radios and using the zbdump in each channel, we were able to collect the WirelessHART frames. Additionally, in order to analyse the contents of the IEEE802.15.4 frames, a WirelessHART dissector developed by [Wightman, 2018] was used. The output can be seen in Figure 5.2 (b). Lastly, the Contiki rssi-scanner was used to analyse RSSI changes.



```
No.   Time            Source    Destination   Protocol   Lengtl Info
   11 19.999918000    0100      ffff          WiHART         64 advertise
   12 21.999958000    0100      ffff          WiHART         64 advertise
   13 23.999998000    0100      ffff          WiHART         64 advertise
   14 25.999968000    0200      ffff          WiHART         42 advertise
   15 26.999939000    0100      ffff          WiHART         64 advertise
   16 28.999978000    0100      ffff          WiHART         64 advertise
   17 29.999976000    0004      f980          WiHART        123 data

▶WiHART DLPDU
▼WiHART ADVERTISE
   absolute slot number: 199154032
   join control: 0x11
```

Figure 5.2: Attack tools

5.4 Attacking an WirelessHART Network

As presented in section 2.2.2, the WirelessHART standard provides several security mechanisms that make the network more robust to attacks. Firstly, sensor nodes use a pseudo-random sequence that allows the nodes to communicate in different channels in the same timeslot. Thus, it's difficult to follow the communication of one sensor node and, consequently, several sniffers need to be used in order to collect the traffic generated in all the channels. Additionally, to authenticate the packets at the different layers (DLL and NWK), the standard authenticates each layer with a MIC generated by a nonce, that changes with time and it is based on the ASN and use distinct keys. To forge a packet, the attacker needs to be synchronized with the network and know the exact timeslot, channel, and key that the node will use for transmission in order to generate a valid MIC accepted by the receiver (use the NK to send to neighbours or one of the SKs if the message is sent to the NM).

Lastly, as presented in Figure 5.1 (a), to receive such a packet, the attacker needs to transmit the packet in the guard time period (need to be synchronize). However, despite these aspects that turn hard the reception of a malicious packet, there are a set of attacks that can be made to the network without need to synchronize the attackers with the network. In this section two types of attacks will be used: firstly, we perform the most basic attack, a jamming attack; secondly, we explore an information manipulation attack that, at the same time, performs exhaustion of the sensor node resources (e.g., battery), and the isolation of the new joining devices. As far as we know, this attack has not been described in the literature.

To perform the attacks against the network, we start by putting a TelosB node in promiscuous mode to capture network packets with the zbwireshark tool (e.g., data, ack, keep-alive, advertise and disconnect DLPDUs). To perform the two attacks, two types of packets were captured: an advertisement packet and a data packet sent between two nodes.

5.4.1 Jamming

Because of sharing the physical medium like the other wireless technologies, jamming attacks are the simplest that can be performed over a WirelessHART network (see section 3.4.2). The attack consists in the disruption of the original radio signal, using another signal with the same modulation technique and frequency. As a result, jamming attacks can be performed by WirelessHART compatible nodes and by any other device that is able to reproduce a signal with these characteristics. In order to evade jamming attacks, the WirelessHART network can use the channel blacklisting feature. However, this functionality is not well defined and described in the standard. For instance, when the current testbed was installed, we notice that the NM only supports the manual insertion of the noise channels in the blacklist. Thus, we did not find evidence that the NM has support to automatic detection of the noisiest channels.

To perform a jamming attack, there are two types of attacks that can be made. In the first, the attacker performs it by emitting a continuous signal. The radio needs to be continuously in the TX mode. However, this attack cannot be performed with current release of the KilllerBee tool, because this feature has not already been implemented in the TelosB interface. The second approach to interrupt the WirelessHART network is by transmitting IEEE802.15.4 packets in a fast-enough rate that is capable to collide with existing network communications. The collision will make the packet to be dropped and the victim node will postpone the transmission of the packet for the next timeslot. Additionally, the WirelessHART also support in each timeslot, the use of Clear Channel Assessment (CCA), that can mitigate this attack. However, we note that this feature is disable by default. To replicate this attack using the KillerBee tool in combination with the TelosB nodes, some changes in the dev_telosb.py code were done. Here, the code as a limitation that waits 10 milliseconds for a transmission to complete, not allowing the sensor nodes to continuous transmit packets to a higher rate. In our jamming attack we change this value to 2 milliseconds, improving the capacity of the collisions.

Lastly, to simulate the difference between broadband and narrowband interferences, the jamming attack was performed in two distinct ways. By putting the array of attackers in one channel (narrowband), or by splitting it by the available WirelessHART channels (broadband). Additionally, to understand the impact of distinct packet types with distinct packet sizes, we also performed the attack by using unicast packets and broadcast packets.

5.4.2 Advertisement Based Attack

In section 5.2, after presented the several keys defined by the standard, an important statement was made. The security of the WirelessHART hardly depends on the secrecy of the JK. If an attacker obtains access to it, the other remaining keys can be retrieved in the security handshake. As a result, a list of insider's attacks can be performed, if the JK is collected. However, there is an interesting attack that can happen before the handshake procedure, and resides in the fact that, the WK is defined and documented in the standard specification. Thus, in this section we present an attack that performs the resources exhaustion of joining nodes, and at the same time its isolation, without the access to the JK and the remaining keys obtained in the security handshake. Additionally, this attack can also be seen as an information manipulation attack. To understand it, we will start by presenting the distinct phases of the joined procedure defined in the standard, and then presenting some of the characteristics that allow the advertisement packet to be forged.

The joining of a new device in the network starts by the configuration of the network ID and the JK. After that, the device enters in the join procedure, changing it state to the "listen mode". In this state, the device will try to listen for DLPDUs for a defined amount of time, changing between channels if needed, in order to synchronize with the network. When the device is in a synchronize mode, the device is already in a state that is able to receive the advertisement messages. Lastly, before proceeding to the join request, the device configure itself with some of the data contained in the advertisement message.

Until now, the new device only knows two security keys, the JK (that will be used in the join procedure) and the WK (that is used to authenticate the advertisement message). When an advertisement is received, the new device compares the network ID of the advertisement message with the network ID configured previously. Additionally, the advertisement message is validated by its MIC using the WK. When the packet arrives, the new device collects some of the valuable information needed to join the network. Advertisement messages transport in its payload network information like: the ASN, the size of channel map array, the graph structure, the number of the superframes and the join links that the new device should be used to start the join procedure. Faking such message with inaccurate data will force the joining device to send a join request that will neither be listening or accepted as valid by the NM. The node will wait until the timeout of the join request expires and will need to start all the process again, expending sensor nodes resources.

Faking an advertisement message in this context is not difficult, perhaps it is relatively easy, and there are two ways: 1) the attacker performs traffic analysis and collects the network ID, builds an advertisement message and computes its MIC with the WK; and 2) the attacker collects an old advertisement message, with outdated ASN and join links. This last message is valid to the joining device, because in this phase the device is not able to understand the age of the packet (does not know the ASN of the network). Furthermore, if this attack was performed together with a de-authentication attack, it is possible to put all the network down without the opportunity to recover and re-join the nodes again to the network. In order to perform this attack over the WirelessHART testbed, we choose to replay an old advertisement packet, previous captured in an older network configuration phase.

5.5 Detecting the Threats

To monitor the network against security threats, the monitoring architecture collects two distinct sets of data: sensor node metrics and network-related metrics. As presented in section 4.6.2, the sensor node monitoring agent collects the number of MCU cycles, functions execution time, energy consumption, and call traces. On the other hand, the collected network metrics are the ones already defined by the WirelessHART standard, and are used by the network manager to perform its tasks. Specifically, the metrics comprise device health reports, neighbour health reports, and the neighbour signal levels. Device health reports sum up all the communication statistics of a network node. These include: packets generated by the device, packets terminated by the device, MIC failures at DLL or NWK layers, CRC errors, and nonce errors. Additionally, neighbour health reports include all the statistics of a network node concerning its connected neighbours. The report contains: number of linked neighbours, Received

Signal Level (RSL), packets received from the neighbour, failed transmissions, and packets transmitted. Lastly, the neighbour signal levels include the RSL of all linked and non-linked neighbours. This report was not used in the work presented in this thesis.

In order to model the normal behaviour of our network, a normal dataset was acquired for two months. On the other hand, two anomaly datasets were also collected: the jamming attack dataset, and the exhaustion attack dataset. The jamming attack dataset is divided into two attacks, performed in two distinct

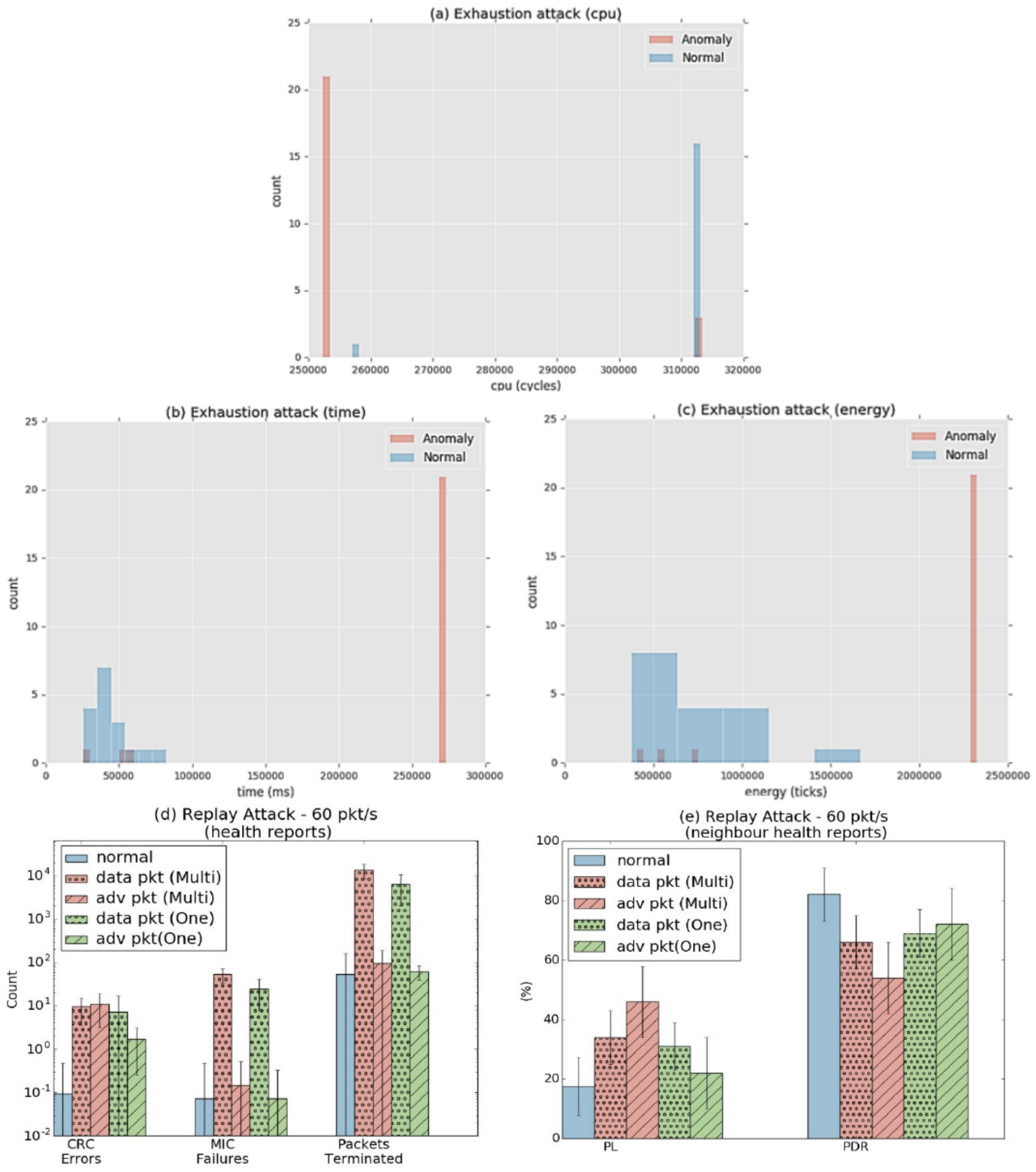


Figure 5.3: WirelessHART attacks overview

modes: a replay attack in continuous mode of a broadcast packet (DLL advertisement with 60 bytes, generating 28.800Kbps per attacker node), and a replay attack of a unicast packet (DLL data packet with 16 bytes, generating 7.680Kbps per attacker node). The attacks were performed in the one-channel and multi-channel modes, each one with 6 hours duration (1 day for the full attack combination). Lastly, the exhaustion dataset was also collected by sending fake advertisement packets, captured at an earlier period, to the network. The packets were sent in multi-channel mode to maximize the probability of listening by the new devices. A reset was performed to the sensor nodes to force a new network join. Figure 5.3 presents an overview of the two attacks in the different modes. Figure 5.3 (a), (b) and (c) represent the impact of the exhaustion attack made in the joining phase on sensor node resources in terms of CPU, execution time, and spent energy. Additionally, Figure 5.3 (d) and (e) represent the jamming attack and its impact, in terms of network health and neighbour health.

The impact of the replay attacks (jamming) is notorious in all the attack combinations, but with some particularities. The CRC metric increases in all the variations of the attack but is less expressive in the one-channel mode case due to the lower probability of packet collisions. Additionally, other important metric is the DLL MIC failures. The metric represents all the fake MIC detected by the DLL. Detection of DLL MIC failures requires to perform the verification of the field with the AES-128 (counter with CBC-MAC). As presented in 5.3(d), the metric increases when data packets (unicast) are replayed, but not with advertisement (broadcast) packets. The packets terminated metrics (packets terminated by the DLL) also increments when data packets are replayed. Lastly, by using the metrics available in neighbour health reports, it's also possible to compute the Packet Loss (PL) and the packet delivery ratio (PDR) of each node. As presented in Figure 5.3, the heaviest attack is the replay attack, which generates more traffic (28.800Kbps in each channel).

Differently from the replay attack, the exhaustion attack cannot be detected by network reports. At this phase the radio is not connected to the network

Table 5.1: Network diagnostic using the health packets

Step	**Description**	**Broadcast Replay**	**Unicast Replay (each test)**
Train	50% normal events(n)	4460(n)	4482(n)
Validation	25% normal events(n) 50% abnormal events(a)	2230(n) 10(a)	2241(n) 8(a)
Test	25% normal events (n) 50% abnormal events (a)	2230(n) 10(n)	2241(n) 8 (a)

Table 5.2: OCSVM classifier results

Attack	**Mode**	**Precision**	**Recall**	**F1-score**
Broadcast replay (*health reports)	Multi-channel	0.70	0.538	0.609
Unicast replay (*health reports)	One-channel	1.0	0.615	0.762
	Multi-channel	1.0	1.0	1.0
Exhaustion (Join) (firmware log)	Multi-channel	1.0	0.722	0.839

and does not generate any report. However, as presented in Figure 5.3 (a), (b) and (c), the attack can be identified through sensor node log metrics. In this attack, the node receives a fake advertisement and, as already explained, it cannot detect if it is fake or not. Thus, by sending a join request assigned with a wrong ASN to a non-existing join link, the radio will wait until the request times out. As presented in (b) and (c), the extra time spent in this function clearly points to an abnormal behaviour. Furthermore, the node wastes more energy in this period. On the other hand, the CPU cycles spent by the function are less, because the node needs to compute different actions in each of the two scenarios.

Lastly, the OCSVM was used to detect the anomalies in each attack scenario. The OCSVM classifier is one of the most appropriate algorithms to classify anomalies in WSNs due to its semi-supervised capabilities (the algorithm only needs training with normal data) [Chandola et al., 2012]. To start, a selection of the most representative features was made for the health reports and the firmware log datasets. After that, the data was standardized and divided randomly as follows: train (50% of the normal datasets); validation (25% of normal dataset and 50% of anomaly datasets); and lastly, testing (25% of the normal datasets and the remaining 50% of the anomaly datasets), see table 5.1. All the data was chosen randomly. To train the OCSVM classifier a radial basis kernel was selected. Additionally, at validation phase a grid-search was used to tune the hyper-parameters. The results of the classifier are presented in table 5.2. The classifier obtained good results in the detection of the exhaustion attack (f1=0.839), and in the unicast replay attack (one-channel: f1=0.762 and multi-channel: f1=1.0). In the broadcast replay, despite the increase of the mean in metrics like the CRC errors, the PL, and the PDR, the classifier performed badly and was not able to discriminate the anomalies.

5.6 Summary of the Chapter

In the last years, new security threats to industrial control systems appeared and, consequently, new solutions also emerged. However, some of these new solutions do not address IWSN standards like WirelessHART. In this chapter, the effectiveness of the proposed architecture was evaluated, as well as, to the best of our knowledge, a new security threat that has never been described. By using the monitoring architecture proposed in this thesis, we proved that it is possible to detect jamming and exhaustion attacks, using standard WirelessHART network metrics and firmware based-node metrics. Additionally, this chapter also demonstrated how easy and cheap it is to attack networks using this standard.

The effectiveness of the proposed architecture, in terms of anomaly detection on in-node components (hardware and firmware) is presented in the next chapter.

Chapter 6

Security and Fault Detection in In-node components of IIoT Constrained Devices

"I can predict things. I can improve the uptime and the reliability. I can intervene and cause a better outcome before there's a problem."

(Michael Dell)

Contents

In order to keep up with the requirements of new Industry 4.0 applications, sensor nodes software and hardware are becoming more complex and, thus, more prone to faults. In this chapter, using the monitoring architecture proposed in this thesis, we injected and subsequently proceeded to detect representative firmware and hardware anomalies (namely, buffer overflow attacks, SPI faults, under-voltage, and high temperature faults) that can be used by attackers to cause major losses or even damage industrial control systems. We evaluated the performance of several machine learning techniques commonly used to detect anomalies (i.e. OCSVM, kNN, AutoEnconder), in order to determine if they could be useful to detect such faults. The obtained results demonstrate that simple and broad scope classifiers, using features that consume little resources, can be developed to detect such faults.

6.1 Introduction

Until the last decade, ICS largely remained disconnected from the Internet, relying on proprietary technologies and using the airgap principle to ensure security [Foglietta et al., 2018]. With the adoption of common technologies like the TCP/IP and Ethernet (e.g., Modbus TCP/IP), new threats emerged for such systems. At the same time, new wireless standards were developed, designed for fulfilling the requirements of industrial applications. IEEE 802.15.4 [Wang and Jiang, 2016] based standards like ZigBee, WirelessHART, ISA100.11a, WIA-PA, and 6TiSCH can offer low-cost and low-latency solutions for some types of industrial applications and, at the same time, coexist with the legacy technologies, as presented previously (see section 2.2.2).

Until recently, known attacks only targeted wired-based industrial technologies or the SCADA software that controls industrial systems. However, as can be seen in the literature [Granjal et al., 2015; Giannetsos et al., 2010], wireless-based technologies are also subject to such attacks, in different perspectives. Differently from wired technologies already in the field, wireless-based products are less mature in terms of development (e.g., different types of hardware architectures, fragmented operating systems, bare metal development), and typically use resource constrained components. As pointed in section 1.1, all of these facts contributed to the lack of monitoring systems in IWSN. Moreover, despite the efforts to secure industrial systems, current proposals only address technologies such as ModBus, CanBus and Profibus[Maglaras et al., 2018], neglecting the fact that the Compound Annual Growth Rate (CAGR) of the wireless technologies amounts to 30% [Kim et al., 2019], and are being operated with well-known security vulnerabilities.

In this context, a monitoring architecture was proposed in chapter 4, that is able to monitor the condition of hardware, firmware, and the network of the four main industrial wireless standards (WirelessHART, ISA100.11a, WIA-PA, and ZigBee), with small impact on the network and on the sensor nodes resources, without the addition of new hardware (as proved in chapter 4). In chapter 5, the architecture was used to effectively monitor and detect a novel network attack to the WirelessHART standard. Being an architecture that also focuses on the condition of hardware and firmware, in this chapter the referred architecture is used for detecting representative firmware and hardware faults injected in the WirelessHART testbed. The injected faults include: 1) a buffer overflow vulnerability used for smashing the call trace of devices with the Von Neuman architecture [Giannetsos et al., 2010], and for running arbitrary code; 2) two SPI faults, that simulate sensor stuck-faults, 3) an undervolting fault, using six different voltage levels, and 4) a high temperature fault [Hutter and Schmidt, 2014], commonly used by attackers to manipulate the MCU behaviour, and extract information. After the injection of the mentioned faults, three different types of semi-supervised machine learning algorithms for outlier detection were used: a linear-model based, namely One-Class SVM (OCSVM); a proximity-based algorithm, namely k-Nearest Neighbours (kNN); and a neural-network-based algorithm, namely AutoEncoder.

In view of the stated objectives, the remainder of this chapter is organized as follows. Section 6.2 details the various injected anomalies and discusses the impact of the faults on the used monitoring metrics. Section 6.3 presents the performance results for each classifier, in what concerns their ability for detecting the injected anomalies. Lastly, section 6.4 provides the conclusions and sums up this chapter.

6.2 Injecting Anomalies

The assessment of the proposed, from the in-node components perspective, was made by injecting several abnormal events/behaviours in the testbed. This section provides details on the injected anomalies, explaining how they were injected, what are they used for, and what are their consequences on the various metrics under consideration. From the firmware perspective, a stack-based buffer overflow fault was injected that enabled us to run malicious code. On the other hand, from the hardware perspective, low-voltage faults were used, as well as SPI communication faults and high temperature faults.

6.2.1 Firmware: Stack-Based Buffer Overflow

Memory-related vulnerabilities, like buffer overflows, are one of the most studied vulnerabilities in the WSN domain (see section 3.4.4). Over the years, different studies have shown that, similarly to traditional networks, it is also possible to create worms and viruses that self-propagate, infecting all the network nodes [Giannetsos et al., 2010]. One of the reasons for this lies in the intrinsic characteristics of these systems (e.g., shared RAM and ROM memory, shared access buses). Another one, lies in the use of the same program in all the nodes, which allows attackers to create worms that easily infect all the network. However, the most important reason lies in the low resources available in the nodes, and consequently on the use of weakly typed programming languages like C and C++.

Differently from the strongly typed languages, weakly typed languages do not check array bounds, for instance, allowing the program to access position n+1. Infecting the WSN firmware may have serious consequences when operating inside an ICS. In such scenario, the acquisition of data from sensor nodes cannot be trusted, and ICS and operators can act based on misleading data, resulting in potentially catastrophic events and high financial losses.

The anomaly described in this section addresses such a scenario. Here, a buffer overflow vulnerability in the code will be used to inject malicious code that will manipulate the temperature sensor with fake data. Using the monitoring architecture, an analysis of the impact of the anomaly on the monitoring metrics will be made in section 6.3. However, firstly, the injection procedure will be explained in this section.

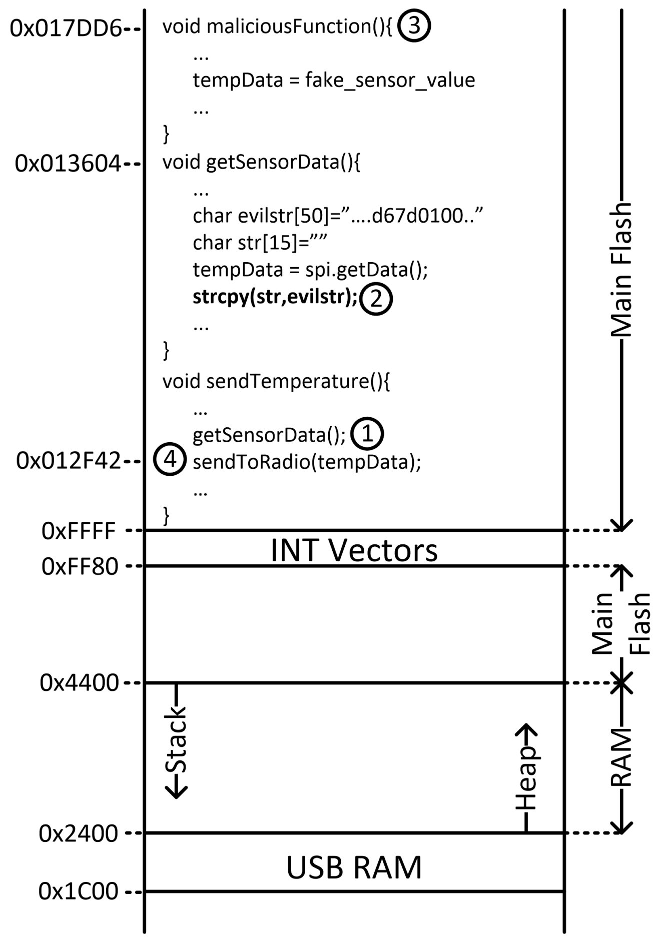
From an architecture point of view, in the WSN domain, there are two main microcontroller architectures: the Harvard and the Von Neumann families. In Harvard microcontrollers, program and data memories are physically separated. The CPU can only write in data memory and read instructions from the program memory. Thus, data memory cannot be executed by the CPU, and, as such, code injection in the stack is almost impossible [Francillon and Castelluccia, 2009]. On the other hand, the MSP430 microcontroller family uses the Von Neumann architecture, in which the memory is physically shared between data and program, using the same address space. Consequently, in this type of architecture, the CPU can load instructions from the data memory address space, and stack-based attacks can be conducted.

As presented in Figure 6.1, the MSP430F5 family used in the experiment shares its memory with RAM, FLASH, and interrupt vectors. The RAM consists of two different zones – the USB RAM (0x1C00-0x2400) and the RAM used by the CPU (0x4400-0x2400) – that implement two important data structures: stack and heap. The stack structure starts on the top of the RAM (0x4400) and grows downwards. Local variables, activation function records, parameters, interrupts and the return addresses are allocated on the stack. Here, the stack pointer (SP/R1) points to the last value pushed on the stack. Contrarily to the stack, the heap starts on the bottom of the RAM (0x2400) and grows upwards, and is used mainly for dynamic allocation. Lastly, the main FLASH can be used to store code or data. In our scenario, the main application is stored in the main FLASH and the functions used in the attack started at addresses 0x013604 and 0x017DD6.

A stack-based buffer overflow occurs when an input data runs over and overflows the stack, writing the information in a memory zone that is not intended to be used with that structure (e.g., past the buffer's upbound/writing in address n+1). If carefully selected, the location to which the SP points can be rewritten to another memory address, resuming to a different program call. Here, as shown in [Giannetsos et al., 2010], worms sent in network packets, can easily gain control over the main program, and cause significant damages to the ICS.

In the scenario of Figure 6.1, the main program is collecting the temperature from the ADT7301 sensor (over SPI), and sending it to the network. The sendTemperature() is the function that handles this process, by calling the getSensorData() function (1). This function uses a global variable called tempData. By exploiting a vulnerability added to the code (2), a buffer overflow will happen in the getSensorData() and, at the same time, the address of the malicious code will overwrite the stack space to which the SP is pointing with the 0x017DD6 address. Thus, when the function returns, the new function maliciousFunction() will be called (3). Here, a fake value will be assigned to the tempData variable, after which the code will be redirected to the old return value (0x012F42), the next instruction in the sendTemperature() function (4). From the ICS perspective, the sensor node is running normally and the temperature data is correctly acquired, but in fact the collected temperature is not the real temperature.

Analysing the metrics collected by the monitoring tool (execution time, energy counter, MCU cycles), Figure 6.2 shows that all the metrics diverge from the normal behaviour. When compared with the normal behaviour, the number of cycles executed by the CPU increases by 52% (μnormal=61160, μanomaly=92933), the time spent to execute the function increases by 50% (μnormal=7.18, μanomaly=10.75), and lastly, the energy spent increases by 57% (μnormal=61160, μanomaly=92933). The increase in the execution time and CPU load shows that a larger code routine was executed by the CPU, when



```
0x017DD6--  void maliciousFunction(){ ③
                ...
                tempData = fake_sensor_value
                ...
            }
0x013604--  void getSensorData(){
                ...
                char evilstr[50]=”….d67d0100..”
                char str[15]=””
                tempData = spi.getData();
                strcpy(str,evilstr);②
                ...
            }
            void sendTemperature(){
                ...
                getSensorData(); ①
0x012F42--  ④ sendToRadio(tempData);
                ...
            }
```

Figure 6.1: Buffer Overflow stack-based attack on MSP430F5 family

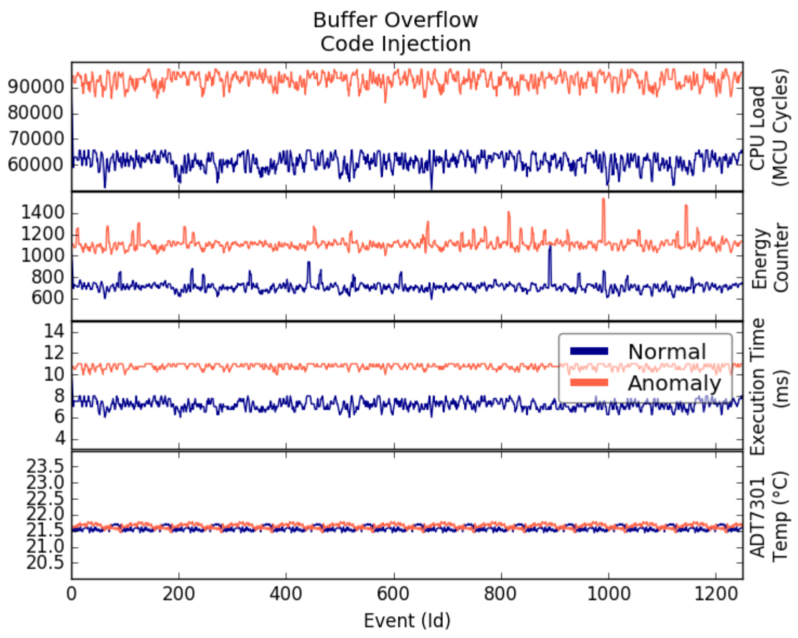


Figure 6.2: Firmware anomaly: stack-based buffer overflow attack

compared to the normal behaviour.

6.2.2 Hardware: Low-voltage Faults Analysis

When considering hardware fault injection on embedded devices, one of the most used non-invasive hardware injections is the power-glitch or VCC-glitch (see section 3.4.11 to see energy supply faults). As presented in several studies, abrupt changes in the voltage signal can be used to skip certain instructions and manipulate the execution flow, due to CMOS gates that are vulnerable to negative spikes [Gomina et al., 2014]. If the CPU has a power glitch, the signals will not reach the registers or paths properly. Low-voltage vulnerabilities can be explored both in the Harvard and in the Von Neumann architectures (e.g., ATMega, PIC and MSP microcontrollers). From skipping authentication routines, to accessing protected registers, this type of attack can be used to collect important information inside the nodes' memory or to manipulate the program flow.

Additionally, also related with low-voltage faults, sensor nodes consist of several hardware components with different characteristics, and different operation requirements, like the VCC range levels in which the hardware component operates. When a component operates below the minimum VCC, its operation can be faulty and unreliable. An example of this behaviour can be found in Figure 6.3, where the temperature sensor ADT7301 when operating a 2.2V measures 15ºC instead of 22ºC. In the used testbed, all of the components operate in the different VCC ranges. The WirelessHART radio operates between 2.1V and 3.76V, the ADT7301 operates between 2.7V and 5.25V, and the MSP430F5229 operates between 1.8V to 3.6V. Thus, in the case of the testbed, the node will

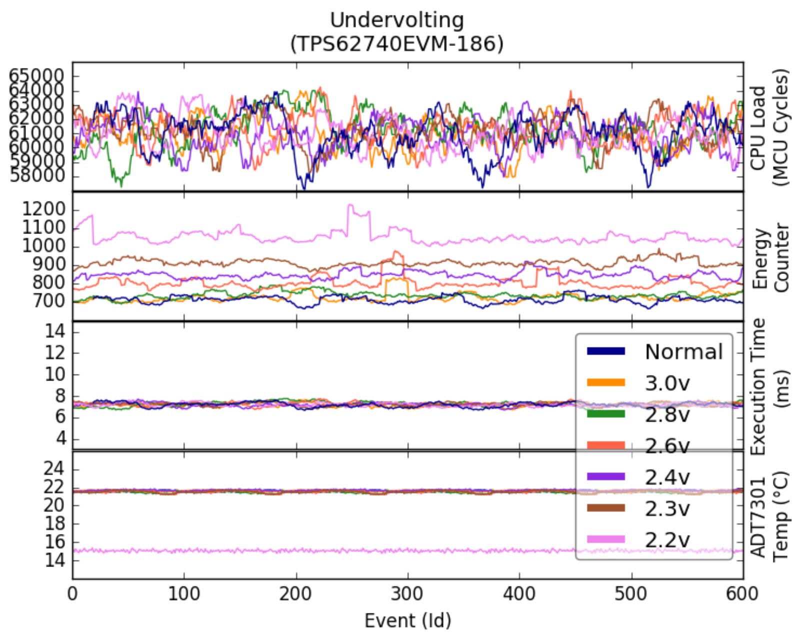


Figure 6.3: Hardware based anomaly: undervolting anomaly

only lose connection with the network if the WirelessHART radio operates below 2.1V. Until then, wrong temperature readings will be sent to the ICS.

In this scenario, several VCC levels were configured in the TPS62740EVM-186 boost converter using the appropriate jumper scheme, in order to understand the impact of a low-voltage scenario on the temperature sensor readings and on the collected metrics, as shown in the Figure 6.3.

From the normal 3.3V VCC level, to the lowest 2.2V, the energy counter increases by 43% (μnormal=710; μanomaly(2.2V)=1019). The increase in this metric is associated with the increase in the current consumed by the MSP430. Specifically, this microprocessor has a dedicated low-dropout voltage regulator, LDO, to power the core. By reducing the input VCC, the efficiency of the internal LDO will be lower, consuming more energy to guarantee the same VCORE.

6.2.3 Hardware: SPI Faults

Serial Peripheral Interface (SPI) uses a master-slave paradigm, and can be found in embedded systems to communicate between chips like RAM memory, or other components like sensors. The main advantages of this type of communication are its simplicity and high bandwidth, when compared to other technologies, such as UART. An SPI data bus has Clock (CLK), Master Input Slave Output (MISO), Master Output Slave Input (MOSI) and Slave Select (SS) data lines. Communication starts when the master device pulls the SS line down, waking up the slave device. Then, the master issues a train of clock pulses, and will continue to do so until the end of the communication. The master starts the communication by sending a command to the slave, using the MOSI line. In turn, the slave answers to the master through the MISO line.

In the conversation between a master device and slaves connected to the bus there are some faults that may occur (see section 3.4.11 to see communication faults). In terms of clock speed, if one part talks too fast, and the other cannot keep, some communication faults will occur. Additionally, if for some reason a stuck-at-0 or stuck-at-1 fault occurs in one of the SS or clock lines, communication between the two devices will not be possible. Specifically, if a stuck-at-0 fault occurs in the SS line, the slave device will never sleep. Otherwise, it will never wake-up. On the other hand, if the stuck-at-0 occurs in the clock line, the communication between two nodes will also not occur, because the master needs to continuously toggle the clock until the slave finishes the communication.

In terms of security, the SPI data buses usually suffer from sniffing attacks. Attackers can use, for instance, logic analysers to sniff the communication between the CPU and external memories, and gain access important data.

In our injection scenario, two types of anomalies were injected in the testbed: a stuck-at-0 fault related with the clock line, and a VCC fault. The results of the experiment are presented in Figure 6.4. Here, we use the same function than before, that collects the sensor data and sends it to the WirelessHART radio. As presented in the graph, there are major variations in all the collected metrics: execution time, energy counter, and CPU load. All the metrics present a sort of variation in relation to the normal behaviour, due to the processing of the different events related with the SPI communication (e.g., hardware interruptions, etc). Under normal operation, the CPU load has a mean value of μnormal=61149 with σnormal=3776, the energy counter has μnormal=710.28 with σnormal=65.9, and the execution time has μnormal=7.17 with σnormal=0.56.

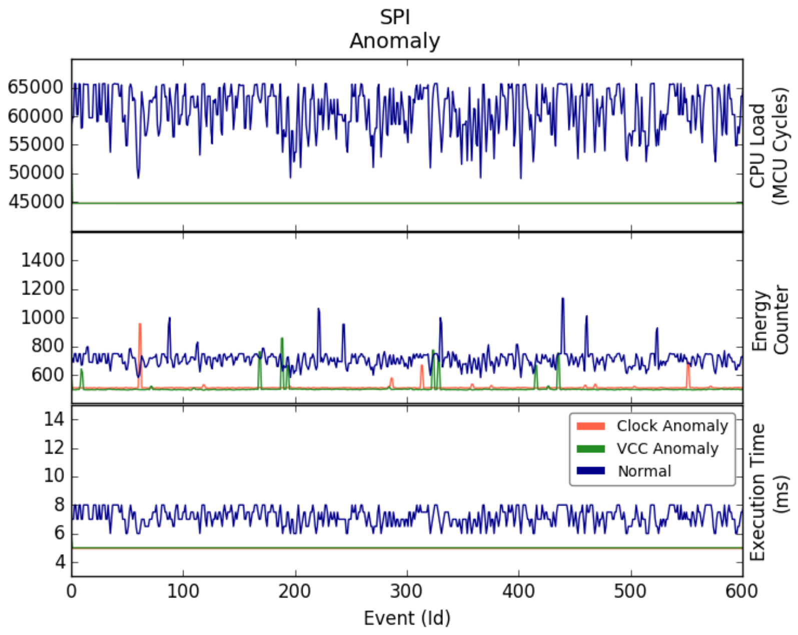


Figure 6.4: Hardware based anomaly: SPI VCC and Clock anomaly

On the other hand, in the case of the VCC anomaly, the CPU is not able to communicate with the sensor and it executes less instructions, spending less time on the function. Moreover, as the sensor is disconnected, less energy is spent. In the case of the clock anomaly, the sensor spends more energy, but it is not able to communicate with the CPU (VCC anomaly: μnormal=503.57; Clock anomaly: μnormal=511.8).

6.2.4 Hardware: High Temperature Faults

When deployed in industrial environments, both monitored and monitoring systems may be subject to harsh environmental conditions, such as high temperature (see section 3.4.3, to see phenomenological faults). As presented in [Hutter and Schmidt, 2014; Stanislowski et al., 2014], abrupt variations in ambient temperature can lead to serious effects in radio synchronization, and can also be used to perform security attacks to embedded devices. In [Stanislowski et al., 2014] the authors study the effects of clock drifts under high temperatures, in synchronized networks that use the IEEE 802.15.4e standard, similar to WirelessHART. Such faults can result in communication losses or extra energy wasted in synchronization. Additionally, the use of abrupt temperature variations has been also a subject of study from the beginning of embedded systems [Hutter and Schmidt, 2014]. Attack strategies range from the use of active temperature attacks that freeze SRAM memories to recover their content, to high-temperature attacks that lead to memory errors and allow intruders to inject faults on the systems.

In this experiment, a high temperature fault was injected directly in the microcontroller core, using a heat gun, in order to simulate the condition of a system

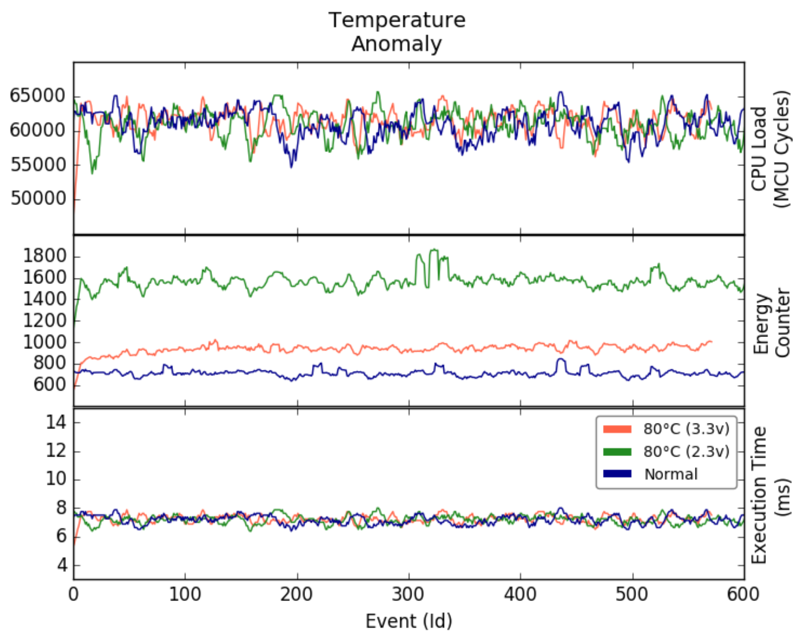


Figure 6.5: Hardware based anomaly: Temperature anomaly

under high temperature. The MSP430F5229 microcontroller can operate in a free-air range between -40ºC to 85ºC, and a storage memory between -55ºC to 150ºC. In the experiment, the surface temperature of the core die was maintained during 1 hour to its maximum (85ºC). As in the previous experiments, the execution time, the energy counter, and the CPU load metrics were collected for the same function. Here, the temperature sensor was put far away from the heating point, in order not to affect the ambient temperature collected by the ADT7301 sensor.

As can be seen in Figure 6.5, there are some major variations in the energy counter when the microcontroller operates under high temperature (μnormal=710.54; μanomaly(3.3V)=933.58), as expected. In [Borgeson et al., 2012], the authors claim that temperature is one of the most overlooked features in systems design, and in some cases the current can increase 10 to 15 times under extreme temperatures. This is similar to what happens during a low-voltage state (2.3V) (see Figure 4 6.3). On the other hand, the CPU load and execution time metrics do not show any evidence of errors when compared the normal case. As mentioned in [Hutter and Schmidt, 2014], memory errors and faults are only expected to occur when the storage temperature exceeds 150ºC.

6.3 Detecting Anomalies And Security Threats

As seen in the previous sections, industrial wireless sensor networks are not immune to security threats and faults, which points to the need of Intrusion Detection Systems (IDS), anomaly detection, and monitoring tools. In this section, experimental results from three anomaly detection approaches will be presented, with the aim of demonstrating that the proposed monitoring approach is suitable for building tools capable of detecting firmware and hardware faults. The selected anomaly detection approaches were the following: a linear-based model (the One-Class Support Vector Machine, OCSVM); a proximity-based model (the k-Nearest Neighbours, kNN); and a neural-network based model (AutoEncoder). These approaches are presented in subsection 6.3.1. Subsection 6.3.2 discusses the obtained results.

6.3.1 Data Splitting Strategy and Classifiers

In machine learning, the selection of the algorithm and the splitting strategy of the data depend on the type of available data and on the approach to follow in order to solve the specific problem. There are two main approaches: supervised learning, used when the available data is labelled (the classifier is trained for all the classes); and unsupervised learning, used when the data is not labelled (similar to the approach followed in chapter 5). In the latter approach, grouping techniques like clustering are used for classifying data in a specific group or class, using distance measures.

Anomaly detection or outlier detection is a specific field of machine learning that refers to the problem of finding data that is not conformant with an expected behaviour (e.g., anomalies, outliers, exceptions, etc). In this classification problem, typically only the normal class is available. The approaches used for this classification are unsupervised learning, when no label is known; and semi-supervised learning, when only the normal data is labelled. Using a semi-supervised approach, it is possible to train a binary classifier that infers the normal system behaviour and defines the boundaries of the normal region. All the events that lie inside this region will be considered normal events. Otherwise, they will be classified as anomaly. Despite the fact that in the experiment all the labels were known, when building a solution for monitoring the network in real-time such thing is not possible. Thus, a semi-supervised approach was used.

Table I presents of overview of the splitting strategy of the data collected from the experiment, and Figure 6.6 the procedure to transform the data and to create the machine learning models for each classifier (OCSVM, kNN, AutoEncoder).

The training process (Figure 6.6) starts by the pre-processing step. In this step, all the data labels are transformed into a binary problem (0,1). Next, all the used features (CPU load, execution time, energy counter and the function ID) are standardized (μ=0 and σ=1), before starting the training of the classifiers. Each distribution is stored, in order to be used in the prediction phase, when new data arrives. The next phase is the generation of the machine learning model, and it is divided into three steps: the training step (1), the validation step (2), and the test step (3). In the training step (1), the training dataset will be used to train the model of each classifier. Then, in the validation step (2),

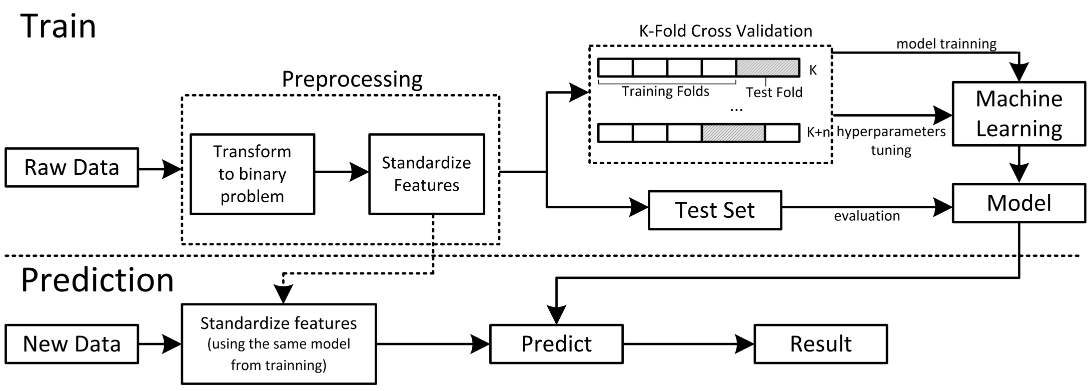


Figure 6.6: Machine Learning strategy and process

Step	Description	SPI	Undervolting	Temperature	Firmware
Train	80% normal events (n)	(n) 17274			(Not used)
Validation	50% abnormal events (a)	(a) 675	(a) 2297	(a) 620	
Test	20% normal events (n)	(n) 4318			
	50% abnormal events (a)	(a) 675	(a) 2296	(a) 620	(a) 625

Table 6.1: Splitting strategy in training, validation, and test phases

the model will be evaluated used the validation dataset. To ensure less bias, and to test how the model handles new data, the k-fold cross validation technique was used (k=5), to tune the model hyperparameters. The best hyperparameters will be selected based on the best f1 score mean (k=5). Finally, in the test step (3), the classifier will be tested using the performance metrics presented in table 6.2.

Table 6.1 shows how the data was divided in each phase. In the training step, 80% of normal events were randomly selected for training and validation (17274 events). 50% of anomalies were used in the validation phase, in order to evaluate the classifier (SPI:675, Undervolting:2297, Temperature:620). The test used the remaining 20% of normal events (4318 events) and 50% of anomalies (SPI:675, Undervolting:2296, Temperature:620) were used for the test phase. The firmware data (625 events) was intentionally not used in the training and validation phases, in order to show how the classifiers handle different anomalies not used in the validation step. The model was then saved and ready to use in the prediction phase. The process was developed using Python, with the scikit-learn framework.

Lastly, using a point anomaly approach, three distinct classification approaches were selected: the OCSVM, the kNN, and the AutoEncoder. OCSVM is one of the most used classifiers in anomaly detection. When used in a semi-supervised approach, like the one followed in this article, the classifier creates a model for the normal data, using the normal data training dataset only. To create the model, several hyperparameters need to be configured. To select the most appropriate parameters, a grid-search was conducted, in the k-fold cross validation step with the parameters ν and γ. The ν parameter controls the percentage of anomalies in the normal dataset. Additionally, the γ parameter defines how far the influence of a single training example reaches (low means far, and high means close). All the remaining parameters were used in the default mode. Differently from OCSVM, the kNN classifier is a nearest-neighbour-based technique, which uses distance metrics to compute the distance of a specific data point to the kth nearest neighbour as the anomaly score (the classifier uses one of the simplest approaches in the anomaly detection field). A grid-search was performed in order to select the best k. Lastly, a neural network classifier was used, namely AutoEncoder, which is a specific type of multi-layer neural network that performs hierarchical and nonlinear dimensionality reduction of the data, using the same number of nodes at the input and output networks. The idea behind the use of this approach in anomaly detection is to train the output to reconstruct the input, using the normal data, as closely as possible. Thus, if the network is able to reproduce the normal data, when an anomaly occurs in the system the network will be not able to reproduce it, and the resulting error will be large. To train the classifier, a grid-search was also conducted, with different hidden layers and epochs.

6.3.2 Results

Table 6.2 presents the results of the performance metrics for the used classifiers, divided by type of anomaly (firmware or hardware). The classifier that yielded the best results is OCSVM, with a f1=0.95 for hardware anomalies and f1=0.857 for firmware, also showing that it is capable of detecting anomalies that were not used in the validation phase. Two important metrics are the false positive ratio (FPR) and the recall (or true positive ratio (TPR)). On one hand, the FPR tells us how many normal events were considered as anomalies. On the other hand, the recall metric tell us how well we identify all the anomalies. In the case of OCSVM, this was only 5%. Only in the undervolting case, the classifier missed

Classifiers	Dataset		F1	Precision	Recall	FPR	TP	TN	FP	FN	Performance Metrics
OCSVM (ν=0.05 ϒ=1000)	Hardware	All	0.95	0.938	0.961	0.05	3453	4091	228	139	
		SPI	0.853	0.743	1	0.05	659	4091	228	0	
		Undervolting	0.922	0.905	0.94	0.05	2174	4091	228	139	
		Temperature	0.743	0.591	1	0.05	329	4091	228	0	$F1 = 2.\frac{precision \times recall}{precision + recall}$
	Firmware		0.857	0.749	1	0.05	625	4110	209	0	
kNN (k=1)	Hardware	All	0.947	0.914	0.982	0.08	3526	3988	331	66	$precision = \frac{TP}{TP + FP}$
		SPI	0.799	0.666	1	0.08	659	3988	331	0	
		Undervolting	0.919	0.872	0.973	0.08	2250	3988	331	63	$recall = \frac{TP}{TP + FN}$
		Temperature	0.663	0.497	0.994	0.08	327	3988	331	2	
	Firmware		0.81	0.681	1	0.07	625	4026	293	0	$FPR = \frac{FP}{FP + TN}$
AutoEncoder (epoch=20 hl=[3,2,2,3])	Hardware	All	0.739	0.642	0.87	0.4	3124	2575	1744	468	
		SPI	0.43	0.274	1	0.4	659	2575	1744	0	$TPR = \frac{TP}{TP + FN}$
		Undervolting	0.626	0.514	0.799	0.4	1847	2575	1744	466	
		Temperature	0.274	0.159	1	0.4	329	2575	1744	0	
	Firmware		0.417	0.263	1	0.4	625	2569	1750	0	

Table 6.2: Classifier results for hardware and firmware anomalies

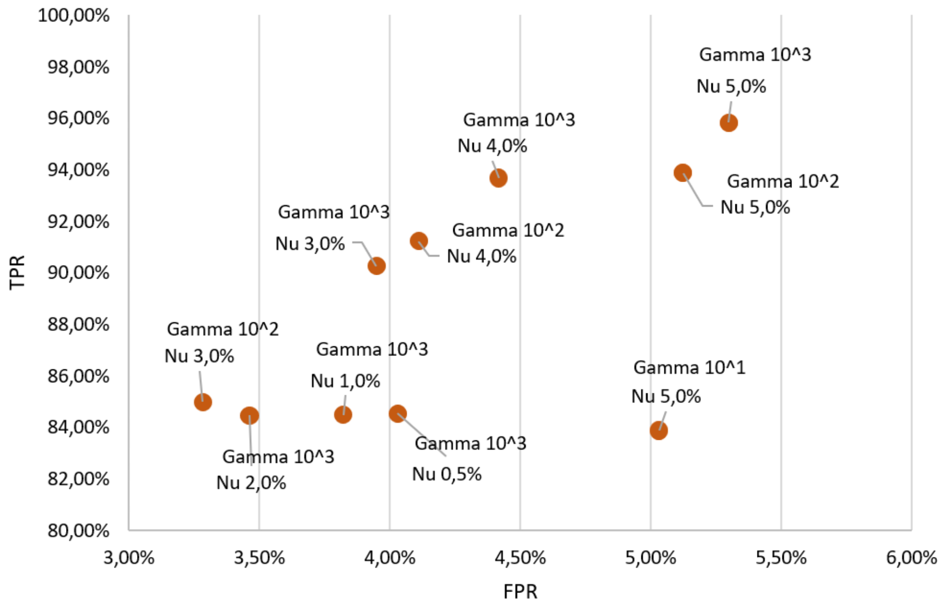


Figure 6.7: OCSVM TPR/FPR with different parameters

some anomalies, that were wrongly classify as normal events. With also good results, the kNN classifier was the second classifier with the best results. The classifier performed similarly to OCSVM, however with higher FPR (8%). In terms of recall, kNN performed even better than OCSVM, in the classification of undervolting faults. Lastly, the performance of AutoEnconder was weak, in part due to the low dimensionality of the data, limiting the hidden layers that can be used. Here, another approach clearly needs to be followed in the future, e.g., using a sliding window to increase the dimensionality of the data. The OCSVM TPR/FPR results for different values of ν and γ are presented in figure 6.7.

6.4 Summary of the Chapter

As part of an ongoing work that proves the effectiveness of the proposed architecture, the work presented in this chapter demonstrated that it is possible to detect representative hardware and firmware faults, and also security threats, in industrial scenarios, using the defined monitoring strategy. In the scenario explored in this chapter, we have shown that the OCSVM and kNN classifiers are capable of detecting significant injected faults, with very low false positive ratios and good recall.

Chapter 7

Conclusions and Future Work

"A conclusion is the place where you got tired thinking."

(Martin H. Fischer)

Contents

The industrial field is experiencing a great transformation, in terms of technologies, applications and threats. Current technologies keep increasing their processing and memory capabilities, improving the low power consumption, but with an important drawback, its complexity and fault prone nature. At the same time, with the demand of CPS applications, new hardware starts to incorporate specific AI components that confer micro-intelligence to IoT devices, instead of relying on centralized models. As presented, hardware manufacturers like ARM, operating systems like FreeRTOS, and other key-manufacturers start to understand the need to incorporate monitoring tools in their platforms. Thus, it is expected in the following years new types of monitoring tools for these systems, like the one proposed in this thesis. This chapter summarizes and resumes all the contributions of the work performed. Additionally, at the end, a forward-looking perspective will be conducted, by presenting some future work related with the proposed architecture.

7.1 Synthesis of the Thesis

Industrial wireless technologies like ZigBee, ISA100.11a, WIA-PA, and WirelessHART were designed to fulfil the low latency requirements needed by industrial systems. Despite the growing use of these technologies in such environments, there is a lack of monitoring tools capable to identify and prevent faults, lowering the availability of these systems. Such shortage happens due to an important fact: current operating systems and hardware architectures are fragmented, making it difficult to build a general monitoring model to the four main standards. In order to propose such monitor architecture, chapter 1 started by identifying these issues, addressing the research challenges that need to be fulfilled by a new proposal.

The state of the art was presented in chapter 2, as general background. Industrial applications requirements were revised, in detail, along with legacy wired technologies, IWSN standards, IoT management standards, current diagnostic tools and techniques, finishing with the efforts made by the industrial and academic community to propose similar solutions. The chapter presented the technology journey of industrial systems and monitoring tools, culminating in today's efforts and proposals. By detailing the technologies, techniques and some of available solutions, chapter 2 sustains the proposed model. Not less important, chapter 2 presents other small contributions. For instance, the network metrics shared by these standards for routing purpose are presented and described. This is one of the contributions of this thesis, that cannot be found, as far as known, in the most relevant literature.

Monitoring tools are used in order to increase the MTTF of industrial systems, and to prevent security threats, combined with IDS. So, it was important to provide an appropriate characterization of fault nature in WSNs, that until now was missed in the literature. Chapter 3 presented a WSN fault taxonomy, that not only characterize, WSN faults, but also the impact of such faults on sensor nodes components and systems. Examples of faults were given in each taxonomy viewpoint, to make it clear. In addition, this chapter specified the definition used for the concepts of fault, error, failure and anomaly, used in the next chapters. Such contribution was important to chapter 5 and 6, in the selection of faults, when fault injection was used to test the effectiveness of the architecture.

The main contribution of this thesis was presented in chapter 4, when the monitoring model was proposed. The architectural model consists of several agents that collect in-node metrics and network metrics, sharing these metrics to a central point, the industrial gateway. During this chapter, all these agents were described, and some monitoring techniques recommended for each one. The network monitoring relies on the available routing metrics described in chapter 2. The firmware monitoring is performed by using instrumentation techniques. Lastly, the hardware state metrics are collected using techniques that can be implemented in most commonly used hardware architectures (also described in chapter 2). At the end, all the monitoring information is shared at the industrial gateway using existing management protocols (e.g., LWM2M, COMI or RESTCONF). To prove the low impact of the monitoring model, a testbed using the WirelessHART standard was built and the impact of the architecture assessed, by measuring: the overhead of each metric in terms of energy consumption; the memory, processing, and latency overhead added to the main program; the network overhead; and lastly, the latency introduced by the instrumentation code in the main program. The results showed that the proposed model can be used in industrial scenarios, without compromising the sensor node lifetime and the main application execution.

After evaluating the proposed model, chapter 5 presented the first use case, by using the model in the detection of faults and security attacks to the network. To test the effectiveness of the monitoring model, several faults were injected in the WirelessHART testbed using a security research toolkit for IEEE802.15.4. Thus, chapter 5 started by making an in depth analysis of the WirelessHART standard from a security perspective. Several experiments were conducted using the Killerbee research toolkit to test the security of the WirelessHART standard. Through the analysis of the standard and experimentation, an important attack vector was identified, and tested using this testbed. A new, and not known attack to the standard was identified, allowing an attacker to conduct an exhaustion attack, by manipulating the information of the advertisement messages. Such discovery, represents an additionally contribution of this chapter, and was used to show the importance of the in-node metrics in the detection. Finally, a detection mechanism using the OCSVM classifier was built. The evaluation of this classifier showed that it is possible to detect specific attacks and faults like interferences or jamming with good results, using the sensor in-node and the network metrics shared by the standard.

The second and last use case evaluated the effectiveness of the model to detect faults and security attacks in the firmware and hardware of the sensor nodes. Chapter 6 presented a series of faults and security attacks that can be conducted to the existing IoT technologies. A stack-based overflow attack was conducted against the main program, and the impact of this attack was evaluated using the metrics collected by the monitoring testbed. Additionally, from the hardware perspective, several faults were injected: a low-voltage fault, an SPI fault, and a high temperature fault. The impact of each fault was also described and compared with the normal behaviour of the sensor node. At the end, similarly to chapter 5, a detection mechanism was also proposed, by comparing several anomaly detection classifiers: OCSVM, kNN, and the AutoEncoder. The classifier was built using a semi-supervised approach, allowing the detection mechanism to identify new attacks that were not used in the training phase. The results show that the mechanism is able to detect with high recall and false positive ratio the injected anomalies when using the OCSVM and the K-NN classifiers, proving that the proposed model can be used to detect network/hardware/firmware faults and attacks in industrial deployments.

7.2 Future Work

Throughout this thesis, several contributions were made, a new architecture was proposed and evaluated, and several fault detection mechanisms developed, improving the current state of the art in the domain of industrial IoT management and anomaly detection. Nevertheless, the work performed in this thesis opens more research questions that need to be answered in a near future, as well as new insights to some other contributions, that could not be answered during the entire process of producing this thesis. Complementary to the various contributions of this thesis, several research questions arise. Thus, this section is organized in two topics. The first topic presents some new insights and minor contributions that can be added to this thesis in the future. On the other hand, the second topic presents a more futuristic perspective, where the impact of new technologies in the current architecture will be discussed.

As future work, the monitoring architecture should be further explored by extending and refining the existing implementation. Although Chapter 4 provides a detailed explanation of the experiments conducted using WirelessHART and explains how each agent can be adapted to the other relevant technologies, it was not possible to develop a testbed that incorporated the four standards at once, in the available time frame, in order to show that the proposed architecture can seamlessly monitor equipment complying with each and all of the four standards. Additionally, the proposed architecture and agents were designed to be compatible with several firmware architectures (see chapter 4). Apart from the existing operating system fragmentation, another contribution to the following proposal would be the implementation of these agents in operating systems like FreeRTOS, Contiki and RiOT. Such contribution would allow showing that the proposed model is general and can be deployed in a large variety of IoT devices, with increased potential for its adoption by developers. For instance, in FreeRTOS, such agents could be used and incorporated in the TraceAlyzer tool, by using the new public Software Development Kit (SDK). The synchronization between in-node metrics is another feature that is missing in the presented testbed. Chapter 4 presents the industrial standards that can support this feature. Using this additional metric, faults associated with all the network or clusters could be more easily identified. More advance instrumentation techniques, like those proposed in [Schuster et al., 2014; Dong et al., 2013; Shea et al., 2009], can also be tested, reducing the code instrumentation time. Better log compression techniques can also be explored, in order to reduce the overhead created over the network. The current approach does not explore any type of log compression. In what concerns hardware architectures, the developed testbed can also be extended to incorporate ARM microprocessors with the CoreSight technology. As presented in chapters 2 and 4, this technology provides a rich set of monitoring metrics related with hardware and firmware condition, that could help in the identification of more advanced anomalies. Lastly, in chapters 5 and 6 a dataset was collected, containing the normal state of the system, along with the time period of network and in-node anomalies injection. The anomaly detection mechanisms were built using a semi-supervised approach with classifiers like OCSVM, kNN and AutoEnconder. By using the same dataset, other machine learning techniques and approaches can be used and compared with the ones used in this thesis. For instance, Hidden-Markov chains can be tested to model the states of the code during normal behaviour, and detecting anomalies when an uncommon jump occurs. Lastly, deep learning techniques can also be explored.

Cybersecurity is nowadays one of the most important topics in the industrial field. The use of ICT technology, such as TCP/IP protocols, exposes these systems to new security threats. In the near future, security will certainly be the feature that will contribute the most to the adoption of monitoring tools, IDS, and machine learning in industrial systems. At the same time, the introduction of more complex and capable hardware, with specific AI components, are contributing to the CPS concept on which Industry 4.0 is based. Moreover, the introduction of such components in small microprocessors will also create an opportunity to have monitoring systems capable of detecting anomalies in sensor nodes, or even more advanced methods (e.g., sensor node neighbours computing metrics for regions or sets of nodes). An example of such platforms is the Apollo 3 board [Allan, 2019], developed by Google and Ambiq, that is capable to provide edge computing by running TensoFlow lite. By giving such capabilities to sensor nodes, it will be possible to create more distributed processing algorithms, and at the same time reduce the overhead of sharing this information over the network. The enhancement of these new capabilities will be followed by an even greater reduction of energy consumption in the sensor nodes, by using Bulk Acoustic Wave(BAW) resonators technology, for instance, instead of quartz crystals technology [Yoshida, 2019]. All of these advances in technologies will allow to have more state metrics inside the sensor nodes, of which the CoreSight technology is an example. Future operating systems will incorporate these technologies as interfaces, helping in the development of monitoring architectures that can be easily integrated in industrial applications. Other important innovation is the appearance of new wireless standards, that rely on open technologies like 6TiSCH. The standard is based in more common technologies from the IoT domain, maintaining at the same time the deterministic nature and reliability. Last but not least, there is a remaining issue in the state of the art of this topic, that should be addressed as future work. In chapter 5, when attacks to the WirelessHART were conducted, the Killerbee toolkit was used, but without synchronization capabilities. In order to prevent more sophisticate attacks to these networks, tools that stay synchronized with the network are needed, in order to better assess the security of industrial wireless standards.

Bibliography

Alcaraz, C. and Lopez, J. (2010). A Security Analysis for Wireless Sensor Mesh Networks in Highly Critical Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(4):419–428.

Alena, R., Gilstrap, R., Baldwin, J., Stone, T., and Wilson, P. (2011). Fault tolerance in ZigBee wireless sensor networks. In *2011 Aerospace Conference*.

Ali, A. and Tixeuil, S. (2010). Advanced Faults Patterns for WSN Dependability Benchmarking. *MSWIM'10 - 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*, page 39.

Allan, A. (2019). Introducing the SparkFun Edge.

Alliance, Z. (2015). Zigbee Specification. *Zigbee Alliance website*, pages 1–542.

Arm (2017). CoreSight Debug and Trace. `http://www.arm.com/products/system-ip/coresight-debug-trace`.

Avižienis, A., Laprie, J. C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*.

Baccelli, E., Hahm, O., Günes, M., Wählisch, M., and Schmidt, T. (2013). RIOT OS: Towards an OS for the Internet of Things. In *The 32nd IEEE International Conference on Computer Communications (INFOCOM 2013)*, Turin, Italy.

Barry, R. (2018). FreeRTOS, a free open source RTOS for small embedded real time systems. `http://www.freertos.org`.

Bayou, L., Espes, D., Cuppens, N., Bayou, L., Espes, D., Cuppens, N., Cuppens-Boulahia, N., and Cuppens, F. (2016). Security analysis of WirelessHART communication scheme. In *International Symposium on Foundations and Practice of Security*, pages 223–238. Springer.

Bhadriraju, A. R., Bhaumik, S., Lohith, Y. S., Brinda, M. C., Anand, S. V. R., and Hegde, M. (2012). 6PANview: Application performance conscious network monitoring for 6LoWPAN based WSNs. *2012 National Conference on Communications, NCC 2012*.

Bierman, A., Bjorklund, M., and Watsen, K. (2017). Restconf protocol. RFC 8040, IETF. `https://tools.ietf.org/html/rfc8040`.

Birolini, A. (1999). Quality and reliability of technical systems. *Reliability, IEEE Transactions on*, 48(2):205–206.

Bjorklund, M. (2010). Yang - a data modeling language for the network configuration protocol (netconf). RFC 6020, IETF. `https://tools.ietf.org/html/rfc6020`.

Borgeson, J., Schauer, S., and Diewald, H. (2012). Benchmarking MCU power consumption for ultra-low-power applications. *Texas Instrum.*, page 8.

Bormann, C., Ersue, M., and Keranen, A. (2014). Terminology for constrained-node networks. RFC 7228, IETF. `https://tools.ietf.org/html/rfc7228`.

Bosch, R. (1991). CAN Specification Version 2.0. *Rober Bousch GmbH, Postfach*, 300240:72.

Cao, Q., Abdelzaher, T., Stankovic, J., Whitehouse, K., and Luo, L. (2008). Declarative tracepoints: a programmable and application independent debugging system for wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 85–98. ACM.

Case, J. D., Fedor, M., Schoffstall, M. L., and Davin, J. R. (1990). Simple network management protocol (snmp). STD 15, IETF. `https://tools.ietf.org/html/rfc1157`.

Chandola, V., Banerjee, A., and Kumar, V. (2012). Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839.

Chang, T., Tuset-Peiro, P., Vilajosana, X., and Watteyne, T. (2016). OpenWSN amp; OpenMote: Demo'ing a Complete Ecosystem for the Industrial Internet of Things. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–3.

Chang, W. G. and Lin, F. J. (2016). Challenges of incorporating OMA LWM2M gateway in M2M standard architecture. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6.

Charles, J. (2018). Killerbee. `https://github.com/riverloopsec/killerbee`.

Chaturvedi, S. K. (2016). *Network Reliability: Measures and Evaluation*. Performability Engineering Series. Wiley.

Chen, B.-r., Peterson, G., Mainland, G., and Welsh, M. (2008). LiveNet: Using Passive Monitoring to Reconstruct Sensor Network Dynamics. In Nikoletseas, S. E., Chlebus, B. S., Johnson, D. B., and Krishnamachari, B., editors, *Distributed Computing in Sensor Systems: 4th IEEE International Conference, DCOSS 2008 Santorini Island, Greece, June 11-14, 2008 Proceedings*, pages 79–98. Springer Berlin Heidelberg, Berlin, Heidelberg.

Chouikhi, S., El Korbi, I., Ghamri-Doudane, Y., and Azouz Saidane, L. (2015). A survey on fault tolerance in small and large scale wireless sensor networks. *Comput. Commun.*, 69:22–37.

Cinque, M., Cotroneo, D., Marno, C. D., Russo, S., Testa, A., Barrenetxea, G., Ingelrest, F., Schaefer, G., and Vetterli, M. (2009). Avr-inject: a tool for injecting faults in wireless sensor networks. In *IEEE international Symposium on parallel and distributed processing (IPDPS)*, volume 1, pages 1–8.

Coronato, A. and Testa, A. (2013). Approaches of Wireless Sensor Network dependability assessment. In *2013 Federated Conference on Computer Science and Information Systems*.

de Souza, L. M. S., Vogt, H., and Beigl, M. (2007). A survey on fault tolerance in wireless sensor networks. *Sap research, braunschweig, germany*, pages 168–173.

der Stok, P. V., Bierman, A., Veillette, M., and Pelov, A. (2017). CoAP Management Interface. Technical Report draft-ietf-core-comi-00, Internet Engineering Task Force.

Do, V. L., Fillatre, L., Nikiforov, I., and Willett, P. (2017). Feature article: security of SCADA systems against cyber–physical attacks. *IEEE Aerosp. Electron. Syst. Mag.*, 32(5):28–45.

Dong, W., Chen, C., Bu, J., Liu, X., and Liu, Y. (2013). D2: Anomaly detection and diagnosis in networked embedded systems by program profiling and symptom mining. *Proceedings - Real-Time Systems Symposium*, pages 202–211.

Dong, W., Huang, C., Wang, J., Chen, C., and Bu, J. (2014). Dynamic logging with Dylog for networked embedded systems. *2014 11th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2014*, pages 381–389.

Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 455–462, Washington, DC, USA. IEEE Computer Society.

Dutta, P., Feldmeier, M., Paradiso, J., and Culler, D. (2008). Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proceedings - 2008 International Conference on Information Processing in Sensor Networks, IPSN 2008*, pages 283–294.

Eclipse (2017). Eclipse Leshan. `https://github.com/eclipse/leshan`.

Enns, R., Bjorklund, M., Schoenwaelder, J., and Bierman, A. (2011). Network configuration protocol (netconf). RFC 6241, IETF. `https://tools.ietf.org/html/rfc6241`.

Evans, D. (2011). The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. *Cisco Internet Bus. Solut. Gr.*, (April).

Fairbairn, M. L., Bate, I., and Stankovic, J. A. (2013). Improving the dependability of sensornets. *Proceedings - IEEE International Conference on Distributed Computing in Sensor Systems, DCoSS 2013*, pages 274–282.

FieldComm (2014). HART Technology Detail. https://fieldcommgroup.org/technologies/hart/hart-technology-detail.

Foglietta, C., Masucci, D., Palazzo, C., Santini, R., Panzieri, S., Rosa, L., Cruz, T., and Lev, L. (2018). From Detecting Cyber-Attacks to Mitigating Risk Within a Hybrid Environment. *IEEE Syst. J.*, 13(1):424–435.

Fovino, I. N., Carcano, A., Masera, M., and Trom-Betta, A. (2009). Design and implementation of a secure Modbus protocol. In *IFIP Adv. Inf. Commun. Technol.*

Francillon, A. and Castelluccia, C. (2009). Code injection attacks on harvard-architecture devices. *Proceedings of the 15th ACM conference on Computer and communications security (CCS).*

Giannetsos, T., Dimitriou, T., Krontiris, I., and Prasad, N. R. (2010). Arbitrary code injection through self-propagating worms in von Neumann architecture devices. *Comput. J.*, 53(10):1576–1593.

Gomina, K., Rigaud, J. B., Gendrier, P., Candelier, P., and Tria, A. (2014). Power supply glitch attacks: Design and evaluation of detection circuits. *Proc. 2014 IEEE Int. Symp. Hardware-Oriented Secur. Trust. HOST 2014*, 1:136–141.

Granjal, J., Monteiro, E., and Silva, J. S. (2015). Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues. *IEEE Communications Surveys Tutorials*, 17(3):1294–1312.

Graveto, V., Rosa, L., Cruz, T., and Simões, P. (2019). A stealth monitoring mechanism for cyber-physical systems. *Int. J. Crit. Infrastruct. Prot.*, 24:126–143.

Grimaldi, S., Gidlund, M., Lennvall, T., and Barac, F. (2016). Detecting communication blackout in industrial Wireless Sensor Networks. *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, 2016-June.

Gungor, V. and Hancke, G. (2009). Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches. *IEEE Trans. Industrial Electronics*, 56(10):4258–4265.

Hahm, O., Baccelli, E., Petersen, H., and Tsiftes, N. (2016). Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(5):720–734.

Holenderski, M., Van Den Heuvel, M., Bril, R. J., and Lukkien, J. J. (2010). Grasp: Tracing, visualizing and measuring the behavior of real-time systems. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 37–42.

Hutter, M. and Schmidt, J. M. (2014). The temperature side channel and heating fault attacks. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 8419 LNCS:219–235.

IEC (2010). Industrial communication networks - Wireless communication network and communication profiles - WirelessHART, IEC 62591:2010. Technical report, International Electrotechnical Commission, Geneva, Switzerland.

IEC (2015). Industrial networks - Wireless communication network and communication profiles - WIA-PA, IEC 62601:2015. Technical report, International Electrotechnical Commission, Geneva, Switzerland.

IEEE (2002). IEEE 802.15.1-2002 IEEE Standard for information technology - Telecommunication and information exchange between systems - LAN/MAN - Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks.

IEEE (2006). Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs), IEEE802.15.4-2006. Technical report, Institute of Electrical and Electronics Engineers.

IEEE Std (1994). IEEE Standard Classification for Software Anomalies. *IEEE Std 1044-1993*, pages i–.

Instruments, T. (2017a). CC2430 Single-Chip 2.4GHz IEEE 802.15.4 Compliant and Zigbee Ready RF Transceiver. `http://www.ti.com/product/CC2420`.

Instruments, T. (2017b). CC2650 SimpleLink multi-standard 2.4 GHz ultra-low power wireless MCU. `http://www.ti.com/product/CC2650`.

Instruments, T. (2017c). Code Composer Studio (CCS) Integrated Development Environment (IDE). `http://www.ti.com/tool/ccstudio`.

Instruments, T. (2017d). MSP EnergyTrace Technology. `http://www.ti.com/tool/energytrace`.

Instruments, T. (2019). MSP Driver Library. http://www.ti.com/tool/MSP-DRIVERLIB.

ISA (1972). ISA50, Signal Compatibility of Electrical Instruments. `https://www.isa.org/isa50/`.

ISA (2009). Wireless systems for industrial automation: Process control and related applications, ANSI/ISA-100.11a-2009. Technical report, International Society of Automation.

Islam, K., Shen, W., Member, S., Wang, X., and Member, S. (2012). Wireless sensor network reliability and security in factory automation: A survey. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 42(6):1243–1256.

ISO/IEC 7498-4 (1989). *Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4 : Management Framework*. International Organization for Standardization.

Jalote, P. (1994). *Fault Tolerance in Distributed Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Jose Da Cunha, M., De Almeira, M. B., Fernandes, R. F., and Carrijo, R. S. (2017). Proposal for an IoT architecture in industrial processes. *2016 12th IEEE International Conference on Industry Applications, INDUSCON 2016*.

Kelava, M., Gavranić, I., and Deškin, J. (2008). Practical experience with inspection in plants at risk of explosive atmospheres. *5th PCIC Europe 2008: Petroleum and Chemical Industry Conference Europe*.

Khan, M. M. H., Le, H. K., Ahmadi, H., Abdelzaher, T. F., and Han, J. (2008). Dustminer: troubleshooting interactive complexity bugs in sensor networks. *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 99–112.

Khan, M. M. H., Luo, L., Huang, C., and Abdelzaher, T. (2007). SNTS: Sensor Network Troubleshooting Suite. In Aspnes, J., Scheideler, C., Arora, A., and Madden, S., editors, *Distributed Computing in Sensor Systems: Third IEEE International Conference, DCOSS 2007, Santa Fe, NM, USA, June 18-20, 2007. Proceedings*, pages 142–157. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kim, A. N., Hekland, F., Petersen, S., and Doyle, P. (2008). When HART goes wireless: Understanding and implementing the WirelessHART standard. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 899–907.

Kim, H., Kim, S., Kwon, S., Jo, W., and Shon, T. (2019). *A Novel Security Framework for Industrial IoT Based on ISA 100.11a*. Springer International Publishing.

Kim, S., Lim, H., Lim, S. M., and Shin, I. H. (2018). Study on cyber security assessment for wireless network at nuclear facilities. *6th Int. Symp. Digit. Forensic Secur. ISDFS 2018 - Proceeding*, 2018-Janua:1–5.

Knowles, W., Prince, D., Hutchison, D., Disso, J. F. P., and Jones, K. (2015). A survey of cyber security management in industrial control systems. *Int. J. Crit. Infrastruct. Prot.*, 9:52–80.

Kobayashi, K. (2015). LAWIN: A Latency-AWare InterNet architecture for latency support on best-effort networks. In *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8.

Koushanfar, F., Potkonjak, M., and Sangiovanni-Vincentelli, A. (2003). On-line Fault Detection of Sensor Measurements. *Sensors, 2003. Proceedings of IEEE*, Vol.2:974–979.

Krunic, V., Trumpler, E., and Han, R. (2007). NodeMD: Diagnosing Node-level Faults in Remote Wireless Sensor Systems. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, MobiSys '07, pages 43–56, New York, NY, USA. ACM.

Kumar S., A. A., Ovsthus, K., and Kristensen., L. M. (2014). An industrial perspective on wireless sensor networks-a survey of requirements, protocols, and challenges. *IEEE Communications Surveys and Tutorials*, 16(3):1391–1412.

Kunzel, G., Winter, J. M., Muller, I., Pereira, C. E., and Netto, J. C. (2012). Passive monitoring software tool for evaluation of deployed WirelessHART networks. *Brazilian Symposium on Computing System Engineering, SBESC*, pages 7–12.

Lampin, Q. and Barthel, D. (2018). Sensorlab2: A monitoring framework for IoT networks. *PEMWN 2017 - 6th IFIP International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks*, 2018-Janua.

Lodder, M., Halkes, G. P., and Langendoen, K. G. (2008). A global-state perspective on sensor network debugging. In *In HotEmNets*.

Luo, L., He, T., Zhou, G., Gu, L., Abdelzaher, T. F., and Stankovic, J. A. (2006). Achieving Repeatability of Asynchronous Events in Wireless Sensor Networks with EnviroLog. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–14.

Ma, R. M. R., Xing, L. X. L., and Michel, H. E. (2006). Fault-Intrusion Tolerant Techniques in Wireless Sensor Networks. *2006 2nd IEEE International Symposium on Dependable Autonomic and Secure Computing*, pages 85–94.

Maglaras, L. A., Kim, K.-H., Janicke, H., Ferrag, M. A., Rallis, S., Fragkou, P., Maglaras, A., and Cruz, T. J. (2018). Cyber security of critical infrastructures. *ICT Express*, 4(1):1–4.

Mahapatro, A. and Khilar, P. M. (2013). Fault Diagnosis in Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials*, 15(4):2000–2026.

Marotta, M. A., Both, C. B., Rochol, J., Granville, L. Z., and Tarouco, L. M. R. (2014). Evaluating management architectures for Internet of Things devices. *2014 IFIP Wireless Days (WD)*, pages 1–7.

McCloghrie, K., Perkins, D., and Schoenwaelder, J. (1999). Structure of management information version 2 (smiv2). STD 58, IETF. `https://tools.ietf.org/html/rfc2578`.

Miao, X., Liu, K., He, Y., Papadias, D., Ma, Q., and Liu, Y. (2013). Agnostic Diagnosis: Discovering Silent Failures in Wireless Sensor Networks. *IEEE Transactions on Wireless Communications*, 12(12):6067–6075.

Mukhtar, H., Kang-Myo, K., Chaudhry, S. A., Akbar, A. H., Ki-Hyung, K., and Yoo, S. W. (2008). LNMP- management architecture for IPv6 based low-power Wireless Personal Area Networks (6LoWPAN). *NOMS 2008 - IEEE/IFIP Network Operations and Management Symposium: Pervasive Management for Ubiquitous Networks and Services*, pages 417–424.

Neumann, A., Ehrlich, M., Wisniewski, L., and Jasperneite, J. (2017). Towards monitoring of hybrid industrial networks. *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*.

Nobre, M., Silva, I., and Guedes, L. A. (2015). Routing and Scheduling Algorithms for WirelessHARTNetworks: A Survey. *Sensors (Basel, Switzerland)*, 15(5):9703–9740.

OMA (2017). Lightweight Machine to Machine Technical Specification– Approved Version 1.0, OMA. Technical report, Open Mobile Alliance.

OMA (2018). OMA LightweightM2M (LwM2M) Object and Resource Registry. http://www. openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html.

Palattella, M. R., Thubert, P., Vilajosana, X., Watteyne, T., Wang, Q., and Engel, T. (2014). 6TiSCH Wireless Industrial Networks: Determinism Meets IPv6. *Internet of Things*, 9:111–141.

Paradis, L. and Han, Q. (2007). A survey of fault management in wireless sensor networks. *Journal of Network and Systems Management*, 15(2):171–190.

Pasqualetti, F., Dörfler, F., and Bullo, F. (2015). Control-theoretic methods for cyberphysical security: Geometric principles for optimal cross-layer resilient control systems. *IEEE Control Systems*, 35(1):110–127.

Petersen, S. and Carlsen, S. (2011). WirelessHART versus ISA100.11a: The format war hits the factory floor. *IEEE Industrial Electronics Magazine*, 5(4):23–34.

Qi, Y., Li, W., Luo, X., and Wang, Q. (2014). Security analysis of WIA-PA protocol. In *Lecture Notes in Electrical Engineering*, volume 295 LNEE, pages 287–298.

Quan Wang, W. P. (2010). A Finite-State Markov Model for Reliability Evaluation of Industrial Wireless Network. *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference*, pages 2–5.

Ramanathan, N., Chang, K., Kapur, R., Girod, L., Kohler, E., and Estrin, D. (2005). Sympathy for the Sensor Network Debugger. In *Proceedings of the*

3rd International Conference on Embedded Networked Sensor Systems, SenSys '05, pages 255–267, New York, NY, USA. ACM.

Raposo, D. (2017). WirelessHART network metrics YANG example. `https://github.com/dgraposo/wirelesshartnetworkmetrics/blob/master/wirelesshart-network-metrics.yang`.

Raza, S., Slabbert, A., Voigt, T., and Landernäs, K. (2009). Security considerations for the wirelessHART protocol. *ETFA 2009 - 2009 IEEE Conference on Emerging Technologies and Factory Automation.*

Ringwald, M., Römer, K., and Vitaletti, A. (2006). Snif: Sensor network inspection framework. Technical Report 535, Department of Computer Science, ETH Zurich.

Robertson, J. and Riley, M. (2018). The big hack: How china used a tiny chip to infiltrate u.s. companies.

Rodenas-Herráiz, D., Fidler, P. R. A., Feng, T., Xu, X., Nawaz, S., and Soga, K. (2017). A handheld diagnostic system for 6lowpan networks. *Annual Conference on Wireless On Demand Network Systems and Services (WONS)*, 1:104–111.

Rodrigues, A., Camilo, T., Silva, J. S., and Boavida, F. (2013). *Diagnostic tools for wireless sensor networks: A comparative survey*, volume 21. Springer.

Rodrigues, A., Silva, J. S., and Boavida, F. (2014). An Automated Application-Independent Approach to Anomaly Detection in Wireless Sensor Networks. In Mellouk, A., Fowler, S., Hoceini, S., and Daachi, B., editors, *Wired/Wireless Internet Communications: 12th International Conference, WWIC 2014, Paris, France, May 26-28, 2014. Proceedings*, pages 1–14. Springer International Publishing, Cham.

Romer, K. and Ma, J. (2009). PDA: Passive distributed assertions for sensor networks. In *2009 International Conference on Information Processing in Sensor Networks*, pages 337–348.

Rost, S. and Balakrishnan, H. (2006). Memento: A Health Monitoring System for Wireless Sensor Networks. In *IEEE SECON*, Reston, VA.

Sailhan, F., Delot, T., Pathak, A., Puech, A., and Roy, M. (2010). Fault Injection and Monitoring for Dependability Analysis of Wireless Sensor-Actuators Networks. *Gedsip.*

Saleae (2018). Saleae. `https://www.saleae.com/`.

Scheible, G., Dzung, D., Endresen, J., and Frey, J. E. (2007). Unplugged but connected [Design and implementation of a truly wireless real-time sensor/actuator interface]. *IEEE Industrial Electronics Magazine*, 1(2):25–34.

Scherer, B. and Horváth, G. (2012). Trace and debug port based watchdog processor. *2012 IEEE I2MTC - International Instrumentation and Measurement Technology Conference, Proceedings*, pages 488–491.

Scherer, B. and Horvath, G. (2014). Microcontroller tracing in Hardware in the Loop tests integrating trace port measurement capability into NI VeriStand. In *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*, pages 522–526.

Schoenwaelder, J. (2003). Overview of the 2002 iab network management workshop. RFC 3535, IETF. `https://tools.ietf.org/html/rfc3535`.

Schuster, H., Horauer, M., Kramer, M., Liebhart, H., and Ag, K. T. (2014). A log tool suite for embedded systems. In *7th International Conference on Advances in Circuits and Microelectronics*, pages 16–20. Citeseer.

Sehgal, A., Perelman, V., Kuryla, S., Schonwalder, J., and In, O. (2012). Management of resource constrained devices in the internet of things. *Communications Magazine, IEEE*, 50(12):144–149.

Shea, R., Cho, Y., and Srivastava, M. (2009). Lis is more: Improved diagnostic logging in sensor networks with log instrumentation specifications. *Univ. California, Los Angeles, CA, USA, Tech. Rep. TR-UCLA-NESL-200906-01*.

Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (coap). RFC 7252, IETF. `https://tools.ietf.org/html/rfc7252`.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition.

Špírek, P. (2017). JetConf. `https://github.com/CZ-NIC/jetconf`.

Stanislowski, D., Vilajosana, X., Wang, Q., Watteyne, T., and Pister, K. S. J. (2014). Adaptive synchronization in IEEE802.15.4e networks. *IEEE Transactions on Industrial Informatics*, 10(1):795–802.

Tanyakom, A., Pongswatd, S., Julsereewong, A., and Rerkratn, A. (2017). Integration of WirelessHART and ISA100.11a field devices into condition monitoring system for starting IIoT implementation. *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan, SICE 2017*, 2017-Novem:1395–1400.

Tavakoli, A., Culler, D., Levis, P., and Shenker, S. (2008). The case for predicate-oriented debugging of sensornets. In *Proceedings of the 5th Workshop on Hot Topics in Embedded Networked Sensors (HotEmNets), Charlottesville, VA*. Citeseer.

Technology, L. (2017). DC9007A - SmartMesh WirelessHART Starter Kit. `http://www.linear.com/solutions/3100`.

Teslya, N. and Ryabchikov, I. (2018). Blockchain-based platform architecture for industrial IoT. *Conference of Open Innovation Association, FRUCT*, pages 321–329.

Thubert, P., Palattella, M. R., and Engel, T. (2016). 6TiSCH centralized scheduling: When SDN meet IoT. *2015 IEEE Conference on Standards for Communications and Networking, CSCN 2015*, 1(October):42–47.

Tolle, G. and Culler, D. (2005). Design of an application-cooperative management system for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, pages 121–132.

Tran, T.-D., Oliveira, J., Sá Silva, J., Pereira, V., Sousa, N., Raposo, D., Cardoso, F., and Teixeira, C. (2015). A scalable localization system for critical controlled wireless sensor networks. In *International Congress on Ultra Modern Telecommunications and Control Systems and Workshops*, volume 2015-Janua.

Trappey, A. J., Trappey, C. V., Govindarajan, U. H., Sun, J. J., and Chuang, A. C. (2016). A Review of Technology Standards and Patent Portfolios for Enabling Cyber-Physical Systems (CPS) in Advanced Manufacturing. *IEEE Access*, 3536(c):1–1.

Wang, Q. and Jiang, J. (2016). Comparative examination on architecture and protocol of industrial wireless sensor network standards. *IEEE Communications Surveys and Tutorials*, 18(3):2197–2219.

Warriach, E. U., Aiello, M., and Tei, K. (2012). A Machine Learning Approach for Identifying and Classifying Faults in Wireless Sensor Network. *2012 IEEE 15th International Conference on Computational Science and Engineering*, pages 618–625.

Whitehouse, K., Tolle, G., Taneja, J., Sharp, C., Kim, S., Jeong, J., Hui, J., Dutta, P., and Culler, D. (2006). Marionette: using RPC for interactive development and debugging of wireless embedded networks. In *2006 5th International Conference on Information Processing in Sensor Networks*, pages 416–423.

Wightman, R. (2018). WirelessHART dissector. `https://github.com/reidmefirst/WirelessHART-Parser`.

Yang, J., Soffa, M. L., Selavo, L., and Whitehouse, K. (2007). Clairvoyant: A Comprehensive Source-level Debugger for Wireless Sensor Networks. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 189–203, New York, NY, USA. ACM.

Yoshida, J. (2019). TI Claims Breakthrough BAW Technology. `https://www.eetimes.com/document.asp?doc_id=1334373`.

Yuan, D., Kanhere, S. S., and Hollick, M. (2015). Instrumenting Wireless Sensor Networks - A survey on the metrics that matter. *Pervasive and Mobile Computing*, 37:45–62.

Yuan, F., Song, W. Z., Peterson, N., Peng, Y., Wang, L., Shirazi, B., and LaHusen, R. (2008). A Lightweight Sensor Network Management System Design. In *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 288–293.

Zand, P., Chatterjea, S., Das, K., and Havinga, P. (2012a). Wireless industrial monitoring and control networks: The journey so far and the road ahead. *Journal of Sensor and Actuator Networks*, 1(3):123–152.

Zand, P., Dilo, A., and Havinga, P. (2012b). Implementation of WirelessHART in NS-2 simulator. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA.*