Masters' Degree in Informatics Engineering
Dissertation
Final Report

Integration of SDN technologies in SCADA Industrial Control Networks

Rui Miguel da Conceição Queiroz
rqueiroz@student.dei.uc.pt

Supervisors:
Paulo Simões
Tiago Cruz
January 15, 2017

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

I feel a little funny about this because we have this term SDN and in a funny way it's always been about software, even the original packet switches at the ARPANET were based on software. It's just that you're in the middle of a kind of renaissance in thinking about what networking can do because there's more that we can get the software to do, there's more that we can get the hardware to do in cooperation with it. And so, in a very interesting way you have an opportunity to reinvent the whole notion of networking and so I'm very excited about that. It's nice to see this kind of fresh new thinking happening in spite of the fact that the internet it has been around in concept for 40 years and it has been in operation for 30. Here's an opportunity in the beginning of the 21st century to think differently and new about what networking is all about.

Vint Cerf[1]

[1] Open Networking Summit, April 2013

Acknowledgements

First and foremost, I wish to thank my thesis supervisors, Paulo Simões and Tiago Cruz, for the opportunity to emerge into the field of scientific research, as well for all the support, guidance and help provided throughout since the beginning of this project.

I would also like to thank my laboratory colleagues, Jorge Proença, Konstantia Barbatsalou and Luís Rosa, for being the best people one could wish to work with. Thank you for your willingness to help at all times and contagious joy, which gave me the strength and will to go further.

To my longstanding friend, Pedro Sousa, for his support concerning industrial control systems and also for allowing himself to be convinced to provide some material without which this research would not have been so appealing.

Lastly, this journey would not have been possible without the support of my parents and my sister all my life, who allowed me to embark on the path that lead me to this point.

Content

Abbreviations and Acronyms

ACL	Access Control List
AMQP	Advanced Messaging Queuing Protocol
ANSI	American National Standards Institute
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
BGP	Border Gateway Protocol
CS	Critical Systems
CPU	Central Processing Unit
DDC	Direct Digital Control
DCS	Distributed Control System
DSCP	Differentiated Service Code Point
DiffServ	Differentiated Services
DPI	Deep Packet Inspection
ERP	Enterprise Resource Planning
ForCES	Forwarding and Control Element Separation
GUI	Graphical User Interface
HMI	Human Machine Interface
IACS	Industrial Automation and Control Systems
ICMP	Internet Control Message Protocol
ICS	Industrial Control System
ICT	Information and Communications Technology
IoT	Internet of Things
IIoT	Industrial Internet of Things
ITU-T	International Telecommunication Union Telecommunication Standardization Sector
ISA	International Society of Automation
IETF	Internet Engineering Task Force
LISP	Locator ID Separation Protocol
MPLS	Multi-Protocol Label Switching
MPLS-TP	Multi-Protocol Label Switching Transport Profile
NAT	Network Address Translation
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NOS	Network Operating System
ODL	OpenDaylight
OF	OpenFlow
OFS	OpenFlow Switch
ONF	Open Networking Foundation
OVS	Open vSwitch
OVSDB	Open vSwitch Database
OXM	OpenFlow Extensible Match
PAC	Programmable Automation Controller
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
PoC	Proof-of-Concept
QoS	Quality of Service
REST	Representational State Transfer (Protocol)
RESTCONF	Representational State Transfer Configuration (Protocol)

RSTP	Rapid Spanning Tree Protocol
RTU	Remote Terminal Unit
SDN	Software-Defined Networking
SCADA	Supervisory Control and Data Acquisition
SOC	Security Operation Centre
TCAM	Ternary Content-Addressable Memory
TLV	Type-Length-Value
VLAN	Virtual Local Area Network
VM	Virtual Machine
WAN	Wide Area Network

List of figures

List of tables

Abstract

In recent years, Supervisory Control and Data Acquisition (SCADA) Industrial Control Systems (ICS) – a kind of systems used for controlling industrial processes, power plants, assembly lines, among others – have become a serious concern because of security and manageability issues.

After years of trusting in air-gaped isolation and obscurity security, the increased coupling of operational and information systems, together with the absence of proper management and security policies, disclosed several weaknesses in SCADA ICS. Despite not constituting any novelty within the information and communications technology (ICT) domain, which has dealt with similar problems for decades, the practices applied there could not be easily ported to the ICS domain due to the distinct priorities and requirements of each of those domains along with the limitations of the existing networks.

The rise of Software-Defined Networking (SDN) constituted a paradigm shift. Through the usage of abstraction and simplification, it brought significant gains in terms of: firstly, allowing more control over networks and information flows; secondly, bringing a new and powerful (yet simplified) network management; thirdly, enabling more flexible and adaptable networks; fourthly, and most importantly for business managers, lowering the managing and maintenance costs and allowing faster time to market, which are some of the most important parameters in competitive markets.

However, the benefits brought by SDN are yet to reach the ICS domain.

In this sense, this thesis provides an overview on the usage of such technologies and presents some solutions, based on synergies between both, which can address the problems mentioned above in order to improve SCADA ICS manageability, availability and security.

Keywords: Critical Infrastructure, Industrial Control Systems, ICS, Supervisory Control and Data Acquisition, SCADA, Software-Defined Networking, SDN

Introduction

During the last five decades, the industrial production environment has suffered several changes and improvements, in a continuous improvement effort towards the achievement of four key goals: increased production, improved quality, lower costs and maximum flexibility [1].

Until the 60s, all the production control systems were simply mechanical or hydraulic. Later on, then the 4-20 mA analogue interface was established as a standard for instrumentation technology, setting itself as the first standardized communication interface across manufactures. In most cases it was used as a simple interface between operators and relay panels.

In 1969, the first programmable logic controller (PLC) was invented in the scope of the automotive industry in order to replace the relay panels and hard-wired programming, which had several downsides. The rising of the PLC was probably the biggest responsible for the mass implementation of digital industrial control, even though some hardware preforming digital control (direct digital control – DDC) already existed, since the beginning of the 60s, in the industrial control world [2] (despite having different characteristics and costs). Soon after, in 1971, the first microprocessor [3] is launched on the market and becomes the major precursor for the release of the Honeywell TDC 2000 and the Yokogawa Centum, the first distributed control systems (DCS's). Since then, we have witnessed a widespread of digital computers and distributed control systems. The latest were intended to allow systems to spread out over a geographical area. However, the existing communication networks were unable to support such a wide geographical range and so the major manufactures were forced to develop their own industrial communication networks, with proprietary protocols, to support their needs.

In the meantime, while DCS's technology was evolving and was being adopted in all kind of industries, communication networks outside the industrial environment were also changing. In 1974, the initial TCP specification [4] is presented and the following year a two-network TCP/IP communication test is performed. In 1976, a paper [5] is published describing the results of an experience using new "…communication system for carrying digital data packets among locally distributed computing stations" that could support any computer. A few years later, in the beginning of the 80s, TCP/IP became standardized. Non-industrial computer networks started to proliferate, due not only to the emergence and evolution of personal desktop computers and user friendly operative systems, but also to the increasing use of TCP/IP, among other developments.

In the 90s, computer networks were already present in the offices of several public and private institutions.

Each of these networks (industrial, later called Industrial Control System (ICS); non-industrial, referred to as Information and Communication Technology [ICT]) followed different paths as to accomplish different purposes. While ICS networks were intended to achieve availability, reliability, real-time capabilities and were based on proprietary technology, ICT networks prioritized security, confidentiality, resilience – moreover, their core technology mostly consisted on open standards.

Nevertheless, in the early 2000, bearing in mind the four key goals of industry and, hence, as to improve business and organizational workflows, executives started to realize the importance of integrating the field information directly into enterprise resource planning systems (ERP). In some cases [6], predictions pointed out to a 70% efficiency gain. In order to accomplish this integration, it would be necessary to merge the ICT and ICS networks. The natural choice for this merging felt over Ethernet and TCP/IP. Not only had ICT networks had a worldwide massive deployment, but Ethernet and TCP/IP had also evolved considerably since their first appearance and were already pretty mature technologies built on open standards. Furthermore, ICT networks presented the best overall cost/benefit. Since these ICS networks demanded specific requirements and were connected to industrial hardware that was only prepared to function with specific protocols, the chosen solution was to adapt the existing industrial protocols, to enable them to work over the new network.

Up to that moment, ICS ecosystems benefited from an air gap isolation and, since most of its implementations were based on proprietary technology, its specificities were known only by a handful of technicians and programmers ("obscurity approach") working for the companies which developed such technologies. Being so, decision makers never thought security could be an issue and so it was never taken into account. This approach was proven wrong in several cases, for instance:

- In 2000, Maroochy water system was attacked [7] [8] on several occasions through the radio signalling system, resulting in 800.000 litres of raw sewage being drained into local parks and rivers and, consequently, causing serious damage to local fauna and flora.
- In 2010, a worm dubbed stuxnet [9] [10] struck and disabled the Iranian nuclear facility at Natanz, causing serious damage to the system. This took its toll on the country's political policies and had a negative financial impact of millions of dollars. Again, the air gap was bypassed, this time by using USB device to transport the "virus".

This became an even more serious problem since the integration of both networks exposed the ICS to the outside world through the ICT network, which by default had already a lot more people accessing it and, to make things worse, was frequently connected to other networks such as internet. Moreover, and despite all the positive features of ICT networks, these weren't ready to deal with all the issues addressed by ICS, just like ICT network managers, since the working and fundamental goals were completely distinct. The combination of these factors paved the way for even more attacks like the ones referred in [11] [12].

According to some network and security companies, there is an increasing report of attacks to ICSs ("worldwide SCADA attacks increased from 91,676 in January 2012 to 163,228 in January 2013, and 675,186 in January 2014" [13]) and its harmfulness ("it is likely or extremely likely that a cyberattack will take down critical infrastructure and cause loss of human life in the next three years" [14]).

Nowadays, ICSs are widely spread, being involved in processes responsible for our water supply, telecommunication, agriculture, food production, electricity production and distribution, public health, natural gas, fuel and oil, financial services, transportation systems or security services, just to mention some examples. Therefore, ICSs have also begun to be referred to as critical systems (CSs), something that clearly demonstrates its strategic importance. Being so, it is urgent to discover and implement new ways to make these systems more secure and reliable, in order to guarantee their stable operation. Until now, and despite the adoption by ICS systems, conventional network technologies have failed short in responding to some of the specific needs of this domain, in a cost-efficient way.

Meanwhile, in 2008, a new network paradigm emerged – Software-Defined Networking –, depicted as a way to bring new potential to networking through abstraction and simplification: firstly, allowing more control over networks and information flows; secondly, bringing a new and powerful (yet simplified) network management; thirdly, enabling more flexible and adaptable networks; fourthly, and most importantly, especially for business managers, lowering the managing and maintenance costs; finally, allowing faster time to market, which is one of the most important parameter in competitive markets. In spite of being a relatively fresh technology, still at a sharp evolution stage and with a low implementation rate in production environments, some case studies brought great credibility to this new approach to networking, for instance Google [15] [16], where since 2012 all the datacentre backbone traffic is carried on top of SDN.

Hence, one might wonder whether this could be an asset to the ICS world. This rationale was the starting point for the thesis hereby documented.

Research Statement

There is a question whether SDN will be able to fill the gaps of traditional networks and simultaneously meet the requirements inherent to real-time systems. Furthermore, one might question if the conjugation of both SDN and ICS will result in more flexible, reliable and smart systems. At first glance it seems possible and realistic to think that innovations to ICS can emerge from synergies between both technologies.

In this sense, the main goal of the research done during this internship, at Laboratory of Communications and Telematics of the Engineer Informatics Department of the University of Coimbra, is the merge between ICSs and SDN to ascertain the potential result of this synergy.

Thesis Outline

This thesis is organized as follows:

- In Chapter 1, the fundamental concepts of programmable networks are described, giving special emphasis to the main elements of an OpenFlow compliant environment, and SCADA systems.
- In Chapter 2, a reflection is made on the improvements that SDN can bring to ICS.
- In Chapter 3, a detailed description of the research that was done is presented, focusing on aspects such as the built testbed, the used tools, the tests carried out and minor conclusions drawn.
- In Chapter 4, the conclusions of this research work are presented and discussed, as well as other ongoing and future developments of this research effort.

Chapter 1 – State of the Art

1.1. Software-Defined Networking

Software-defined networking is a new approach to communication networks which, through abstraction, tries to simplify all its complexity of configuration and management and, at the same time, brings new and more powerful features.

In spite of the reference to the SDN designation in 2009, in a MIT Technology Review's [17] article, the concept comes from a natural evolution from projects and ideas of the past 20 years [18] and gains greater importance in 2008, when a team lead by Nick Mckeown published an article [19] on a new network architecture protocol they named OpenFlow (OF). In this paper, the authors presented a solution that would allow researchers to overcome existing barriers in current networks, which were precluding them to test new solutions that could bring improvements and innovation to the network world, by enabling the overcome of the stagnation of conventional networks.

After some proof-of-concept (PoC), the alpha version (OF 1.0) was made publicly available in early December 2009 and, that same year, more precisely on the last day of the year, the first stable version with support for installation in dedicated hardware solutions [20] was officially launched. This first version already contained several innovative features in comparison to forwarding systems at that time [21] such as: number of the analysed header fields, counters for statistical parameters, range of allowable actions.

By this time, important players encouraged and supported this new technology, as it was the case for Ericsson, with OpenFlow-MPLS project, which intended to accomplish an integration between MPLS and OF [22], or Google that, in the Spring of 2010 [16], initiated the first phase of implementation of OF in their data centres, thus becoming the first major company to deploy and confirm the potential of this technology in a production environment.

In February 2011 the OF 1.1 [23] version was launched with new features, especially in terms of MPLS, multipath, VLANs, logic gates and support for multiple flow tables.

That same month, in a joint effort and in order to leverage the standard development and adoption of this new technology, the creation of the Open Networking Foundation (ONF) [24] was formalized, becoming responsible for the future development and open publication of the OF. Its initial members were: Broadcom, Brocade, Ciena, Cisco, Citrix, Dell, Deutsche Telekom,

ErICSson, Facebook, Force10, Google, HP, IBM, Juniper Networks, Marvell, Microsoft, NEC, Netgear, NTT, Riverbed Technology, Verizon, VMware, and Yahoo.

From then until the current date, as seen on **Figure 1**, four more main versions of OF were released [25] [26] [27] [28] bringing significant improvements to this standard, such as support for IPv6, extensible headers, QoS, Q-in-Q tunnelling (bridging backbone which allows for tunnelling across datacentres), statistical data on each flow, dynamic structures based on type-length-value (TLV) [29], among others.

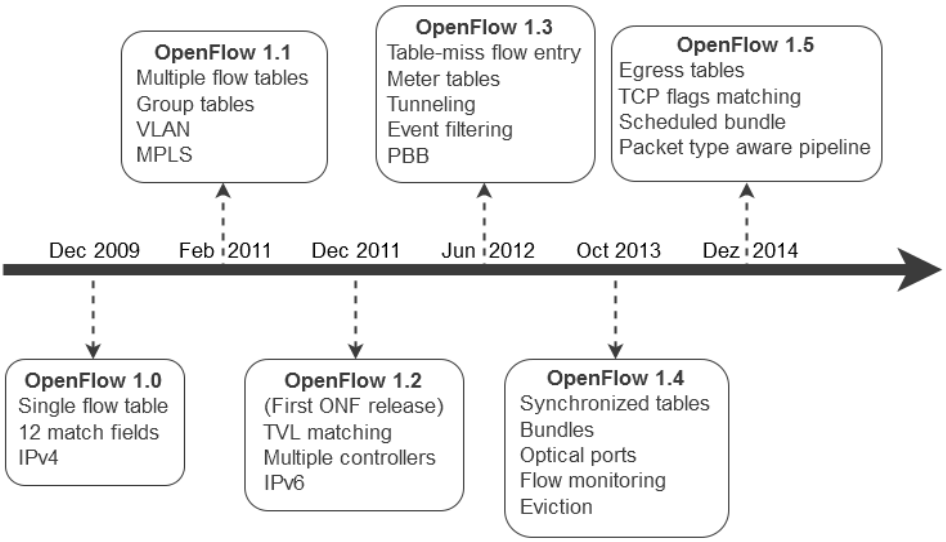


Figure 1 – OpenFlow timeline

1.1.1. Conventional Network Limitations

Until now, the majority of networks is based on hardware which consists of two merged layers (**Figure 2**). These two layers, depending on their respective routing protocols and policies, manage the network traffic behaviour. The data/forwarding plane is responsible for packet forwarding, and the control plane for making decisions on how these packets should be processed.

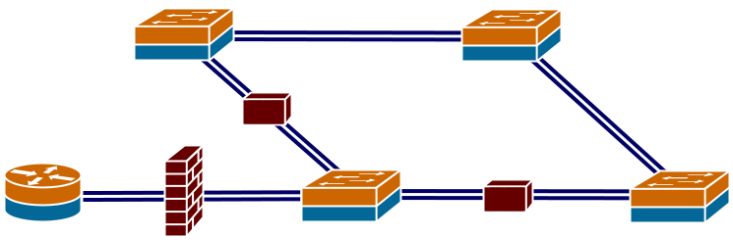


Figure 2 – Conventional network

This architecture allows us to easily identify problems such as:

- **Limitation of the potential for innovation**: in most cases, these solutions are proprietary and closed, making it impossible for a researcher (external "to the brand") to access both the necessary information and the permissions that would allow him to test new theories, mechanisms or solutions (e.g. new routing protocols, security, QoS, etc.) in networks with significance in terms of size, complexity and density (usually: production environments).
- **Vendor lock-in**: for the above mentioned and also by strategic choice of equipment vendors, in most cases, it's mandatory to have a homogeneous network in terms of the brand of the various equipment's in order to enable the full potential of the network equipment and make use of solutions or protocols that represent some improvement.
- **Complexity**: most of the active equipment, especially when from different brands, require individual configuration due to the presence of the control and data planes in each of them (e.g. reconfiguring VLAN's, ACL's, etc.). This need not only increases management and maintenance costs of the networks, but also creates the problems identified below.
- **Performance and safety**: by the previously identified need, the likelihood of inconsistencies across the configuration of multiple equipment is high. This can easily lead to performance loss situations (e.g. traffic prioritization) or to situations where security policies are not uniform across the network, creating permeability situations in it.
- **Adaptability/Scalability**: the complexity described above requires that, whenever there is a need to adapt the network to a new reality, prior knowledge of most settings of the network is demanded so that the appropriate changes can be made in a correct away. A good example are the situations related to physical changes in the networks, as well as situations that call for a policy change, whether security, routing, QoS or others.

If we have a business vision of these factors, we realize that all these problems have one thing in common: all lead to a negative impact, which has multiple repercussions. Moreover, the limitation of the routing and forwarding systems is also something that should be considered, since the processing decisions are taken having, mainly, only two parameters in consideration: physical address (layer 2) and IP address (layer 3).

1.1.2. Architecture

SDN emerged as an attempt to eliminate the negative factors mentioned in the previous subsection.

Strongly based on a predecessor project, Ethane [30], the fundamental idea behind SDN is quite simple and intuitive. Taking advantage of the fact that most of the routers and Ethernet switches contain ternary content-addressable memory (TCAM) [31] – that runs at line-rate (normally used to for QoS, firewalls, NAT and to collect statistics) –, and have a set of functions which are transversal to all, a protocol was created that could extend these common features allowing the creation of fast flow tables, which could be programed in an accessible, dynamic and remote way, independently of the hardware in question. To make it possible, it was necessary to create an abstraction of the physical layer and hence the control and the data plane, which were previously within the same hardware components, were separated (**Figure 3**).

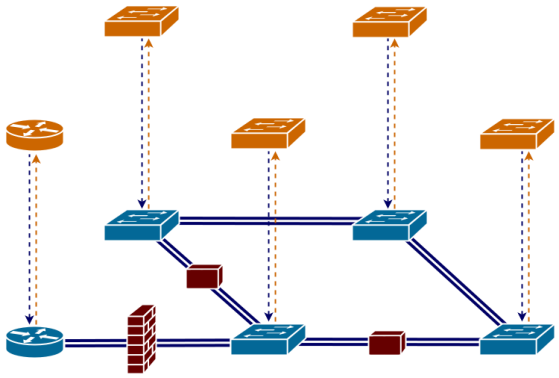


Figure 3 – Data and control plane separation

In this sense, the data plane remains on "the same" equipment where it was allocated before (switches, routers) with the individual device "intelligence" being removed, by passing the control plane to a centralized point (from a logical point of view). Thus, switches and routers become mere forwarding boxes and the entire network programming is concentrated on a single point, greatly simplifying the control and management of the network.

As a result of this separation, SDN networks are composed of three distinct components:

- **Equipment/devices**: off-the-shelf hardware responsible for packet forwarding (such as switches). This action is taken in accordance with rules present in the flow tables it is hosting;
- **controller**: software that takes decisions on the way packets should be processed and then populates the flow tables, present in switches, with the rules derived therefrom;
- **secure channel**: communication channel that allows the exchange of information between switches and the controller and vice versa.

However, after analysing conventional networks in a more detail, we realize that there are other components in the network, such as firewalls, load balancers, NATs, IDSs among others (**Figure 4**).

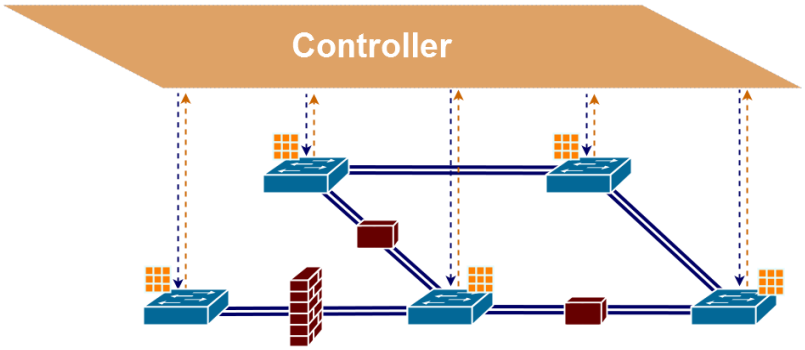


Figure 4 – SDN simplified architecture

So, it did not make sense to remove decision-making capabilities from switches and then keep these middle boxes scattered around the network, as it would bring the same kind of drawbacks. There was also the need to withdraw these middle boxes from the middle of the network, relocating them "above" the controller (**Figure 5**), transforming them in SDN-aware functions.

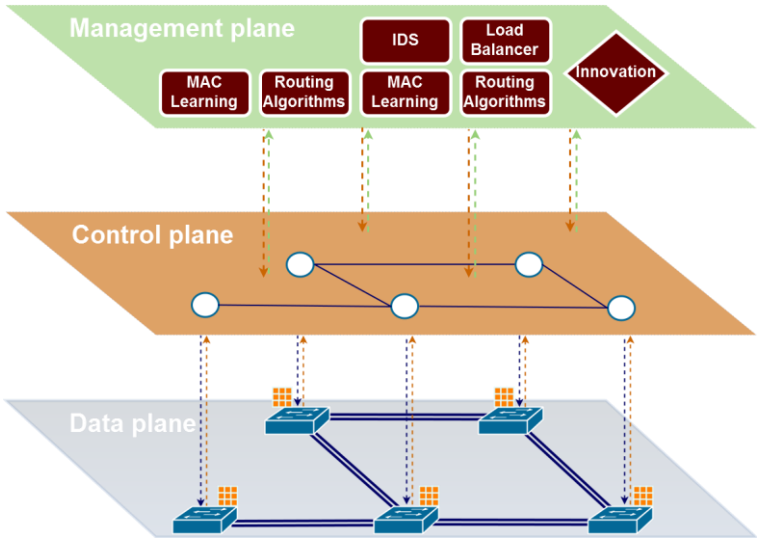


Figure 5 – SDN architecture

Dissecting the architecture of SDN in more detail, we are able to identify some new “inner layers” which may help to better understand all the structure behind SDN and its functioning (**Figure 6**).

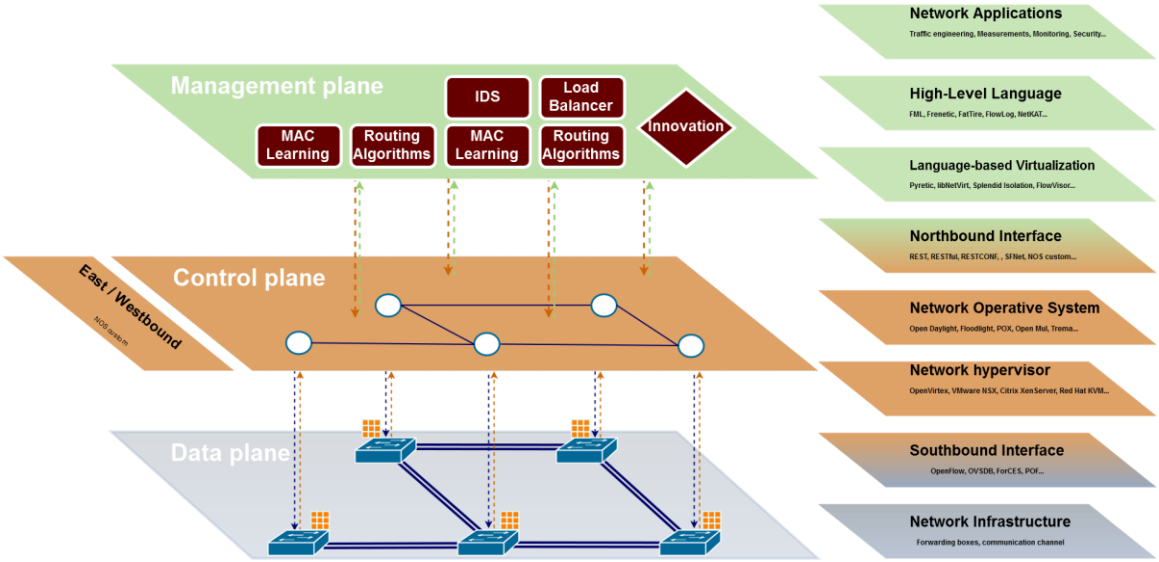


Figure 6 – SDN inner layers

These layers (next described, following a bottom-up approach) encompass several functions, namely:

- Data plane
    - **Network infrastructure**: forwarding hardware (switches) without, in general terms, any type of autonomous decisions capabilities before being programmed by the controller.
    - **Southbound interface**: standard application programming interfaces (APIs) mainly responsible for connecting the forwarding and control elements.
- Control plane
    - **Network hypervisor**: virtual hosting which allows for resource allocation on-demand enabling the provisioning of elastic services in a flexible and easy way.
    - **Network operating system** (NOS) also called **controller**: responsible for providing abstractions from the hardware; essential services such as topology, statistics, notifications, device management, shortest path forwarding and security; APIs for those services.
    - **Northbound interface**: APIs responsible for connecting the control and the management plane. The type of used/available APIs depends on the chosen NOS. Thus far, REST and RESTful are the most used.
    - **East/Westbound interface**: APIs responsible the connecting of multiple controllers in order to create a distributed controlled network. The type of used/available APIs depends on the chosen NOS.

- Management plane
    - **Language-based virtualization**: virtualization based on programming languages that offer abstraction regarding the network (e.g. allowing the use of "network objects" – the object network can represent several switches).
    - **Programming languages**: high-level programming languages that offer abstraction regarding the network.
    - **Network applications**: the intelligence of the entire network. Responsible for processing information and defining the way packets are dealt with.

This abstraction and separation of planes allowed networks to start being treated as a whole, and not as a set of multiple hardware components, which opened the door to a new vision: instead of concentrating on the way a specific behaviour has to be implemented to achieve a particular purpose, the focus should now be on the overall (or higher-level) objective that we want the system to adopt.

As shown in **Figure 7**, and explained by Scott Shenker [32] in his presentation [33], if we intended to stop X from talking to Y in a conventional network, we would have to choose a path and then tell all the including nodes to drop those packets. Then, we would have to check for alternative paths and to do the same with the new nodes that were included. If, for any reason, there was a change in the network architecture, we would have to go through all the process again. In SDN we could just define that X can't talk to Y.

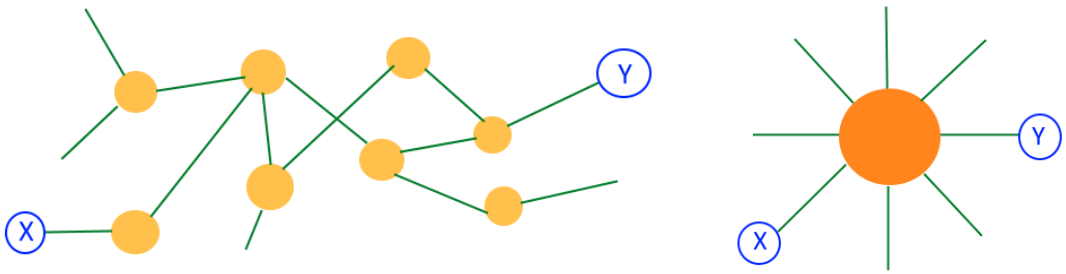


Figure 7 – Conventional networks vs SDN

1.1.3. Southbound Interfaces

As mentioned above, southbound interfaces are standard APIs, which are mainly responsible for connecting the forwarding and control elements. Depending on the protocols in usage, the APIs will change accordingly. There are several possible protocols, such as:

- **OpenFlow** – by specifying the configuration of the communication channel and the operation of forwarding elements (such as OpenFlow Switches – OFSs), it enables the controller to configure the behaviour of OF compliant switches.
- **Open vSwitch Database (OVSDB)** [34] – created by Nicira (Open vSwitch creators), it is a management protocol used to remotely configure the switch itself (bridges, ports, interfaces…) on an SDN environment. Several vendors started to support this protocol.
- **MPLS Transport Profile (MPLS-TP)** [35] – created by a joint ITEF/ITU-T effort, it is a variant of MPLS meant to be used on packet-switched networks, specially by carriers and service providers.
- **Border Gateway Protocol (BGP)** [36] – aims at exchanging network routing and reachability information with other BGP systems.
- **OpFlex** [37] from Cisco – it is based on declarative control and on moving some complexity to the leaf elements. Policies are defined at the controller and then sent to the switches, which all adapt in order to try to accomplish those policies.
- **NETCONF** [38] – it is used to configure and manage network devices.
- **Locator ID Separation Protocol (LISP)** [39] **–** aims at improving the scalability of routing by enabling network virtualization through dynamic multitenant overlays.
- **Forwarding and Control Element Separation (ForCES)** [40] **–** defines a master-slave kind of communications for messages exchange between the control (master) and forwarding planes (slave), which allows the controller to configure the behaviour of OF compliant switches.

Some are well known protocols imported from conventional networks (e.g. BGP, NETCONF), mostly for backward compatibility and for heterogenic environments, others are new protocols created in the context of the architecture of SDN (e.g. ForCES, OpenFlow, OVSDB).

As stated before, currently, the *de facto* standard is OpenFlow. However, it is important to be aware that not all southbound interfaces are competing with each other, some are complementary and others are used for specific purposes that may even be independent of the SDN concept. This means that in a SDN it is possible to have several southbound interfaces active in simultaneous using different protocols.

1.1.4. OpenFlow[2]

Through the usage of this protocol and compliant APIs, the controller, in order for the network to start providing the desired behaviour to achieve a certain objective, can sense and manage all the infrastructure by communicating and programming the OFSs (which are the class of forwarding elements used in this research effort). An *OpenFlow* compliant environment mainly relies on a set of well-defined components, namely: controller, communication channel and OFS (**Figure 8**); where the controller makes use of its southbound OF API to communicate with the OFS over a TCP connection.

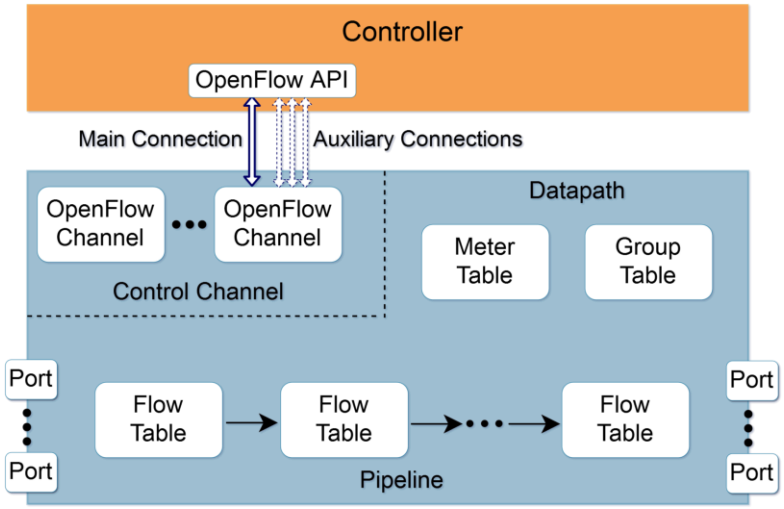


Figure 8 – OpenFlow main components[3]

1.1.4.1. Communication Channel and Messages

The communication channel, which is by default instantiated as a single network, allows the connection and supports the message exchange between the OFSs and the controller. This channel is typically encrypted using TLS or may run directly over TCP.

There are three main **message types**:

[2] The following OpenFlow related information is based on OpenFlow Switch Specification 1.4 (October 14, 2013), unless stated otherwise.
The OF specification contains several optional features which may or may not be present in a certain implementation. It is the controller responsibility to retrieve that information from the switches.
[3] Adapted from [48]

- **Controller-to-switch** – originated by the controller and may or may not demand a response from the switch, such as for requesting available features, configuring and managing or requesting stats update.
- **Asynchronous** – sent by the switch without being actively solicited by the controller to announce a port status change, a flow removal or a packet-in which transfers the control of an arrived packet to the controller.
- **Symmetric** – may be initialized by the OFSs or the controller without solicitation and are used upon connection start-up, as keep-alive notifications or as experimental messages for additional functionalities.

All the messages have guaranteed delivery and processing, except in case of complete channel failure. Nevertheless, switches may reorder the received messages for performance enhancement. In this case, flow rules may be installed in a different order from the one sent by the controller. In some situations, there can be rule dependencies which imply a certain processing order (e.g. when a rule references a group which is still to be created, it is mandatory to first process the message related to the group creation and only after the forwarding action), in order to ensure it, *barrier* messages should be used. This technique forces the processing of all the messages before a barrier and only after it the following messages can be processed.

In case there is a **connection interruption,** which prevents a switch from communicating with any controller, one of two modes can be immediately and automatically activated:

- **Standalone**[4] – the switch becomes autonomous and assumes the processing of all the packets, including the ones that were intended to be send to the controller, adopting the role of a typical Ethernet switch or router.
- **Secure mode** – the switch continues to behave "normally", processing the packets according to the rules which were already installed. All the other packets and messages that are meant to the controller are dropped.

To improve **processing performance**, it is possible to take advantage of the parallelism present in most switch implementations and expand the channel to multiple auxiliary network connections, which can even be assigned with different processing priority.

[4] This mode may not be available on all the switches. It is normally supported by hybrid switches.

1.1.4.2. Flow Table

Every OF compliant switch contains at least one flow table. These are data structures that describe the characteristics of every packet flow and its corresponding processing instruction (and some other information as described next in this chapter). Each flow table has a self-identifying number. Every time a new packet arrives to the switch, a packet matching action takes place, it starts in the flow table number zero and follows an ascending order until a match is found.

Each flow table contains rows, normally called flow entries, consisting of six main components:

Match fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 1 – Composition of a flow entry

- **Match fields**: fields regarding packet headers, ingress port and/or metadata against which a packet is matched in order to identify what instruction to execute. An OFS has to support at least 13 specific mandatory match fields (signalled in bold in **Table 2**). Nevertheless, in the OF specification 1.4 match fields as well as programming structures to allow experimental match fields are considered.

Switch input port	ARP opcode
Switch physical input port	ARP source IPv4 address
Metadata passed between tables	ARP target IPv4 address
Ethernet destination address	ARP source hardware address
Ethernet source address	ARP target hardware address
Ethernet frame type	**IPv6 source address**
VLAN id	**IPv6 destination address**
VLAN priority	IPv6 Flow Label
IP DSCP (6 bits in ToS field)	ICMPv6 type
IP ECN (2 bits in ToS field)	ICMPv6 code
IP protocol number	Target address for ND
IPv4 source address	Source link-layer for ND
IPv4 destination address	Target link-layer for ND
TCP source port	MPLS label
TCP destination port	MPLS TC
UDP source port	MPLS BoS bit
UDP destination port	PBB I-SID

SCTP source port	Logical Port Metadata
SCTP destination port	IPv6 Extension Header pseudo-field
ICMP type	PBB UCA header field
ICMP code	EXPERIMENTAL

Table 2 – Match fields

- **Priority**: value that defines the matching precedence of the flow entry. The higher the value the highest priority it has. This field is normally used to prioritize more specific rules in order to ensure the intended granularity for the network.
- **Counters**: statistical data related to each flow entry, which is updated every time packets are matched.
- **Timeouts**: value that defines the amount of time (hard timeout or idle timeout) before a flow is deleted from the flow table by the switch. Defining timeouts is fundamental for the control of memory allocation and packets processing speed. The smaller the flow tables are, the less memory is needed as well as for the processing power and time. By keeping the hardware requirements lower, one will be able to reduce the cost of the switches.
- **Cookie**: id assigned by the controller, which may be used to filter flow statistics, flow modifications and flow deletion.
- **Instructions**: (or set of instructions) that are executed when a packet is matched. The execution of the instructions will result in changes to the packet, pipeline processing or/and action set. There are six supported types:
    - **Meter** (optional) – sends the packet directly to a stated meter.
    - **Apply-actions** (optional) – immediately applies the actions specified on the action list it includes.
    - **Clear-actions** (optional) – clears the actions present in the action set.
    - **Write-actions** (required) – merges the actions specified into the current packet's action set.
    - **Write-metadata** (optional) – writes the stated metadata into the metadata field.
    - **Goto-table** (required) – designates the next table in the processing pipeline.

The *apply-actions* instruction type is composed by an action list which may contain one or more of the following actions:

- **Set-Queue (optional) –** defines the queue which is attached to the port to where the packet is forward.
- **Drop (required)** – discards the packet.
- **Group (required**) – forwards the packet to the specified group.

- **Push-Tag/Pop-Tag** (optional) – Pushes or pops tags from Ethernet header (e.g. VLAN, MPLS).
- **Set-Field** (optional) – rewrites the specified field(s) of the header of the packet.
- **Change-TTL** (optional) – modifies the value of the time-to-live field (e.g. IPv4 TTL, IPv6 Hop Limit, MPLS TTL).

- **Output** (required) – forwards the packet to the specified OF port, which in turn can be:
  - **Physical** – corresponds to a hardware network interface.
  - **Logical** – can be mapped to several physical ports and include packet encapsulation. These are similar to physical ports in terms of internal structure, being the only difference the possibility of having an extra metadata field (*Tunnel-ID*), and in terms of processing since they interact with OF just like a physical port.[5]
  - **Reserved** – represents generic common forwarding actions or related scenarios[6]:
    - **ALL** – sends a copy of the packet to all ports, for the exception of the ingress port and ports that were previous configured to drop forwarded packets.
    - **TABLE** – sends the packet to the first flow table.
    - **IN_PORT** – sends the packet out through its ingress port.
    - **FLOOD** – sends the packet to all ports through the non-OpenFlow pipeline, with the exception of the ingress port and ports that were previous configured to drop forwarded packets.
    - **LOCAL** – can be used as an ingress or output port in the stack of the switch's local networking and management stack, which, for example, can be useful for creating an in-band controller connection.
    - **NORMAL** – sends the packet out by using a non-OpenFlow pipeline.
    - **CONTROLLER** – when used as an ingress port, it identifies the packet as coming from the controller, when used as an output

[5] Although logical ports demand some more extra processing, this is done outside the scope of OF.
[6] *NORMAL* and *FLOOD* are only available with hybrid switches.

port it encapsulates the packet in a *packet-in* message and sends it to the controller.

- **ANY** – simply represents an absence of port definition in a command such as a wildcard situation.

The above mentioned actions can interact with the following layers:

- **Layer 1** – Port
- **Layer 2** – Ethernet
- **Layer 3** – IPv4, IPv6, MPLS, ARP
- **Layer 4** – TCP, UDP, ICMPv4, ICMPv6, SCTP

The following table gives a clear view of the scope of the OF instructions.

Layer	Protocol/Instruction	Behaviour	Description
Layer 1	---	Drop	Discards packet
	Group	Forward	Group ID: apply group processing
	Output	Forward	Port ID: forward to physical or logical port
			All: flood all but ingress port
			In_Port: send out ingress port
			Controller: send to controller
			Table: controller injected packets
			Local: host networking stack
			Flood: non-OpenFlow flood
			Normal: Non-OpenFlow dataplane
	Queue	Set	Queue ID
Layer 2	Ethernet	Set	Source MAC
			Destination MAC
			VLAN ID
			VLAN Priority
			PBB I-SID
			PBB UCA
		Push/Pop	VLAN ID
			PBB I-SID
Layer 3	MPLS	Set	Label
			Traffic Control
			TTL
			Bottom of Stack
		Decrement	TTL
		Push/Pop	Label
		Copy-In	TTL: copy outermost TTL to next-outermost TTL
		Copy-Out	TTL: copy innermost TTL to next-innermost TTL
	ARP	Set	Opcode
			Sender Hardware Address
			Sender Protocol Address
			Target Hardware Address
			Target Protocol Address

Layer	Protocol/Instruction	Behaviour	Description
Layer 3	IPv4	Set	Source Address
			Destination Address
			DSCP
			ECN
			TTL
		Decrement	TTL
		Copy-In	TTL: copy outermost TTL to next-outermost TTL
		Copy-Out	TTL: copy innermost TTL to next-innermost TTL
	IPv6	Set	Source Address
			Destination Address
			DSCP
			ECN
			Flow Label
			TTL
			Extension Header
		Decrement	TTL
		Copy-In	TTL: copy outermost TTL to next-outermost TTL
		Copy-Out	TTL: copy innermost TTL to next-innermost TTL
Layer 4	TCP	Set	Source Port
			Destination Port
	UDP	Set	Source Port
			Destination Port
	SCTP	Set	Source Port
			Destination Port
	ICMPv4	Set	Type
			Code
	ICMPv6	Set	Type
			Code
			ND Target Address
			ND Link-Layer Source
			ND Link-Layer Target

Table 3 – OpenFlow actions instructions, protocols and layers[7]

1.1.4.3. Group Table

Group tables are used to assign more specific forwarding action to each flow entry. A reference to a specific group table can be achieved by applying the *group* action in the instructions field of a flow entry, giving it a more flexible and complete set of actions.

[7] Based on [112]

Each group entry consists of four elements:

Group identifier	Group type	Counters	Action bucket

Table 4 – Main components of a group entry

- **Group identifier**: unique value used to identify the group.
- **Counters**: statistical data related to each group, which is updated every time packets are processed by that group.
- **Action buckets**: an ordered list of action buckets where each action bucket contains a set of actions to execute and associated parameters.
- **Group type**: label used to declare the type of group among the following possibilities:
    - **All** (required): executes all buckets in the group. The packet is cloned for each bucket and each one is processed according to the actions of respective bucket. It can be used for multicast or broadcast forwarding, among others.

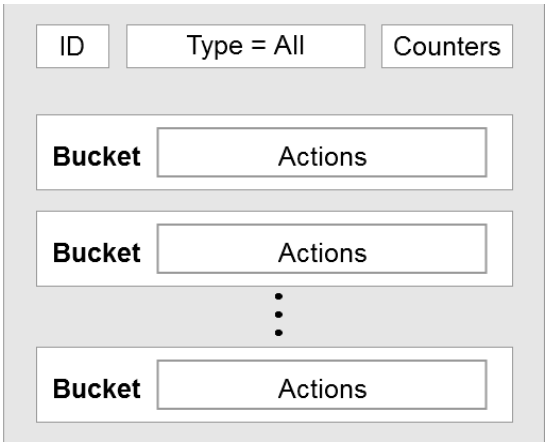


Figure 9 – All group type[8]

    - **Indirect** (required): executes the only defined and allowed bucket in the group. This group type is effectively identical to an "all" group with only one bucket. Since multiple flow entries or groups can point to a common group identifier, common actions can be configured allowing faster and more efficient convergence (e.g. next hops for IP forwarding).

[8] Adapted from [116]

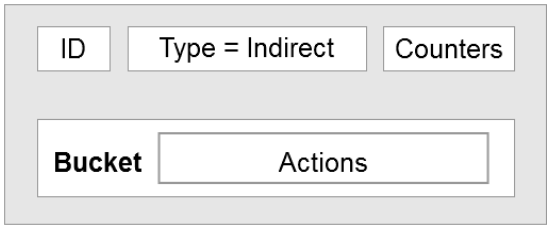


Figure 10 – Indirect group type[8]

- **Select** (optional): executes only one of the buckets in the group. The bucket is selected based on a switch-computed selection algorithm which can go from a simple round-robin to a custom algorithm using some other criteria. It can be used for load balancing or other roles depending on the used algorithm.

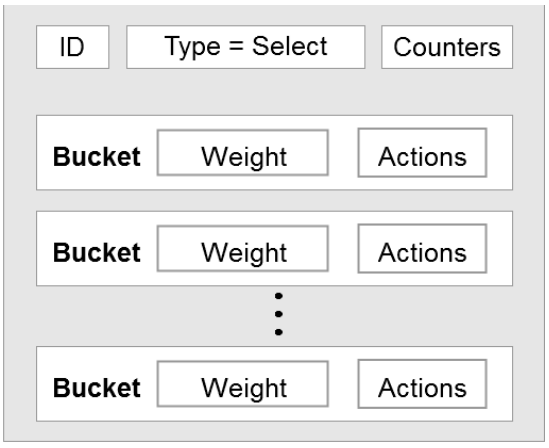


Figure 11 – Select group type[8]

- **Fast failover** (optional): executes only the first live bucket. Each action bucket is associated with a specific port and/or group. When a packet is processed, the buckets are evaluated and the first bucket, which is associated with a live port/group, is chosen. This enables the pre-definition of backup actions in the switch, avoiding the need to wait for a round trip to the controller. It can, for example, be used for fast failover recovery in case of link failure.

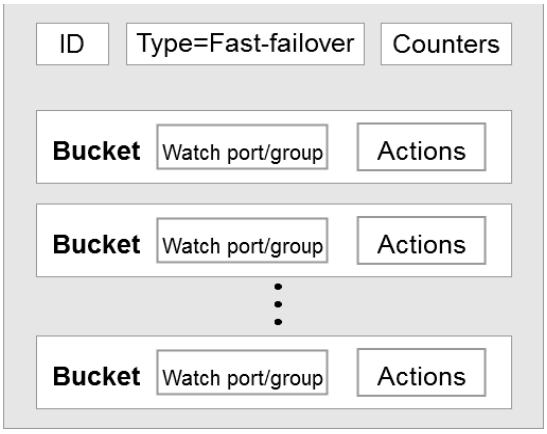


Figure 12 – Fast-failover group type[8]

1.1.4.4. Meter Table

Meter tables (**Figure 13**) allow to monitor the ingress rate of packets defined by each flow entry or by each custom group of flow entries and assign specific actions accordingly before their output. By using the instruction *meter*, packets matching an entry can be pipelined to a specific meter containing one or more pre-defined meter bands.

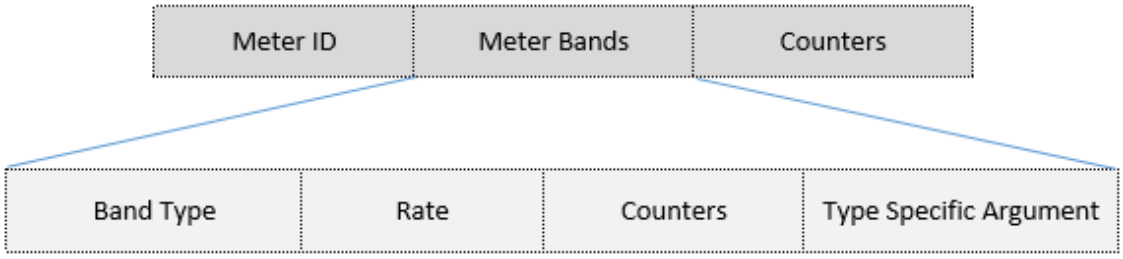


Figure 13 – Main components of a meter entry

Each **meter** entry consists of three elements:

- **Meter identifier**: unique value used to identify the meter.
- **Meter bands**: an unordered list of rate bands where each specifies the maximum allowed rate and the way to process the packet.
- **Counters**: statistical data related to all the meter bands, which is updated every time packets are processed by meter bands.

In turn, each **meter band** entry consists of four elements:

- **Band type:** may define how packets are processed (e.g. drop) or influence the way they will be processed in a next step (e.g. differentiated service code point – DSCP – remark) by using a pre-defined name type (see below, Figure 14). Although there is no required type, two optional ones are commonly available:
    - **Drop –** discards the packet.
    - **DSCP remark –** increases the value of the drop precedence of the DSCP in the IP header of the packet.
- **Rate:** value that defines the lowest rate at which the meter band is applied. By default, the rate value is in Kilobits per second but it can also be defined in packets per second (*OFPMF_PKTPS* flag).
- **Counters:** statistical data related to each meter band, which is updated every time packets are processed by that meter band.
- **Type specific arguments:** field that stores optional arguments of some band types (e.g. precedence value for the DSCP remark).

Despite not being represented above since it is done through the declaration of a non-conceptual internal flag (*OFPMF_BURST*), it is also possible to configure the flexibility of the meter by means of defining a burst value, which is common to all the bands.

When the packet/byte rate exceeds the threshold of one of the pre-programmed rate bands, it triggers it and the respective action is processed (**Figure 14**). This means that actions are taken based on the rate of the received packets. If more than one band is exceeded, the used band is the one with the highest configured rate.

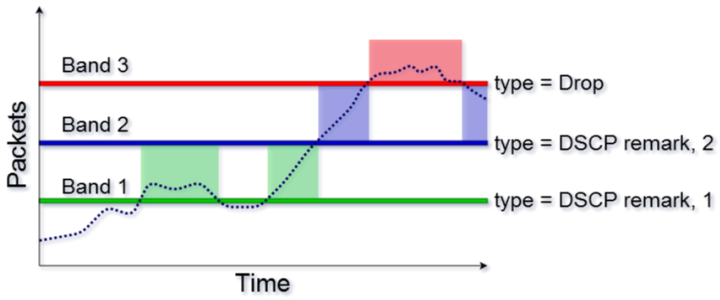


Figure 14 – Meter with 3 bands

Being so, it is possible to create a rate limiter service based on the "drop" meter type. When the rate goes over the threshold we defined, the packet is dropped and the processing rate is assured.

It is important not to confuse meters with queues, which have a rigid behaviour and simply output packets at a certain rate. Also, although queues are supported by OF, in order to provide basic QoS support, their configuration is outside the scope of OF and has to be done by means of an external configuration protocol or through a command line tool (e.g. for OVS queues can be created and configured by using the *ovs-vsctl* command).

However, meters and queues can complement themselves as to provide more complex and flexible QoS services, such as differentiated services (DiffServ). By using the "DSCP remark" meter type, it is possible to dynamically increase or decrease the dropping priority of certain traffic according to some pre-defined logic/algorithm and then feed it to the queues, so that they can deal with the output rate.

1.1.4.5. Flow Instantiation

Regarding packets processing, it is also important to realise that there are three important types of approach:

- **Reactive flow instantiation**: every time a new packet arrives into the switch, a lookup in the flow tables is made. If no match is found, the packet has to be replicated, encapsulated and sent to the controller. Then, the controller (management applications) has to analyse the packet and make a decision and, after that, it has to send a response again to the switch. If we take into account all this steps and the fact that not all of the flow tables are standing on top of application-specific integrated circuit (ASIC) hardware with TCAM memory, the resulting overhead can have a negative impact on the network.
- **Proactive flow instantiation:** flow tables are filled in anticipation, similar to what is currently done with protocols like "routing by rumour", thus eliminating the need for triggering the controller consultation process and, consequently, eliminating the respective overhead. Thus, forwarding shall be done linearly. This strategy also has its drawbacks since, despite ensuring low-latency forwarding of packets, it is quite inflexible to situations where the receiving packets don not have a matching entry in the flow table.
- **Hybrid flow instantiation:** most of the rules tend to be implemented in advance (Proactive flow instantiation) to ensure a low latency. However, the flexibility for situations, when packets arrive and are not associated to any rule, is left open by means of reactive flow instantiation.

It is then easier to conclude that the performance and scalability of the SDN networks also depends on the granularity of the rules present at the flow tables.

1.1.4.6. Counters

Almost every internal component of the pipeline of an OFS has counters (**Table 5**) which record statistical data related to each component. The following table presents all the counters defined on the OFS specification, despite not all being mandatory.

Per Flow Table	Per Flow Entry	Per Port		Per Queue	Per Group	Per Group Bucket	Per Meter	Per Meter Band
Reference Count (active entries)	Received Packets	Received Packets	Transmit Errors	Transmit Packets	Reference Count (flow entries)	Byte Count	Flow Count	In Band Packet Count
Packet Lookups	Received Bytes	Transmitted Packets	Receive Frame Alignment Errors	Transmit Bytes	Packet Count	Packet Count	Input Packet Count	In Band Byte Count
Packet Matches	Duration (seconds)	Received Bytes	Receive Overrun Errors	Transmit Overrun Errors	Byte Count		Input Byte Count	
	Duration (nanoseconds)	Transmitted Bytes	Receive CRC Errors	Duration (seconds)	Duration (seconds)		Duration (seconds)	
		Receive Drops	Collisions	Duration (nanoseconds)	Duration (nanoseconds)		Duration (nanoseconds)	
		Transmit Drops	Duration (seconds)					
		Receive Errors	Duration (nanoseconds)					

Table 5 – OpenFlow counters

This data can be provided automatically or on demand to the controller and above managing applications, in order to deliver a clear view of what is happening in the network.

1.1.4.7. Pipeline

It is not possible to describe a strict OF pipeline which matches all the cases and implementations, since the OF specification offers several optional features and customizations. Nevertheless, a generic description can be made with the intention of exemplifying the internal operation of an OFS.

To make it more noticeable, a pipeline is presented using the following diagram (**Figure 15**) as well as a simplified description.

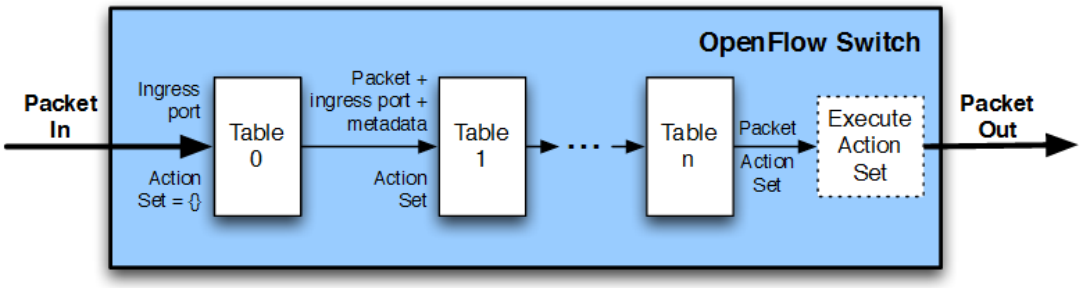


Figure 15 – OpenFlow switch pipeline[9]

When the packet arrives to the switch, its ingress port is registered and a new action set is initialized. Then, it is matched against the first table, and so on, until a flow entry match is found or, if none is found, the packet is dropped. Assuming there is a match, the instructions in that flow entry are executed accordingly with their priority. These may, for example, result in changes to the packet, updates to the match fields, updates to the action set and/or to metadata. Then, depending on the outcome, the packet can be forward to the next table, so it can be processed again, and new instructions can be performed or it may reach the end of its pipeline – in which case, the packet is processed with its associated action set and, most of the times, forwarded to an output port.

However, this is a generic description of an OF pipeline which means that distinct behaviours can be programmed and different paths can be followed by the packet during the pipeline, such as:

- cloning the packet and forward it to multiple ports;
- cloning the packet and while one copy is forwarded out the other is reinserted in the pipeline for extra processing;
- sending the packet to *meters* and/or *queues* for QoS processing where it can be dropped, halted or burst.

In each step of the pipeline, the respective active counters (**Table 5**) are updated.

[9] Retrieved from [27]

Commonly, flow tables are programmed with an entry which has the lowest priority and wildcards all the match fields. This is usually called table-miss entry and it is used to specify how to process unmatched packets. Normally, one of three options is taken:

- the packet is dropped;
- the packet is passed to another table (not possible if it is already the last table);
- the packet is sent to the controller by means of a packet-in encapsulation and the OFS will wait for the controller answer indicating how the packet should be processed (reactive flow instantiation).

1.1.4.8. Experimental Extensions

Since one of the reasons that lead to the creation of OF was to bypass the closure of proprietary systems and help to leverage research and the creation of new features, its implementation offers experimental extensions for most of the OF objects such as basic messages, OXM matches, instructions, actions, queues, meters and error.

1.1.5. Forwarding Hardware

The designation "switches" in SDN network gains a new scope since, as we have seen, these relay boxes can easily take over functions belonging to different layers and that were usually attributed to routers, firewalls, load balancers or other type of middle boxes. Everything depends on the information they receive from the controller and how it translates in terms of flow entries.

As a conceptual example[10], by looking at the partial representation of a flow table (**Figure 16**) we can see the following distinct functionalities:

- switching (first row) – the packet is outputted according to the physical address;
- routing (second row) – the packet is outputted according to its IP address;
- firewall (third row) – the packet is blocked according to the source and destination IP addresses in conjunction with the TCP port;

[10] This example does not represent OpenFlow structures in a strict way. It serves merely the purpose of facilitating the perception of the functionalities that an OpenFlow switch can take.

- load balancing (fourth and fifth rows) – packets with the same destination can be forwarded through different ports.

Ingress port	MAC src	MAC dst	VLAN id	IP src	IP dst	TCP src port	TCP dst port	Pirority	Action
*	00:01:1…	*	*	*	*	*	*	300	Port 1
*	*	*	*	*	10.0.1.0/24	*	*	300	Port 2
*	*	*	*	192.168.2.1	192.168.2.7	*	80	250	Drop
*	*	*	*	10.6.6.1	10.6.6.10	*	80	350	Port 3
*	*	*	*	10.6.6.1	10.6.6.10	*	5400	350	Port 4
*	*	*	*	*	*	*	*	1	Controller

Figure 16 – OpenFlow switch different functionalities

In OF environments there are two main types of switches: hardware-based and software-based. The first are commercial switches which tend to have superior performances and higher acquisition costs. The second are applications developed with the same intention and that can be installed on commodity hardware. From this comes its main advantage, the cost of acquisition is notably lower since there is no hardware attached and we may choose to buy cheaper white branded material or just reuse already existing hardware. In some cases, such as Open vSwitch [41], the software acquisition cost is null since some products are freeware.

Inside these types of switches, it is possible to divide them in two different categories: OpenFlow only and OpenFlow hybrid. The first one can only handle OF operations, which means that the packets have to be compliant with the protocol and they will only be processed through an OF pipeline. The second category can also handle OF operations but it also supports the conventional Ethernet switching operations. In addition to distinguishing incoming packets and forwarding them to the respective pipeline, it still allows packets to be sent from the OF pipeline to the conventional Ethernet pipeline by making use of the NORMAL and FLOOD reserved ports (previously mentioned).

In [42] is possible to have access to a fairly long list of the above mentioned OF switches.

1.1.6. Controller

The controller, also referred to as a NOS, is the software responsible for the connection between the physical and virtual network mapping, allowing the communication of management applications, which are responsible for defining the network configuration based on the policies defined by the network manager, with the OFSs. For this reason, it is considered a critical element in a SDN architecture.

In order to support this functionality, the controller has to deal with several operations, such as:

- accepting the connection request done by the OFSs,
- setting and querying configuration parameters,
- identifying the capabilities of a switch and of its OF implementation and configuration (as stated before, many of the OF functionalities are optional or may have optional features),
- verifying the status of the controller-switch connection in terms of liveness, latency and bandwidth,
- receiving and processing packet-in messages,
- adding, deleting and modifying table entries (flow, group, meter),
- pulling statistical data from counters,
- ensuring message dependencies are met (barrier messages),
- receiving notifications for completed or erroneous operations,
- orchestrating multiple controllers and/or connections.

These operations are supported by messages exchange as mentioned on the sub-chapter 1.1.4.1.

According to the chosen controller and network requirements and configuration, the controller may use centralized or distributed architecture (although from a logical point of view it is often referred to as centralized):

- **Centralized**: for small networks a single controller maybe sufficient, however, it creates a single point of failure.
- **Distributed**: for networks that demand scalability or when characteristics like resilience and availability are important. The communication between controllers is supported by East/Westbound APIs, located at the control plane, that are built having in consideration features such as: advanced data distribution mechanisms, distributed concurrent and consistent policies, fault tolerance, interoperable multi-domain and heterogeneous communications, among others.

There are many different available solutions on the market. Created in 2008, NOX [43] [44] was the first one to arise and, since then, many others were made available: POX [45] (python version of NOX), Beacon [46], Floodlight [47], ONOS [48], OpenMul [49], OpenDaylight (ODL) [50], Ryu [51], Trema [52], among many others. In [42] and [53] it is possible to access extensive listings with commercial and freeware options. As in any software market, performance and extra functionalities may vary depending on the choice made.

1.2. SCADA ICS

SCADA is a common designation for the integration of several protocols, technologies and platforms used in ICS for control and automation of production lines, whether on a small or large scale, that can be geographically dispersed over the premises or over vast areas that, eventually, can cross country or continental borders (e.g. water, gas, oil, distribution systems).

Among nowadays technologies, real-time field data can be accessed remotely from anywhere. This allows information to reach decision-makers (governments, businesses, individuals, machine learning software) in order to make data-drive decisions about how to improve their processes. SCADA systems can result in substantial savings of time and money. This is why they are currently placed at the core of several private and public modern industries, such as water, energy, transportation, oil and gas, manufacturing, food and beverage, recycling and many more.

To allow this flow of information to go from the production process and reach the financial and business management (not be confused with manufacturing operations management), it is mandatory that SCADA systems are integrated in the remainder business structure. The following image (**Figure 17**) clearly shows that integration as defined by ANSI/ISA-95 [54].

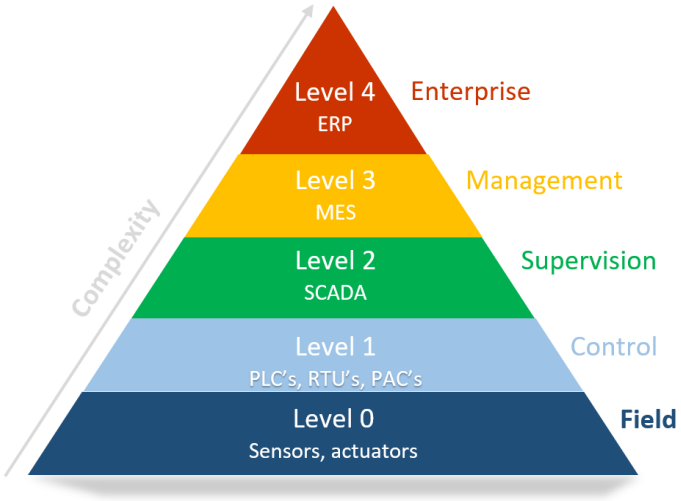


Figure 17 – SCADA integration pyramid

At enterprise level, where all the resource planning, cost analysis, sells and other financial measures are taken, it is possible to make decisions that will have a direct and immediate effect (if necessary) on production and logistics management. In turn, the process-level changes resulting from these decisions are transmitted to the supervisory level where SCADA systems will be responsible for making the necessary changes and/or optimizations on the control equipment, thus producing the necessary effect on the process line in order to meet the decisions taken at the higher level. This is only possible due to the bridge made by SCADA systems that allow field-level data to reach top decision-makers and, conversely, decisions taken by them to be reflected in the production process.

1.2.1. Architecture

The main components of a SCADA systems (which allow data monitoring, gathering and processing and also the control of machines and devices such as motors, valves, pumps) are:

- **Supervisory stations** (deployed on the supervision level) supervise processes, control and monitor slaves and often provide support for Human-Machine Interface (HMI) consoles. They are also frequently connected to other applications, such as databases, to log process data which is often used for presenting trends.
- **Control devices** (PLC's and PAC's[11] – deployed on the control level[12]) are embedded systems connected to sensors and actuators, and also to one or more master stations, being responsible for most of the monitoring and control activities. In general, these devices operate on the basis of pre-defined clock cycles. In each cycle (**Figure 18**), a scan is made to determine the input state, transmitted by the process sensors, which is then loaded into the I/O image memory area. This data is then processed based on the program loaded previously into the PLC. The result of this processing determines the state of the outputs that will be loaded into the I/O image memory area and then sent to the output modules in order to have the actuators fulfilling their tasks.

[11] Although there are some differences between PLC's and PAC's, these are not relevant in the context of this work and, as such, it is assumed a similar functioning for both.

[12] Not to be confused with the controller or control level from the SDN – there is no relation between them.

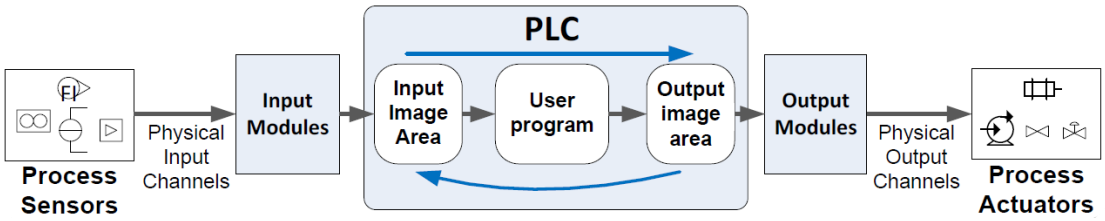


Figure 18 – PLC scan cycle (blue arrows)

- **Field devices** (deployed on the field level) constitute the physical interface with the process, providing information about it (sensors) and enabling the execution of actions affecting its behaviour (actuators).

Typically, SCADA systems use communication protocols in which on one side there is the element that manages information and communications and on the other end one or more elements that generate the information and respond to requests for communication. Within this logic, there are two main architectures, Master/Slave as used by Modbus-TCP [55] and Producer/Consumers as used by Ethernet/IP [56]. The most noticeable difference between the two is that: in the first case, the slave only sends information when requested by the master; while, in the second case, the producer (corresponding to the slave of the Master/Slave architecture) sends the information at predefined time intervals. Normally, the control devices play the role of slave/producer and the supervisory devices play the role of master/consumer. However, in both architectures, it is possible, and not unusual, to have the inverse attributions of roles or even to have the same device acting as master and slave or consumer and producer at the same time.

An example of a common SCADA master/slave system architecture for process control is shown below (**Figure 19**).

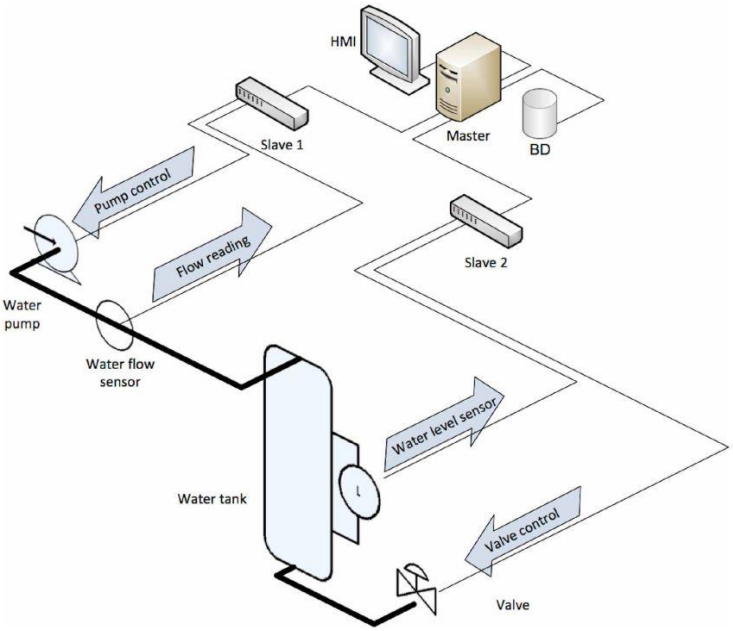


Figure 19 – Simple SCADA system[13]

1. Data from sensors (level 0) is being read by slave devices (level 1) according to its processing rate cycle.
2. According to that data, level 1 hardware may or may not send control data instructions to actuators (level 0) in order to trigger some kind of action.
3. At the same time, the master station sends data requests to the slave devices.
4. Data (from step 1) is then sent by the slave devices to the master station, which analyses and displays information based on it (e.g. it may be raw data, like the actual output sent initially by a sensor, or it can be processed data, like the result from logic and math operations made by the SCADA software) using a graphical user interface (GUI), often referred to as HMI.
5. With that information, operators can make supervisory decisions to adjust the system or to do an *ad hoc* override to the controls of the slave devices, in order to reduce waste and improve efficiency in the manufacturing process or to prevent or solve undesirable situations.

Regarding the size and the purpose of the SCADA system, the complexity of the pipeline may change and, as stated before, the assets of the architecture may slightly differ.

[13] Retrieved from [110]

1.2.2. Communication Channel

The communication channel is one of the most important aspects of an ICS and it is responsible for connecting all the layers of SCADA systems. The latest developments in this area are closely linked to the evolution and adoption of new network technologies. Regarding the scope of this work, we can divide this topic in two: before and after Ethernet.

- **Before Ethernet**

  Between mid-80s until the end of the 90s, the information from the field and control level only reached the supervision level, which was the requirement for controlling industrial processes. In the majority of cases, fieldbuses [57] like Modbus and Profibus [58] were used. This proved to be a very cost-efficient approach, since many devices could share the same set of cables in a multi-dropped fashion and multiple parameters could be communicated per device, whereas, until then, dedicated set of cables per device had to be used and only one parameter could be transmitted.

  In 1994, a major step was given in the ICS world, the Fieldbus Foundation [59] was created in order to achieve an internationally acceptable fieldbus standard. From this moment onwards, the doors for integration of multi-vendor hardware and systems were open (in spite of the "opening" of some protocols, like Modbus, only arrived years after).

- **After Ethernet**

  In early-2000, information started to be a key word in the industrial world. The necessity to bring field data to the management and enterprise levels (so it could be transformed in information) brought Ethernet to ICSs (e.g. as proven on [6], information on items/products stock has to be known by decision makers, suppliers, or others, so as to avoid production stoppages). New protocols were created [60] (in some cases old ones were just encapsulated in Ethernet TCP/IP packets, such as Ethernet/IP, Modbus-TCP or ProfiNet [61]). The adoption of Ethernet also came with advantages like increasing bandwidth, ability to use standard network hardware with lower costs and it was also an enabler for more inter-operable systems. Nowadays, despite being possible to take Ethernet until the field level, in the majority of the ICSs, it is still only used as far as the control level, due to the cost of implementation. However, in a near future, this scenario will likely change.

1.2.3. Programming Languages

The international standard IEC 61131-3 [62], defines five distinct programming languages used to create programs which run on slave devices: Function Block, Ladder, Instruction List, Sequential Function Charts and Structured Text. **Figure 20** shows a simple example of three of these languages.

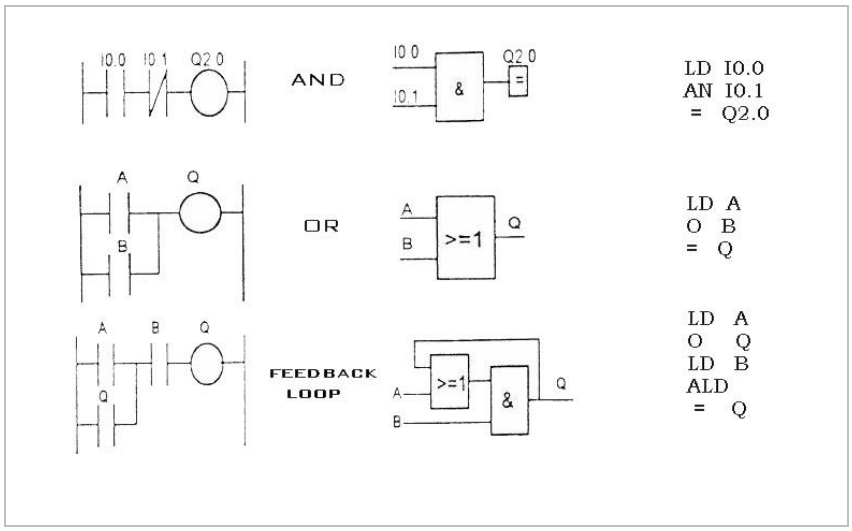


Figure 20 – Ladder, function block and instruction list example[14]

For programming the SCADA HMI, in the vast majority of cases, a drag-and-drop visual programing tool is used which allows the creation of intuitive diagrams like the one shown under (**Figure 21**).

[14] Retrieved from [115]

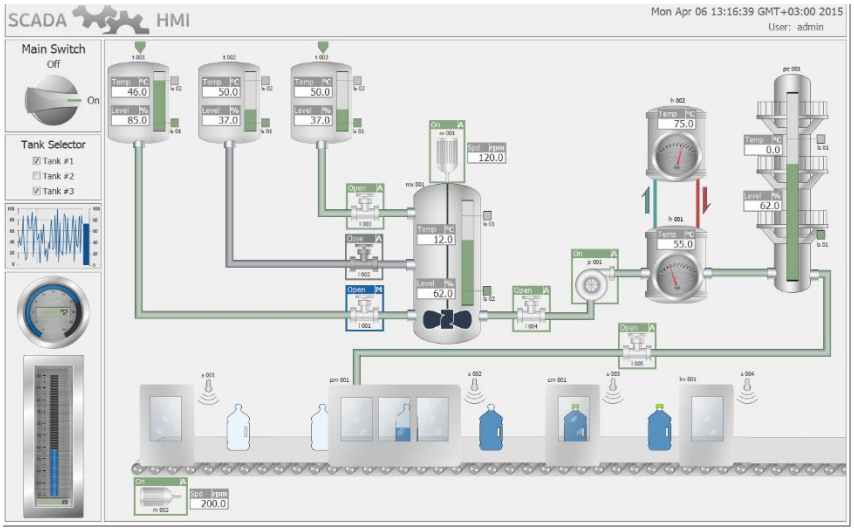


Figure 21 – SCADA diagram[14]

1.2.4. Management and Security

Especially after the adoption of Ethernet, the management of industrial networks became more difficult since its complexity greatly increased. Today, network managers have to deal with two completely different networks (ICT and ICS) which cannot be separated, but have completely different priorities, integration of multiple protocols with different demands, protocols and built-in software without security awareness, impossibility of having downtime to preform maintenance, among other challenges.

As usual, there are some proprietary solutions that claim to make network management easier but, besides being vendor lock-in, do not solve or cover the core problems.

In terms of security, SCADA systems have several problems [63], e.g.:

- Insecure design and implementation:
    - No hardware authentication, which makes it easier to connect non-authorized computers to the system.
    - No access credentials: several systems do not use access credentials, which means their security really only on the belief that no one (non-authorized) will get virtual or physical access to them.
    - No individual access credentials: some systems only allow the setting of general passwords that quickly become known by too many people.
    - Access limitations in control software are often not used.
    - Anonymous access allowed: services like Telnet and FTP often allow for anonymous login.

- No system patching: on the last years several efforts were made to improve the policy on patch management and from these efforts resulted some standards [64]. Nevertheless, once systems go into production, they will likely never be patched, due to the impossibility of having downtime on the production line, the fear that systems may become unstable, have limitations, lack support/updates by the vendor, among others.
- External network connections: nowadays almost all the SCADA systems are, directly or indirectly, connected to external networks like internet, however it is often believed they are completely isolated from the outside world. This means that numerous connections are uncontrolled.

Efforts around security issues have been made, like the creation of ISA99 Committee [65], that brings together cybersecurity experts and other stakeholders from across the globe to develop and establish ISA standards on ICS management and security. This ongoing work is being used by the International Electrotechnical Commission in producing the multi-standard IEC 62443 series, which focuses on industrial automation and control systems (IACS) security.

Likewise, the European Union countries joint efforts aim at the same goals, through European FP7 [66] projects such as MICIE [67], CockpitCI [68] or the ongoing successor ATENA [69] supported by the more recent Horizon 2020 EU Program Framework [70].

1.2.5. Generations

When talking about SCADA architecture, it is possible to identify four main evolutions, being that we are now entering the fourth stage of evolution:

1. **Monolithic**: independent systems with only one SCADA station without external connectivity.
2. **Distributed**: multiple SCADA stations could be connected to one system, increasing processing power and/or improving redundancy and reliability of the system as a whole.
3. **Networked**: systems could be spread across multiple networks, allowing for clear geographically separation. Usage of open standards as Ethernet and TCP/IP permitting the usage of off-the-shelf systems, lowering costs, bringing new players into market (typically ICT vendors) and enabling new information processing and usage. Security issues relevance arises.
4. **Internet of Things** (IoT) also referred to as **Industrial Internet of Things** (IIoT): adoption of IoT principles and related technology with considerably infrastructure costs

reducing and increasing the easiness of maintenance and integration of new elements on the industrial structure. The rising adoption of wireless technologies will provide more flexibility to the industrial process but will also demand more flexible and secure network solutions.

1.3. SDN and SCADA ICS: an overview of related work and use cases

In [71], Dong et al. propose reinforcing the resilience of SCADA networks used for smart grid applications, using a solution relying on three elements (SCADA master, SDN controller, Intrusion Detection System—IDS), which coordinate with each other in order to detect attacks and reconfigure the network so as to mitigate and overcome identified problems. Suggested use cases include the dynamic establishment of routes to transmit control commands only when necessary (to shorten the time window for tampering attempts), automatic rerouting or dropping of suspicious packets to avoid spoofing or flooding attacks from compromised SCADA elements, or implementation of network monitors to deal with delay attacks.

Irfan & Mahmud [72] propose using SDN for dynamic creation of virtual networks in order to isolate distinct traffic and hosts, and to enable traffic prioritization and secure partitioning. The concept is demonstrated using an SDN-controller proxy to create three isolated networks, which share the same physical infrastructure but have their own SDN controllers. Authors discuss the use of this architecture to improve aspects such as authentication, confidentiality, integrity, non-repudiation, and availability. A similar approach is also suggested by Machii et al. [73] as a way to minimize the attack surface by using SDN to dynamically segregate fixed functional groups within the ICS. A dynamic zone-based approach is also proposed, taking advantage of the information obtained from field devices to estimate the operation phase of the ICS (as each phase—such as start-up, normal operation, or load-change—exhibits different behaviour and communications profiles) and to calculate the optimal zone topology, deploying the needed SDN configuration in runtime. This strategy reduces the time and spatial exposure to attacks (effectively creating a moving target) and also provides the means to isolate compromised devices.

Also related to dynamic configuration techniques, Chavez et al. [74] present a security solution based on network randomization, which also encompasses an IDS with near real-time reaction capabilities. This network randomization approach assigns new addresses to network devices in a periodic basis or by request, in order to protect them against attacks that rely on knowledge about the ICS topology (such as static device addresses). The responsible controller application keeps an updated database of all the network specifications (mostly devices and real addresses),

generating overlay IP addresses for the same devices and for each flow, which are used to define the OpenFlow rules on flow tables. This way, all the traffic flowing on the network uses 'fake' overlay addresses that are periodically randomized, reducing their useful lifetime and, consequently, the time window available for any attacker to take advantage of that knowledge. The proposed IDS takes advantage of the predictable, auto-similar, traffic patterns of ICS networks for identifying attacks and triggering defence reactions (a network randomization request, which will render useless any ongoing attack using old overlay addresses). Attack detection makes use of machine learning algorithms and mathematical methods, fed and trained using OpenFlow's statistical counters.

Silva et al. [75] also describe a dynamic technique that makes use of SDN to prevent eavesdropping on SCADA networks. The intended goal is to deter attackers from collecting sequential data, which is essential for breaking encryption, identifying patterns, and retrieving useful information from the payload. By taking advantage of redundant network connectivity, a multi-path routing mechanism enables a flow to be transmitted and split over different paths (see **Figure 22** below) by resorting to an algorithm that calculates the shortest path between two devices, dynamically assigns a cost to each one, and uses an OpenFlow timer (hard timeout) to periodically reinstall new flow rules.

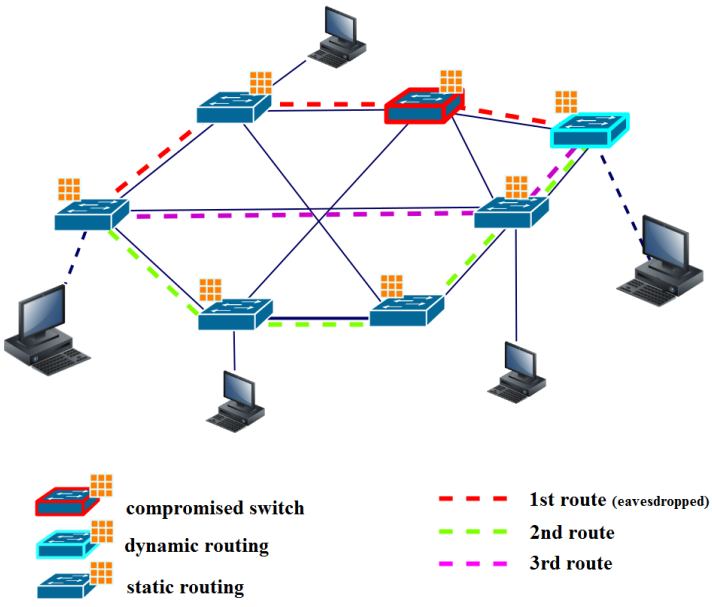


Figure 22 – Multi-flow, redundant routing for flow splitting

Genge et al. [76] propose two distinct SDN-based techniques to mitigate and block ICS cyber-attacks. The first technique (see Figure 23 below), designed for single-domain networks, attempts

to mitigate DoS attacks by rerouting traffic, using information from the SDN controller. SDN controllers feed an application that continuously monitors the state of the network links and communicates with the controller to issue flow reconfiguration operations. Once an attack is detected (few details are provided about this, though), the corresponding data flows are rerouted, in order to protect the ICS (**Figure 23**).

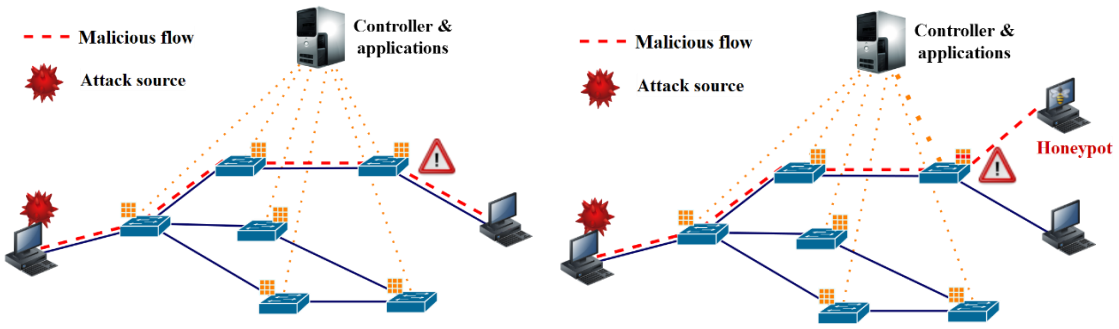


Figure 23 – A single-domain SDN-based security solution

The second technique (**Figure 24**) targets multi-domain networks, with the goal of blocking the attack as close as possible to the entry point in the network.

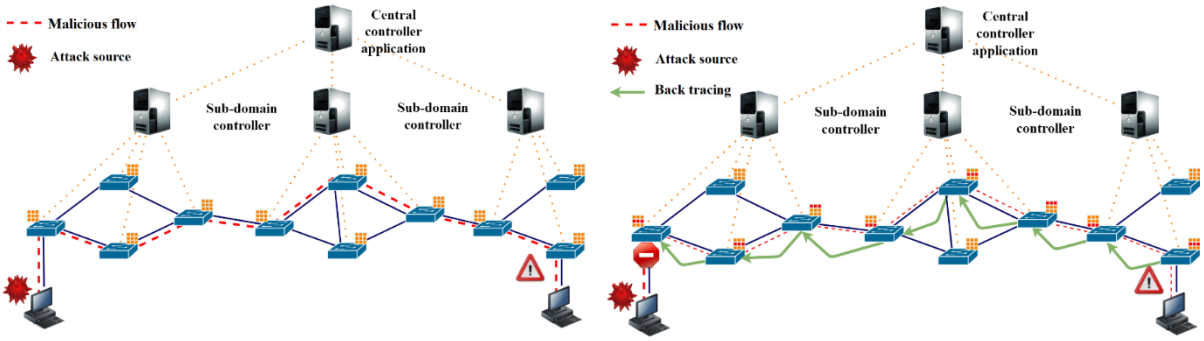


Figure 24 – A multiple-domain SDN-based security solution

For such a multi-domain network, each domain has its own OpenFlow controller, connected to a centralized security application. This application receives information from the SDN controllers, which have access to a global perspective about the network. Once an attack is detected, the security application will backtrack towards its origin by recursively issuing queries about the

related flows to identify the previously paired nodes until the original network entrance point is found.

ICS-specific honeypots and honeynets can also benefit from the introduction of SDN technologies. Honeypots are decoy or dummy targets set up to attract and detect/profile attacks. Exposed to probing and attack, these targets are used to lure and track intruders as they advance, revealing any scouting activities. Traditionally, honeypot systems live in unused address space in the system, waiting for attackers to find them, but their operation can be greatly improved by SDN, which has the possibility of turning them into a more proactive defence.

Using SDN network-flow manipulation capabilities, it is possible to improve honeypot operation and transform it into an active security component by working together with other mechanisms, such as network intrusion detection systems (NIDS). When an unauthorized activity is detected by a NIDS, the SDN controller can divert the anomalous traffic flows to an ICS-specific honeypot, such as the one proposed by Simões et al. [77]. The attacker would not be aware of this diversion and would continue the attack. Meanwhile, the honeypot will log its activity for forensics analysis. **Figure 25**, below, illustrates an example of this approach.

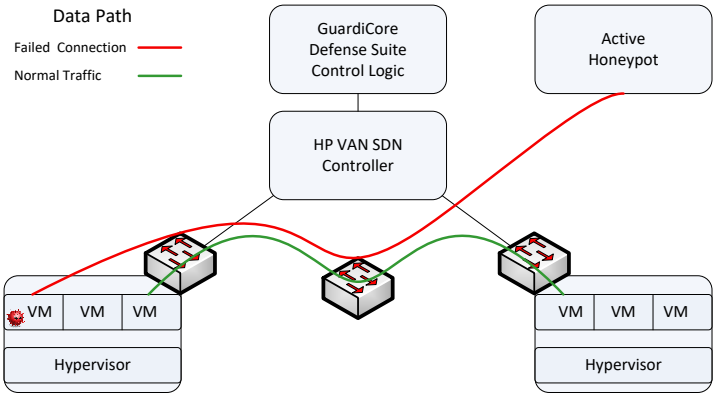


Figure 25 – Active honeypot[15]

Also, Song, Shin, and Choy [78] suggested using honeynets (networks set up with several honeypot devices) together with SDN technologies to detect scouting procedures and collect profiling information about attackers. This is achieved by providing the attacker with false

[15] Retrieved from [113]

information from the honeynet, using OpenFlow to detect the scan attacks by inspecting packets coming towards closed or unused ports, or to detect corrupt packets or sessions. After a successful detection, the infringing packet and the subsequent ones in the same flow will be redirected to the honeynet. Despite being a generic proposal, this solution can be easily ported to most ICS infrastructures.

Adrichem et al. [79] present a SDN network failover solution that should reduce the recovery time in multiple topologies. The proposed process is divided into two steps: firstly, recovery based on preconfigured forwarding rules to guarantee fast end-to-end connectivity; secondly, controllers calculating and configuring new optimal paths. Being the speed of failure detection the main key for improving the recovery time, a short discussion on different failure detecting systems is presented and the best choice is presented in a detailed mode. In [80], N. Dorsch et al. also describe their algorithms and different approaches based on SDN regarding: the efficiency improvement of network recovery time in case of link failure; the real-time processing of messages through the implementation of QoS mechanisms, based on the flexibility of SDN networks. Comparison data is presented and the IEC 61850 open source traffic compliant library, which was used on the testbed, is referred to, which may be useful.

Chapter 2 – Technological and Internship Opportunities and Challenges

Like any new and innovative technology, SDN brings with it a wider range of advantages which open the door to new opportunities, like the integration in ICSs. However, this would not be possible without a number of challenges which, if not addressed correctly, could become disadvantages.

2.1. Technological Opportunities

Due to its architecture, SDN can improve and simplify several aspects of SCADA networks by means of:

2.1.1. Configuration and Management

- Allowing a unified view of the network, regardless of the hardware.
- Providing the ability to write customized and best fitted network management applications that can be reconfigured from a centralized point.
- Enabling self-configuring networks that can reconfigure themselves in a dynamic way, in order to meet changing service requirements, such as new layout on the ICS or the reprogramming of control elements that begin to generate more traffic over some network links, among others.
- Automatically re-configuring a switch when, by some reason, it needed to be reset or replaced. In ICS, switches are often left without any kind of configuration (simply making layer 2 forwarding) so that, in case of need, they can be rapidly swapped without the overhead of having to be reconfigured again.

2.1.2. Security

- "Denying-by-default" traffic, which can be a very simply way to block unexpected potentially dangerous traffic coming from unauthorized machines, such as attempts to over-write data on PLC's memory (allowed by the lack of authentication on the most used ICS protocols).

- Supporting a higher level of granularity based on the multiple flow entries match fields, SDN can filter undesirable traffic at line rate as well as detect new unexpected traffic.
- Detecting changes in traffic patterns, by using the above mentioned granularity capabilities or by means of statistical treatment based on the numerous counters, as a way to detect attacks to ICSs. (e.g. DDoS frequently involve random spoofing of IP and ports, the increase of the number of used TCP/IP ports and addresses may reveal ongoing issues. The number of packets per flow counter can also be used for DDoS detection, since such attacks usually rely on the transmission of a reduced number of packets from a large amount of sources; the number of active flows or packet matches can also be used to detect DoS. The number of single-flows can also be used as a way to detect attacks, since it is possible that the number of unpaired flows increases dramatically at the beginning of a flood attack. This can be calculated on a per interval basis, after subtracting the paired flows from the total.)
- Allowing flexible configurations that can change dynamically when attacks or strange behaviours are detected, in order to neutralize or mitigate attacks. For example, flow-based forwarding can be used to increase the efficiency of a reaction, being used to isolate or divert flows, instead of simply blocking an attack. This is useful to improve existing security techniques such as dynamically diverting attackers to honeypot systems as soon they are detected.
- Using proactive flows instantiation may close the opportunity window to some attacks. ICS communication is normally quite deterministic, hence allowing to predefine in advance the flow entries with a high level of granularity.

2.1.3. Reliability and Availability

- Providing a centralized global view of the network which allows:
    - self-healing solutions, which can actuate quickly on case of link or node failure, ensuring the availability of critical systems;
    - faster converging to optimum resources allocation according to the demand for network resources;
    - easily managing of end-to-end paths.

2.1.4. Innovation

- Allowing high fidelity test environments. Since the backbone of SDN networks are purely software, they can be emulated in order to help research, testing and verification.
- Enabling new services to be introduced quickly and at a lower cost, without the need to change infrastructure and with minimal disruption to the ICS. Quick time to market is also one of the most important characteristic in competitive markets.
- Improving the computing capabilities of the network control hardware, which allows the inclusion of more complex algorithms (e.g. path optimization, multi-path discovery).

2.2. Technological Challenges

Being a new technology, which is still not stable, SDN also carries some sensitive points, such as:

- Specified features from the last two versions of OF may still not be fully implemented or may present some bugs.
- It is still not completely clear what functionalities should reside on the controller and what should be on the switches.
- Multiple instances of the network controller have to be provisioned in order to avoid a single point of failure, since the control of the network is centralized.
- It may send malicious control messages to the network, if a controller is compromised.
- Prone to the existence of software bugs, especially on the controller and management applications.
- Excess of granularity in rules may affect the network scalability. It is important to find a balance between both.
- Latency variability. Being typically a very strict parameter in ICSs, it is important to attain a good structuring/programming of the flows entries, since several factors can influence the individual latency of each. (e.g. number and type of tables, number of flows in each table, instruction type, number and type of actions, “internal recirculation”).

2.3. Internship Aims and Constraints

The research work done during the internship focused on the prospection, design and development of mechanisms that could take advantage of the potential synergies between the two technologies, SDN and ICS SCADA, as well as of the creation of a physical test environment (testbed) where those mechanisms could be tested and validated.

Hence, it was necessary to study and understand the functioning of both technologies to be able to perceive the strengths and weaknesses of each one as to reach theoretical conceptions where the conjunction of the two could result in added value.

After a detailed analysis of the SCADA ICS, it was defined that the objective would be to combine the dynamism, flexibility and statistical data provided by SDN with the determinism of SCADA ICS networks, in order to take advantage of this interactions in terms of security (detection, analysis, reaction), resilience and fault tolerance.

As expected, throughout the research some constraints had to be overcome.

Firstly, since these technologies are not approached in depth (or not at all) in any of the curricular subjects, there was a lack of previous knowledge on SDN and SCADA ICS (which implied a greater effort to gather relevant research literature). There was also an exceeding effort to fully understand both technologies as well as to define the level of expertise required to be able to analyse possible synergies between both technologies.

Moreover, due to SDN being a relatively recent technology, there were some inconsistencies between scientific articles and other academic/technical documentation related to SDN, as well as some difficulty in perceiving which article referred to attainable applications according to the present and which were merely theoretical, having little or no tried or tested results. Furthermore, at different times, there was a need to cope with situations of software failure due to instability and/or incompatibilities.

The scope of the research as well as its associated requirements presented heterogeneous challenges by implying different fields of expertise. Despite proving quite enriching, this diversity led to breaches on the path to the main purpose of this research, which proved to be very time consuming.

All these constraints resulted in a steeper learning curve and a slower pace of progress.

2.4. Methodology and Project Scheduling

It was necessary to do some research on the work methodologies most frequently used in the areas of software development in order to organize the work that would have to be developed during the investigation and as to understand which would best fit the expected needs. Since no dependency on external factors to the investigation was foreseen, the choice was to adopt an approach based on the back-stepping waterfall model. The adaptation made to this model (**Figure 26**) was intended to make it more dynamic, by allowing a slight overlap of tasks and allowing at any time to go back to previous steps for verification or redefinition.

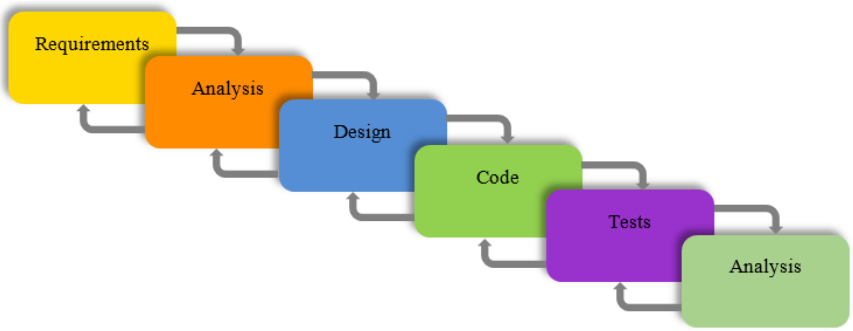


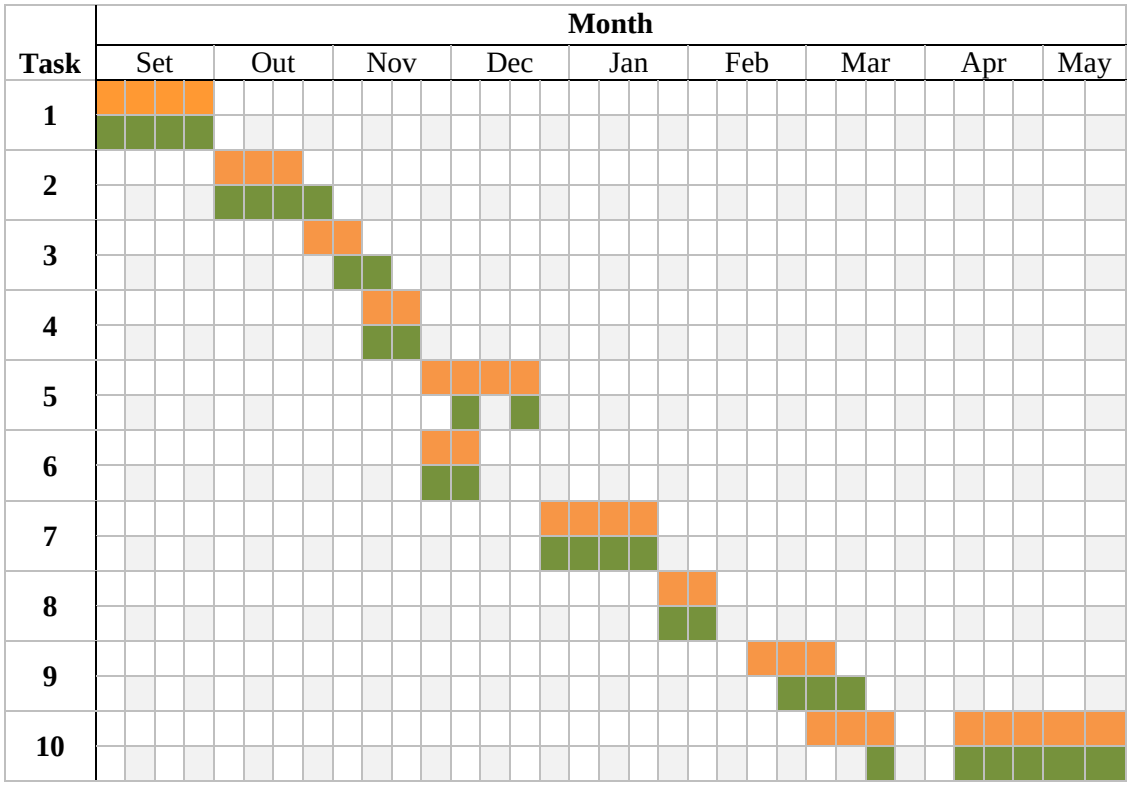
Figure 26 – Adapted stepping-back waterfall methodology

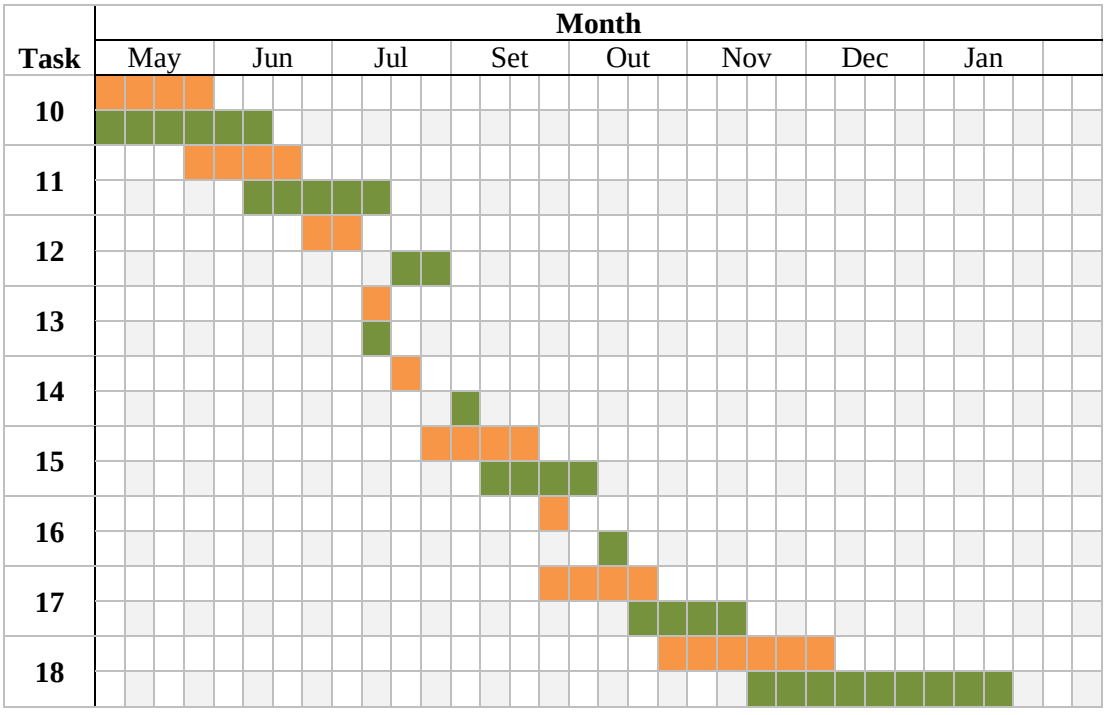
Therefore, the workflow based on the adopted model follows the following path: the tasks are performed in sequence and the outcome of one task acts as the input for the following task. However, since this is a research work and focuses on areas of which there is little knowledge, it is only natural that one would need to look slightly further ahead to help make some decisions regarding the conclusion of the ongoing task. Being so, a task may start slightly earlier, in parallel with the conclusions of the previous one. After finishing a task, if its outcomes so determine, it is possible to step back to previous tasks in order to produce the necessary changes. An exception to this methodology occurred with task 6 which was not scheduled in advance and had to be done in parallel with other tasks.

The tasks performed throughout this research, which were scheduled (**Table 6**) in compliance with the methodology presented above, were:

- acquaintance with the work theme, project and the state of the art (task 1);
- preliminary proposal for solutions to develop relating ICSs and SDN (task 2);
- writing the first version of the testbed architecture (task 3);

- discussion and refinement of the final version of the testbed architecture (task 4);
- implementation of the SDN part of the testbed (task 5);
- writing of the scientific paper entitled “Security implications of SCADA ICS virtualization: survey and future trends” (task 6);
- intermediate report documentation and writing (task 7);
- intermediate presentation creation (task 8);
- refinement of the proposed solutions arising from task #2 (task 9);
- development and integration of those solutions (task 10);
- validation of the developed solutions (task 11);
- prospection of relevant use cases (task 12);
- writing of the scientific paper entitled “Leveraging Virtualization Technologies to Improve SCADA ICS Security” (task 13);
- analysis and selection of the use cases to implement (task 14);
- implementation of the ICS testbed (task 15);
- integration of the SDN and ICS parts of the testbed and general testing (task 16);
- validation of the solutions making use of the final testbed (task 17);
- final documentation and writing of the report (task 18).





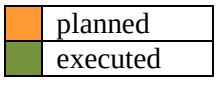


Table 6 – Tasks schedule

Chapter 3 – Research and Development

3.1. Testbed

Being the testbed one of the essential elements developed throughout this work, since it constituted the basis for the rest of research work, a detailed description is next presented. Its main purpose is to provide a development and validation environment for the ideas and concepts to be developed. As such, its architecture should ideally encompass a number of representative ICS/SCADA elements, integrated within the scope of an emulated industrial process. In this perspective, a hybrid testbed scenario was developed and built from scratch, incorporating real SCADA equipment, as well as emulated and virtualized components.

3.1.1. Architecture

The physical simplified testbed architecture is described by **Figure 27**. It is based on a network topology that was planned to accommodate an SDN-aware scenario.

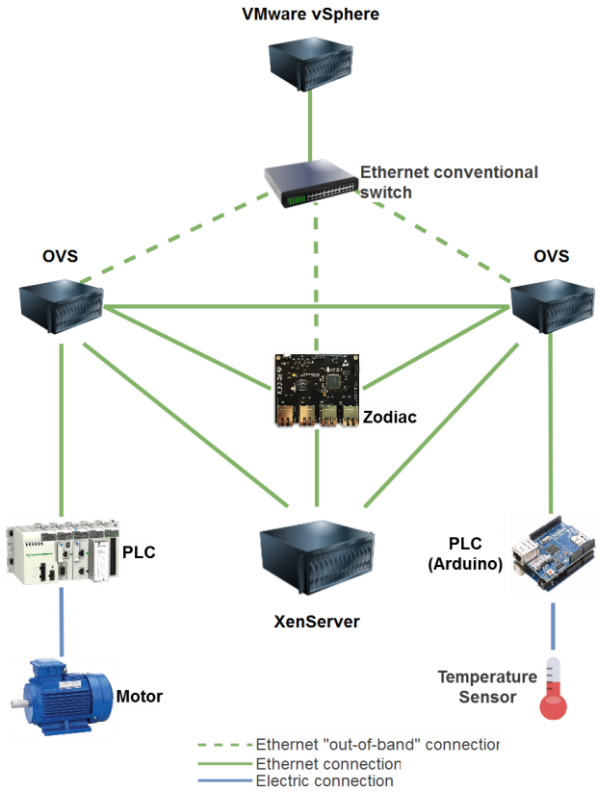


Figure 27 – Testbed simplified physical diagram

From a physical point-of-view, the testbed includes a number of diversified components, namely:

- Bare-metal hypervisor running VMware vSphere 6.0, which hosts the SDN controller, has the following specifications:
    - CPU: Intel Xeon X3200 @ 2.13GHz
    - RAM: 8GB DDR2 800Mhz
    - HDD: 120GB 7200rpm
    - NICs: 2x1Gbps
- Bare-metal hypervisor running Citrix XenServer 7, which hosts the SCADA control (HMI) and other virtual machines used for testing, has the following specifications:
    - CPU: Intel Core2 6300 @ 1,86GHz
    - RAM: 8GB DDR 667MHz
    - HDD: 120GB 7200rpm
    - NICs: 4x100Mbps, 2x1Gbps
- Two similar machines, running CentOS 6 operative system and used as OFSs running OVS, with the following specifications:
    - CPU: Intel Xeon X3200 @ 2.13GHz
    - RAM: 8GB DDR2 800Mhz
    - HDD: 80GB 7200rpm
    - NICs: 6x1Gbps
- A Northbound OpenFlow Zodiac FX switch
- A 3COM Ethernet 4400 switch
- SCADA process equipment, which will be discussed in one of the next sections

From the logical perspective, the testbed architecture is organized as shown in **Figure 28**.

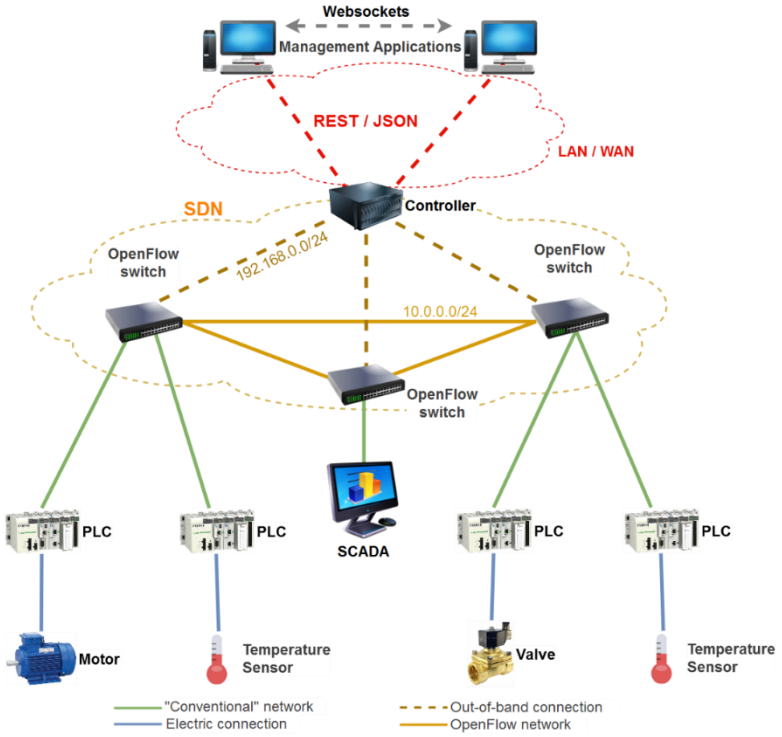


Figure 28 – Testbed logical diagram

There are three distinct areas: the SDN domain composed by the OpenFlow switches and the controller; the LAN/WAN which is where the management applications are held; and the ICS which uses the SDN to establish the communication between the control and the supervisory elements.

The SDN domain is composed of two separate networks: the first (10.0.0.0/24) is the production network and uses the OpenFlow communication protocol; the second, referred to as out-of-band for being in a different range of IP's (192.168.0.0/24), is used for southbound communication between the switches and the network controller.

Northbound communication, which are establish between the controller and the management applications, are done through REST/JSON.

3.1.2. SDN Testbed Architecture: an overview

3.1.2.1. Implementation and Deployment

Since, in the laboratory, there was no adequate infrastructure where the intended tests could be carried out, it was necessary to implement one as it was also one of the objectives of this internship. This required a series of steps which are summarized below.

- Identification of available "on premises" hardware that would suit the intended purposes. Since no new hardware was bought and there was no specific hardware for this project or SDN technologies, a selection among the existing hardware needed to be made. Some components needed to be gathered from different machines and assembled together in other to achieve acceptable requirements for each machine.
- Preparation of active and passive and interconnection of all the testbed components.
- Installation and configuration of virtualization solutions:
    - VMware vSphere v6, for being quite a complete option and being the solution with which the remaining lab researchers were more familiar with since it was already used in the laboratory for other purposes.
    - Citrix XenCenter 7 – for hardware compatibility issues. Some of the available hardware was somewhat older and was not supported by the other chosen hypervisor.
- Creation and deployment of distinct virtual machines (VMs) with different specifications in order to meet the requirements of different services and purposes. A VM containing the SDN controller was deployed into the VMware hypervisor. This VM was running a container-based virtualization solution (Apache Karaf) in order to easily allow the installation of ODL features as well as to simplify the try-out of different ODL versions. Another VM was created and deployed in the same hypervisor so as to host Mininet. Other VMs were created and deployed into the Citrix hypervisor and used as hosts for testing SDN capabilities.
- Execution of preliminary tests in Mininet [81] emulated environments so as to test different solutions and configurations in a more expedite way (e.g. from first understanding how SDN technologies work from an empirical point of view, to test different integrations, to try to identify compatibility issues and other similar situations).
- Installation and configuration of the machines intended to host the OFSs.
- Configuration of the OVSs and ODL controller.
- Undertaking of tests to ensure the correct operation of the testbed in terms of physical connections and software communication.

The SDN part of the testbed is mainly composed of two elements, the switch and the controller.

Being such a recent technology, one of the main criteria of selection was the maturity of each solution in order to try to avoid unstable and buggy scenarios. Also, other factors have been taken into account, such as: being freeware, using open standards, being compliant with the more recent versions of OF and being well documented. The choice for the switch fell on Open vSwitch and for the controller fell on OpenDaylight.

At the final stage of the internship, it was possible to acquire an OF hardware switch (Zodiac) which was also integrated into the testbed.

3.1.2.2. Open vSwitch

Open vSwitch is, by far, the OpenFlow software switch more used among SDN developers. There is also a close collaboration between developers of SDN (particularly from ODL) and developers of the OVS. Just by itself, this would be a strong reason to choose OVS as the OFS to be used in this testbed. Furthermore, it is the solution which appears to be better documented and has more support available (not only from the developers but also from the community). A good indicator of maturity and robustness is the fact that OVS is also an integral part of some hypervisors and cloud managers such as XenServer, Xen Cloud, VirtualBox, KVM, OpenStack, openQRM, OpenNebula, oVirt, and is also full compatible with the main Linux distributions being available as packages, or similar, for Debian, Ubuntu, Fedora, CentOS, FreeBSd, etc. In terms of support and compatibility, at the time, OVS supports most of the OF v1.3 and 1.4 specifications and, taking into account the feedback available on various websites related to the area as well as references in scientific papers, appears to be compatible with the most known SDN controllers. Being the switch used by Mininet tool is another good reason for choosing it.

Although most of the settings are made through the controller, there are always basic settings that have to be made initially, as also the need for some kind of debugging. The commands mainly used for those settings and debugging and which are available through a local console are:

- **ovs-vsctl**: is used to configure the OVS switch operations such as port configuration, adding/deleting bridges, defining VLAN tagging, configuring the connection mode etc.
- **ovs-ofctl**: is used to monitor and administer OF switches allowing the access to active features, configurations, table entries, among others.
- **ovs-appctl**: is used to manage and control the several daemons which compose the OVS.

3.1.2.3. Zodiac

Over the time that lasted this research, there was no OF hardware switch nor was there a possibility to acquire one due to the inherent costs. However, already at the end of the research period, it was possible to purchase a switch which appeared as a crowdfunding project [82] from an Australian networking company that claimed it would create the cheapest OFS on the market. This switch called Zodiac (**Figure 29**) is an open source hybrid switch that fits the needs for basic OF laboratory testing.

- 4 x 10/100 Fast Ethernet ports with integrated magnetics
- Command line interface accessible via USB virtual serial port
- Amtel ATSAM4E Cortex M4 processor
- Support for OpenFlow 1.0, 1.3 & 1.4
- 512 entry software flow table
- 64KB frame buffer with non-blocking store and forward
- 802.1q VLAN support for 64 groups from 4096 IDs
- Per port based 802.1x authentication
- 802.1w Rapid Spanning Tree Protocol (RSTP)
- 16 ACLs per port
- 2KB jumbo frame support
- QoS / CoS prioritisation with 802.1q tag insertion
- Auto MDIX with X-over detection
- Per port link and activity LEDs
- High speed SPI expansion header
- USB powered
- Ultra-small size of only 10 cm x 8 cm

Figure 29 – Zodiac switch[16]

Although, at the current date (v0.72), has got some limitations in terms of OF support, such as not supporting group or meter operations, it can still support most of the basic OF specifications. Having an open source firmware makes it even more interesting for laboratory usage.

The existence of a forwarding box with such a low cost, open source and with the ease of integration in an SDN, like it was possible to observe during its integration in the testbed, proves that the objectives set when creating this type of networks are being reached.

[16] Retrieved from [114]

3.1.2.4. OpenDaylight

In spite of the existence of controllers with a smaller learning curve – which makes them better for quick prototyping, like POX –, there was a will to use technologies that were recognized by the industry, so that any upcoming results from this investigation could be more reliable and have a greater acceptance. Bearing this in mind, the choice was the ODL controller. This controller is supported by the main network players [83] and is highly flexible, permitting centralized and distributed architectures and offering standard REST/RESTCONF APIs for northbound communications.

As stated on [84], having a large number of deployments in companies such as Orange, China Mobile, AT&T, T-Mobile, Comcast, KT Corporation, Telefonica, TeliaSonera, China Telecom, Deutsche Telekom, and Globe Telecom… ODL comes forward as the leading controller of SDN ecosystem. The number of contributing individuals has recently exceeded 500, making it one of the fastest growing open source projects ever. Considering the current dimension of the SDN market, these data seems to indicate some maturity, especially in comparison with the other available solutions.

In terms of architecture, the modular structure of ODL allows us to only load useful modules according to the intended goals and required functionalities. Although there are several modules available and new ones appearing over time, there is one – DLUX – which is quite useful for research and development environments. DLUX offers a GUI and natively includes two useful tools:

- Topology (**Figure 30**) – it allows us to visualize the detected network elements and the current network topology.

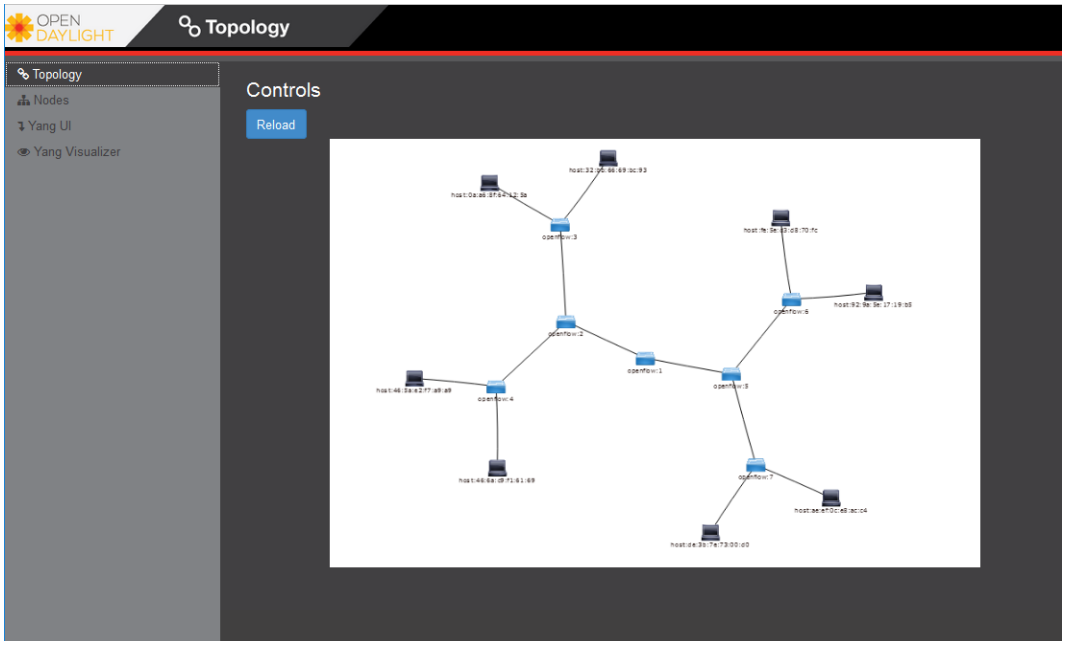


Figure 30 – DLUX: Topology

- Yang UI (**Figure 31**) – by reading and processing YANG modules, it provides us a quick access to the operational and configuration ODL databases (as well as all the ODL structures which are defined using Yang modelling). Moreover, among other functions, Yang UI also allows us to find out what the available REST/RESTCONF APIs and structures are, and enables us to validate REST/RESTCONF requests.

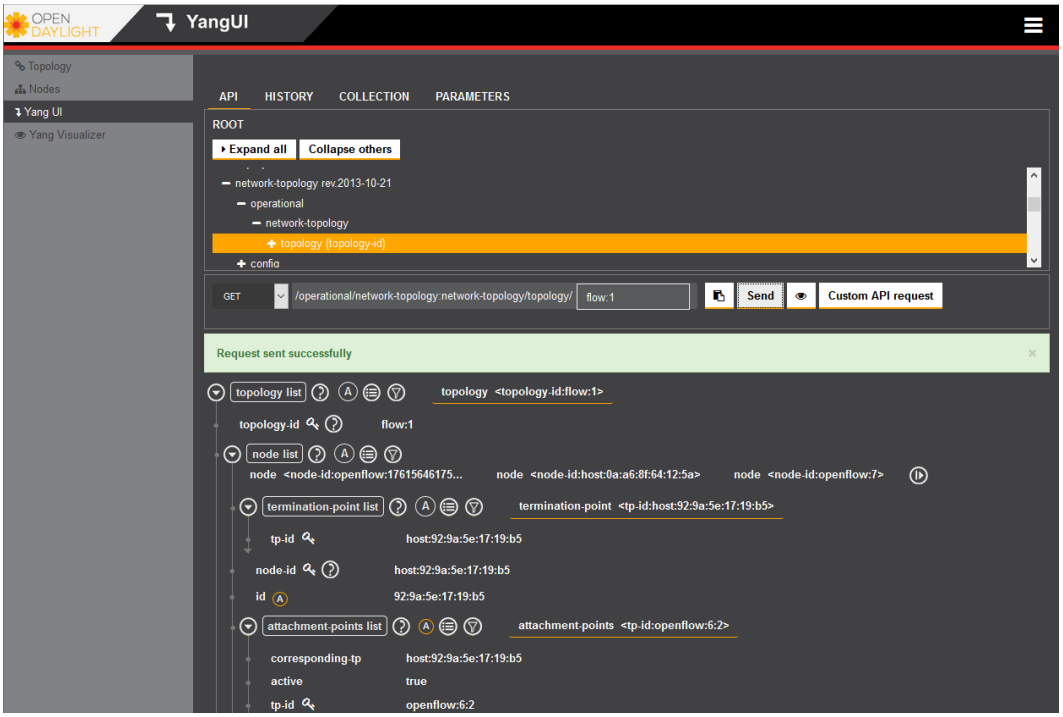


Figure 31 – DLUX: Yang UI

ODL is also well known for having several southbound APIs and plugins available (such as, OpenFlow, NETCONF, BGP, OPFLEX, SNMP, LISP, IoT HTTP/CoAP, LISP) as well as for having standard, well known and easy to use northbound APIs (namely, NETCONF, REST, RESTCONF and AMQP).

3.1.2.5. Management Application

The possibility of easy integration of management applications in SDN is often mentioned as one of its main assets. Management applications can comprise a variety of functions, from simple monitoring of network state, to firewall, to load balancer, to IDS, to QoS optimizer, to paths discovery, among many others. After analysing the type of possible applications and their characteristics, it was concluded that these could be divided into two distinct groups: the first, backend applications that run continuously without human intervention and which can actively change the network programming, according to their algorithms and data obtained from the network or even from other sources; the second, frontend applications that, through their GUI, interact with a user and propagate the repercussions of this interaction to the network.

In the scope of this work, two applications, representing both types, were developed as a PoC.

The first one is a backend application developed in JAVA that collects statistical data from the network and then analyses it. This data is collected through RESTCONF communication (**Figure 32**) with the SDN controller, using GET operations which are executed in a predefined interval, and JSON format for data request and response. The controller receives this data from the OFSs and stores it in the operational database. It is the management application responsibility to request data from the correct database.

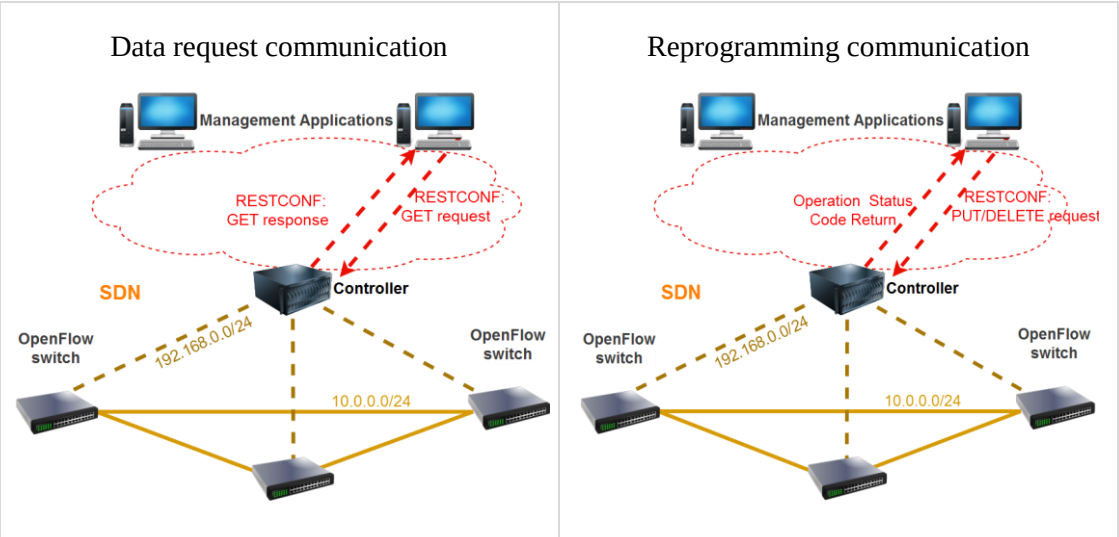


Figure 32 – Management application communication

The application has autonomy to intervene in the network by sending instructions, by means of RESTCONF PUT and DELETE operations, to reprogram the network, such as blocking a flow, whenever necessary, according to the results of the analysis performed. These instructions should target the configuration (config) database.

The second one is a frontend application developed in HTML, CSS and JavaScript, which offers a network monitoring interface with a focus on security, such as those might be used in a security operation centre (SOC). This application does not communicate directly with the network controller but rather with the first application presented, which in turn can communicate with the controller, if necessary. The communication between applications is established using websockets (**Figure 33**) which are used for constant communication (sending statistical data and alerts from the first application to the second, and requesting specific data or sending network instructions from the second application to the first).

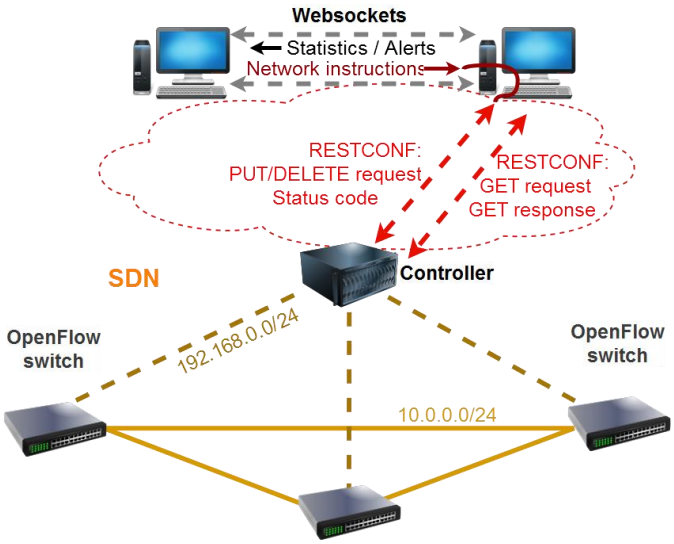


Figure 33 – 2nd management application communication

In **Figure 34**, it is possible to see the GUI from the second application which allows: the visualization of network statistics such as the number of flow packets and bytes; the visualization of alerts sent by the first application; and the possibility for the user to send reprogramming instructions to the network, such as flow blocking.

Figure 34 – Management application GUI

Given the technologies used, any of the applications can be located in the LAN or WAN, thus allowing monitoring and control of the network in the most diverse scenarios.

3.1.3. ICS

3.1.3.1. Industrial Process

The SCADA process implemented in the testbed is based on a recently implemented industrial process (**Figure 35**) of a national steel company. However, it assumes components and a mode of operation that is transversal to several industrial processes. In this particular case, the testbed is based on the process described below.

In order to save costs and increase environmental sustainability of the production process, ray concentrating panels were installed to store thermal energy from renewable sources and integrated in the existing industrial process. This energy is stored in thermal oil, which is then applied in the cleaning process of metal products, which takes place in a cleaning booth that incorporates gas burners. The booth has two areas with independent temperatures: bathing area (water) and drying area (air). As the drying zone needs higher temperatures, in order to optimize the cost of energy consumption, the thermal energy is first consumed in the drying process. The existing burners use an ON/OFF working mode so as to maintain the temperature within a previously specified range of temperature levels. Thus, when the temperature is below a certain level, the burners are turned ON and, when it reaches the intended level, the burners are turned OFF.

In order to interconnect the new energy source to the mentioned booth without breaching the vendor warranty, the following solution was implemented:

1. As to control the system's inlet temperature, there is a pump controlled by a frequency inverter that makes the thermal oil circulate through the solar panels – the longer the retention time of the oil in the solar panels the higher the temperature it reaches. If the oil temperature is too low, the speed of circulation of the oil is decreased so that it remains more time in the panels. If the temperature is high enough, the speed of circulation of the oil is increased.
2. The oil flow directed to each area (drying, bathing) is controlled by two proportional solenoid valves which regulate the entrance rate of oil flow into each of the areas, according to the required temperature in each of the areas, to its current temperatures and also to the oil temperature.

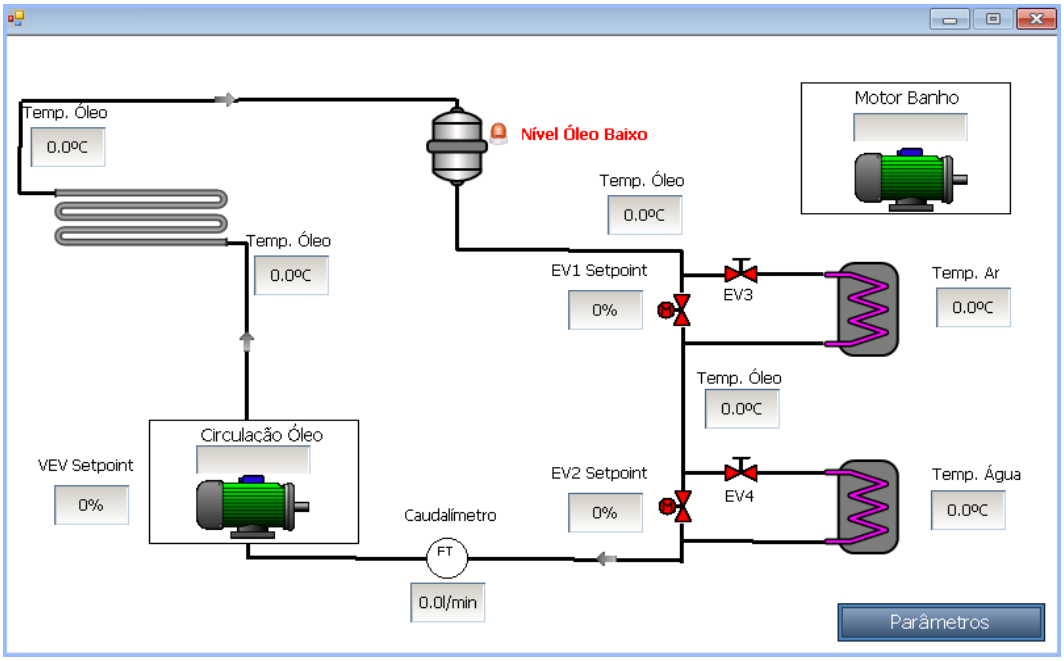


Figure 35 – HMI from a production industrial process

3.1.3.2. Implementation and Deployment

The presented testbed (see below, **Figure 36**) attempts to simulate the circulation of the thermal oil through the circuit[17]. In order to implement and deploy the testbed, it was required to go through a series of steps which are summarized below.

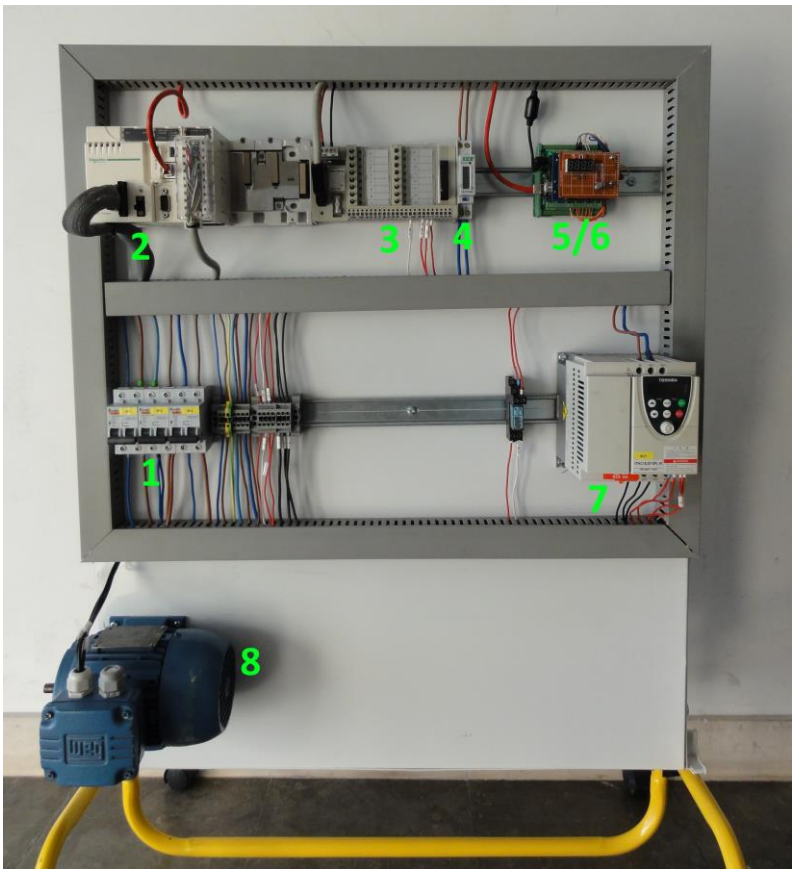
- Identification of the existing material in the laboratory, followed by an analysis of the necessary material to prepare a testbed. On premises there was only a Schneider Modicon M340 Discovery Kit composed by a M340 PLC with a digital I/O module and a Telefast ABE7-TES160 system used for simulating inputs and outputs and for quick connecting the I/O module to the operative parts.
- Construction of a panel to support the components of the testbed.
- Installation of the equipment in the mentioned panel.
- Configuration and wiring of active equipment:
    - **Schneider M340 PLC**[18] and **Telefast ABE7-TES160** – The connecting cable between the two had to be redone since it was not prepared to work with the inputs and outputs made available on the I/O board. This was a time consuming task since there was no

[17] Contingent on the constraints imposed by the type and quantity of the available material.
[18] The programming of this equipment was carried out with external aid.

documentation available regarding the pinout of the I/O module and cable. Subsequently, the PLC had to be programmed according to the intended purpose.

- **Arduino Uno** – Since there was only one PLC in the laboratory, there was the need to search for alternatives, in order to make the testbed more complete. Hence, an Arduino Uno was installed which, through the use of the W5100 Ethernet shield and a Modbus library, was used as if it were a second PLC. The use of this equipment had the advantage of allowing analogue I/O, since the Schneider PLC only allowed digital I/O. A potentiometer was used to simulate the temperature input and a 7-segment display with 4 digits to allow the input value to be displayed. Subsequently, the Arduino had to be programmed according to the intended purpose.
- **Toshiba VFNC1S Variable Frequency Drive (VFD)**[18] – This equipment had to be parameterized in order to produce the desired effect on the motor in accordance with the input connections.
- **Energy counter** – This equipment only needed to be wired since the required programming was carried out in the PLC, so it would correctly process the impulses sent by the energy counter.
- **Motor** – This equipment was wired to the VFD.

- Creation, configuration and deployment, into the VMware hypervisor, of a VM to host a SCADA system.
- Installation and configuration of a SCADA system based on the freeware software Rapid SCADA [85].
- Development of an HMI to enable the visualization of the current state of the ICS.
- Virtualization of the already existing system containing the Schneider software for PLC programing so as to migrate it to the Citrix hypervisor so as to safeguard the software and its legal licenses, given that the hardware in which it was installed was presenting signs of failure, for the sake of energy saving and to test the reprogramming of the PLC from a remote point over the SDN.



1.	Safety switches	**3.**	I/O board	**5.**	PLC 1	**7.**	VFD
2.	PLC 2	**4.**	Energy counter	**6.**	Temperature sensor	**8.**	Motor

Figure 36 – ICS testbed

A specific description of the industrial process implemented in the testbed (**Figure 37**) is presented below:

PLC-1 reads the value sent by the temperature sensor. The temperature value is then sent by PLC-1 (acting as slave device) to PLC-2 (acting as master device) when requested by the latest. This horizontal communication channel is done over Modbus TCP. The value is stored in a memory register in PLC-2. Then, PLC-2 will choose one of the following options dependently of the temperature value, in accordance with the algorithm that has been programmed:

- if it is less than 50 degrees, it shuts off or keeps the engine switched off;
- if it is between 50 and 100 degrees, it starts the engine in medium rotation speed;
- if it is above 100 degrees, it accelerates the engine to the maximum rotation speed.

The motor will only start if the protection switch is on. This parameter is kept in a memory registry of PLC-2.

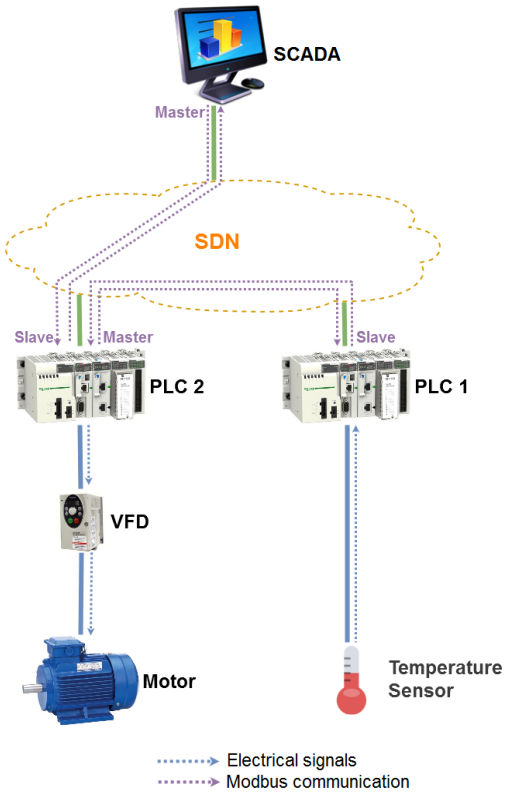


Figure 37 – ICS testbed communications

During the process, the HMI (**Figure 38**) continuously requests data update to the PLC-2 by using Modbus communication. In this case, the PLC-2 acts as slave and the HMI as master. As seen in the figure below, HMI receives updates of the motor security switch, of the current state (speed) of the motor, of the temperature value and of the power consumption.

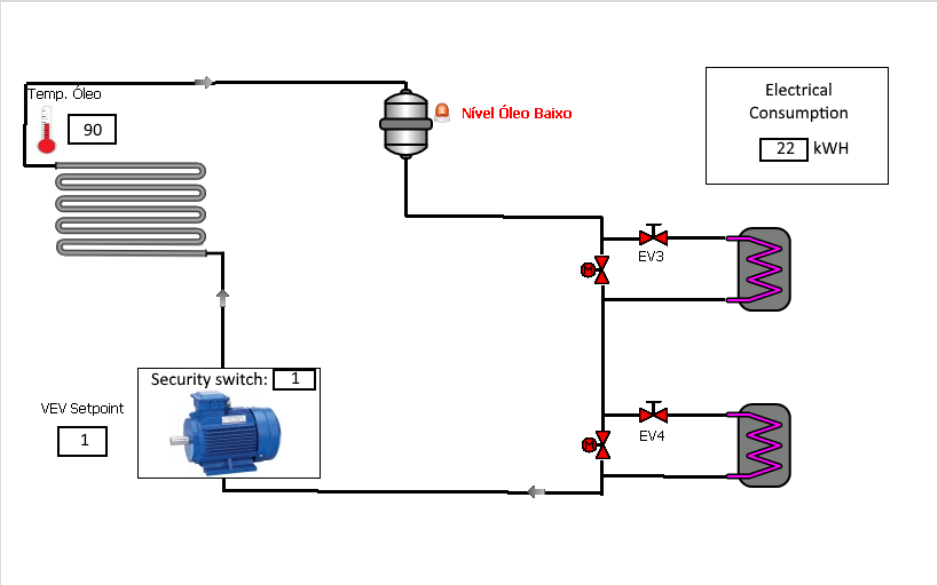


Figure 38 – Testbed HMI

As presented on Figure 37, the PLC-2 does not communicate directly with the motor, instead it communicates with a VFD which, according with the inputs activated by the PLC (through its outputs) and with its pre-defined parametrization, will control the speed rotation of the motor by means of varying the frequency of the electric signal. However, for the sake of simplification, throughout this report this interim link will be omitted.

3.2. Toolset

During the research, several tools were used, for testing, deployment, support and implementation of features and functionalities. A list of those tools as well as a short description of its usage is presented below:

- iPerf [86]– this is a freeware tool for network testing. Its main purpose is to gather values of network metrics (throughput, bandwidth, jitter, packet loss), so that analysis can be made regarding the performance of the network. This is achieved by running the application in two deserved points of the network, one of the instances serving as a client and another as a server. The client establishes a preconfigured connection with the server and from there the measurements are taken. It is possible to define several communication parameters, such as: protocol (TCP/UDP), port, number of streams, packet size, buffer size, window size (TCP), data to send (random or a specific file), among others. During this research, iPerf was used not only for network performance analysis but also for generating flows for OF testing.

- Wireshark [87] – this freeware tool is a network protocol analyser. It allows capturing network live traffic for analysis. It offers a quite intuitive GUI, making it easier to look into each packet. It also supports customized filters to allow only the capture or analysis of the intended traffic. During this research, Wireshark was used to confirm what type of traffic was arriving at certain machines in order to help debugging abnormal situations as well as to check/confirm the size, payload and bytes of Modbus packets.
- Karaf [88] – this freeware tool is a lightweight container, which can be used in a standalone mode or in a complex enterprise system by making use of all the features it includes. This tool was used since ODL is distributed inside a karaf container. Being so, this tool was used to run ODL and to install some of its features (modules).
- Unity Pro [89] – although not being freeware, this tool was included in the Schneider Modicon M340 Discovery Kit and it is used for configuring and programming Schneider Electric Modicon PAC's through its GUI. During this project, it was used to configure and program the Schneider Electric industrial equipment, which was included in the testbed.
- Modbus pool and Modbus slave [90] – these commercial tools, which are also available as trial versions, are Modbus simulators which have the ability to act as master or slave, respectively. They allow to test all the different features of Modbus and provide a GUI to allow the monitoring and visualization of the Modbus messages results. These tools where used to test, verify and debug Modbus communications, such as the ones which were included in the programming PLC's and in the attack script that was used in one of the use cases.
- Rapid SCADA – this freeware tool is a SCADA software. Although being a quite simplistic implementation of a SCADA system, it fulfils all the basic needs of such a system. The fact of being open source and offering a simple interface makes it a good choice for laboratory or small/medium size systems. This tool was used to create the HMI of the testbed.
- Mininet – this freeware and open source network emulator allows the creation of virtual networks with distinct topologies, in a simple "light" and scalable way. The created environments are OF compliant which use OVS's. It also allows the installation of handy network tools, like the ones mentioned above for performing tests, measurements and analysis of performance (some are already natively incorporated). This tool was used for quick prototyping networks, firstly at the beginning of this research in order to better understand the SDN mechanisms and later for testing different possibilities.

3.3. Network Equipment Testing and Validation

In the knowledge that ICSs have high requirements in terms of delays in communications, one of the first issues raised regarding the applicability of SDN in ICSs relates to performance in terms of latency.

Although the OFSs used in this work fell far short of the equipment used in the production environment – since two are software switches running on non-dedicated hardware and another is a hardware switch with very modest specifications, as already mentioned – it was important to have an idea of their performance in order to see if this would be a limiting factor for the testing related to the integration of the SDN in the ICSs. Being so, it was decided to carry out some tests and comparisons so as to gauge their level of performance. The methodology used to measure the switches/network performance is presented below.

A simple architecture composed by two hosts and a switch was used and replicated for each switch. In order to have a comparison term, a conventional Ethernet switch (3COM 4400) was also tested. During the testing no parallel usage of the network was being done.

To be able to retrieve the network metrics *iPerf* and *Ping* tools were used. Since it was used a closed and controlled testing environment which suffered no fluctuations during the testing, it was decided to only repeat each test ten times.

- Ping was used to collect delay and packet loss data. Being a simple and closed test environment with no parallel traffic or alternative paths, the round-trip time retrieved from the execution of the following command was interpreted as being twice the latency.

  ```
  $ ping 10.0.0.2 -c 120 -s [58 | 127 | 248] -i [0.1 | 0.25 | 0.5]
  ```

  Different sizes of packets were tested: 66, 135 and 256bytes. These values were chosen since most of the Modbus messages measured by analysing the traffic on the testbed had a 66bytes size and since the traffic measure on a real production environment was around 135bytes [91]. The values used in the shell command took into account the 8 bytes extra of the ICMP header data. Regarding the interval defined between *ping* commands, it was used 100, 250 and 500ms since these values are similar to the ones used in several ICSs. Each test was performed by sending 120 ICMP packets.

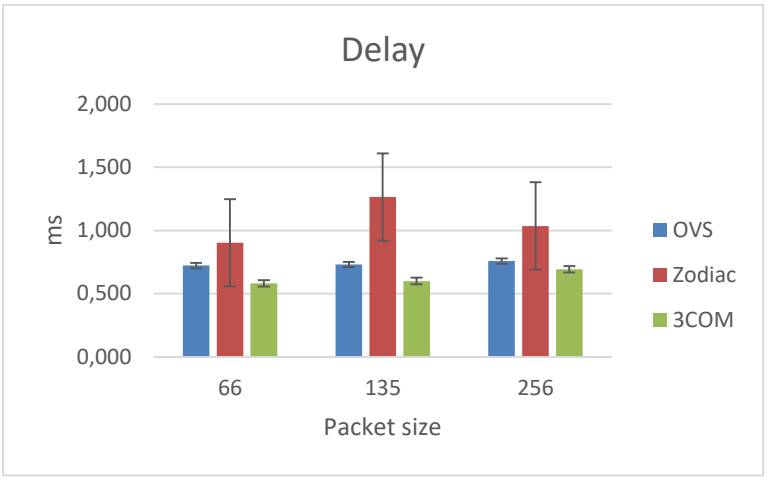


Figure 39 – Network delay

As it can be seen in the above graph (**Figure 39**; raw data presented in the Appendix A), the behaviour of the conventional switch (3COM) and the OVS was quite stable and similar throughout the different tests – hence, only one graph related to the testing is presented. Although the conventional switch always got slightly better results, this was already expected given the non-dedicated hardware in which OVS was installed. The Zodiac switch exhibited a somewhat unstable behaviour, which translated into a few peaks in the measured values which, in turn, had resulted in higher average of delay. In terms of packet loss, none was detected throughout this testing. However, one cannot conclude that in a production environment the same would apply since there would be more network elements and hops, more parallel and heterogeneous traffic and, possibly, packet collisions.

- iPerf was used to analyse the throughput capacity. The following commands were used on the server and on the client's side respectively.

$ iperf -s -w 512K	$ iperf -c 10.0.0.2 -t 120 -w 512K

TCP protocol was chosen since, by default, it uses the maximum available bandwidth provided by the network. A large window size was defined so it would not negatively influence the performance. The tests had the duration of 120 seconds.

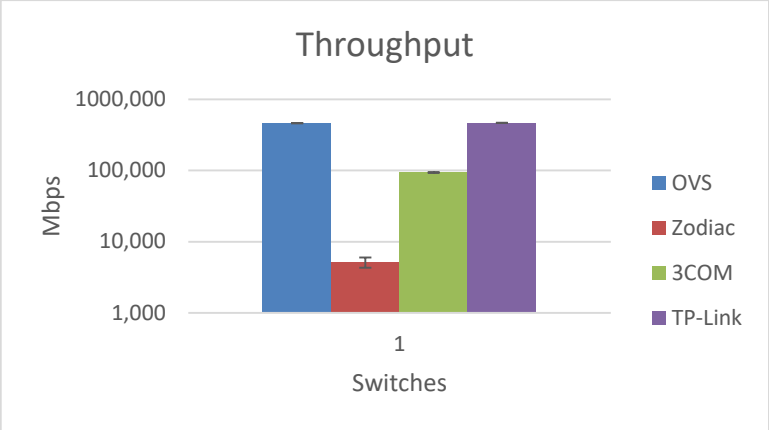


Figure 40 – Network throughput

In the first set of tests, only the 3 switches mentioned above were analysed. However, the discrepancy between the OVS and the conventional switch was quite high (note that the vertical axis of the graph uses a logarithmic scale). This would be due to the fact that the conventional switch had 100Mbps network interfaces, while the OVS had 1Gbps network interfaces. This allowed the positioning of the OVS above a conventional switch at 100Mbps, but not permitting to understand whether the performance obtained would be acceptable for a switch with gigabit interfaces. In order to clarify this doubt, it was possible to arrange a conventional gigabit switch (TP-Link TL-SG108E) to be submitted to similar tests. As it is possible to see in the graph above (**Figure 40**; raw data presented in the Appendix A), this switch obtained very close results to the OVS. Since the measured values were still far from 1Gbps, it was assumed that the network interfaces of the hosts used in the tests were to be responsible for limiting the measured throughput (probably due the usage of a shared PCI bus from the hypervisor machine). Nevertheless, one might conclude that the OVS performance was very positive. As for the Zodiac, the obtained results were quite disappointing. Despite being an experimental low cost switch, it was expected that it would have a better performance. However, it is thought that the poor results obtained may be due to a problem with the latest firmware and not to the hardware itself, since in quick tests carried out at the time of its acquisition it achieved much better results.

One should bear in mind that, in a production environment, there would be several other factors influencing the switches/network behaviour, such as multiple flows running in parallel, distinct protocols, variable packets size, which could all affect the switches performance. Also, regarding latency in OFSs, there are other aspects which can be of significant importance for its performance and which were not covered in detail by the performed tests, such as lookup cost, the recycle cost and the action cost. This happens because these aspects are strongly related to the used hardware

and, due to the type of OFSs used, it did not make sense to perform this tests since the results would reflect, even more, the hardware used and not the OF capabilities as was the objective of this research. Also, these are factors which may vary substantially from case to case dependently on the network programming.

Nevertheless, for laboratory usage, the OVS seems to be up to the task. As for the Zodiac switch, it was not possible to confirm the suspected firmware problems, so further contacts with the vendor will have to be done in order to clarify this situation.

3.4. Use Cases for SDN and ICS/SCADA Integration

Within the different possibilities considered while trying to find synergies between SDN and SCADA ICS, distinct scenarios underwent a more detailed investigation which is presented below. One scenario focused on the area of availability, the other scenarios, more complex, covered the areas of security, reliability, configuration and management.

3.4.1. Availability

Availability is one of the most important parameters in an ICS network. Failure in communications can lead to the loss of visibility of the process as well as making control decisions impossible. This can result in situations of breakdown of ongoing processes, which in turn may translate into costly environmental, financial or even deadly consequences.

The more demanding systems/standards, such as IEC 61850 for electrical substations automation systems, consider recovery times in the order of 0 to 100ms [92], although there is no strictly defined recovery time transversal to every ICS, due the existing heterogeneity on the usage of these systems – hence, the existence of soft real-time and hard real-time nomenclature.

3.4.1.1. Scenario 1 – Link Failover

In this scenario (**Figure 41**), a link failure is simulated in order to verify the SDN performance in terms of recovery time and also of how it is translated in terms of packet loss.

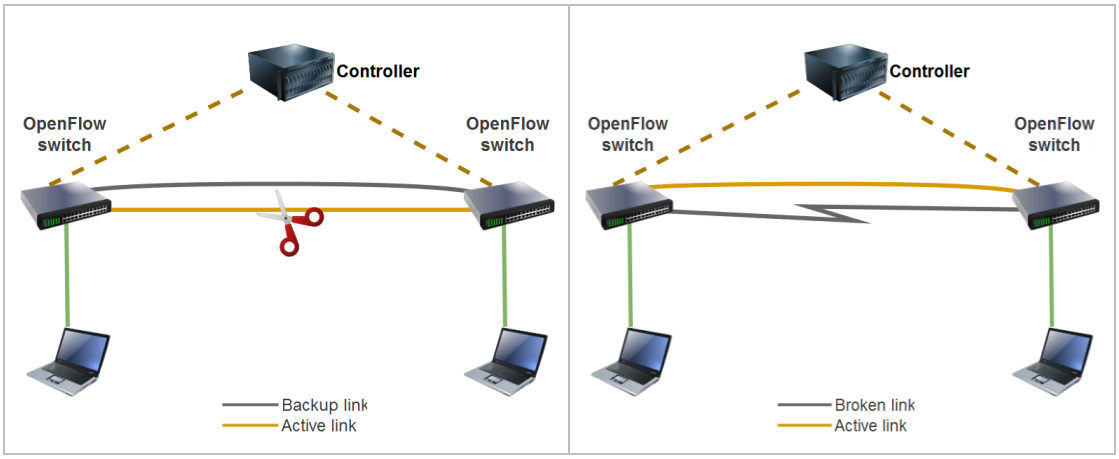


Figure 41 – Failover scenario

Both switches were preconfigured with a fast-failover group type so as to provide a redundant connection for when the main connection becomes unavailable, in which case the switches will automatically change the active link. When this strategy is used, the decision in case of failure is brought to the periphery of the network, thus avoiding all the communication overhead with the controller and not relying on higher layer protocols.

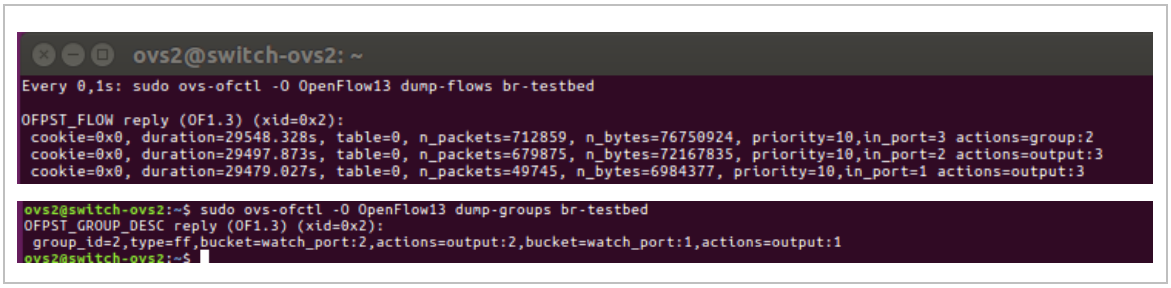


Figure 42 – Fast-failover flow and group entries

In the image above[19](**Figure 42**), it is possible to see the flow entries programmed in one of the switches. The first flow is responsible for outputting the packets coming from where the first device is connected to the group responsible for the fast failover (group id=2), which will use the first available bucket (bucket outputting for port number 2, if it is active, if not, it will use the next bucket, which is outputting for port number 1). The other flows are responsible for accepting the incoming traffic being sent by the other device and which will be entering through port number 2

[19] This image is a representation of the flow entries used just for this testing scenario. They were simplified in order to be the least granular for simplification and in order to fit the report.

or 1 and forwarded to port number 3 (where the first device is connected). On the other switch, similar flows were programmed according to the ports used on that switch.

The test methodology was to manually create two link failures and reconnections, which implied that the cable responsible for the main link was disconnected twice and, after each one, reconnected some time after. Every time the original main link was reconnected, it was again assumed as the active link. This resulted in four link swaps (**Figure 43**).

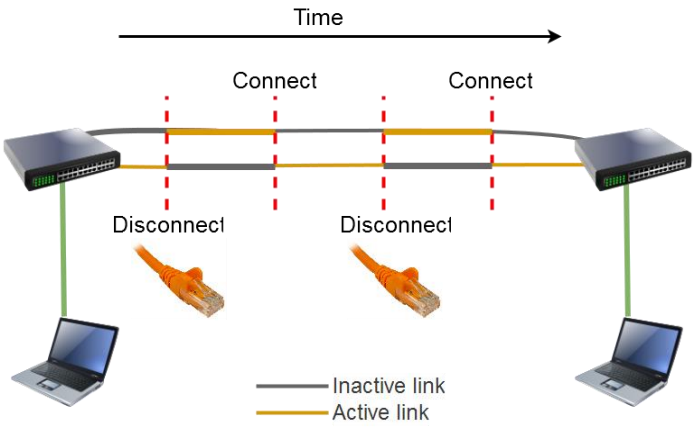


Figure 43 – Link failure scenario

In order to better control the experiment variables, to avoid possible hardware limitations and to better visualize the outcome data, the Ping tool was used in detriment of the ICS testbed. The following command was used to produce a burst of 1000 packets 135bytes size.

```
$ ping 10.0.0.2 -s 135 -c 1000 -i [0,1 | 0,01]
```

Two different rates were assumed for sending the packets: 0,1s and 0,01s. The first one came close to a more usual, yet demanding, value for Modbus TCP communications. The second was intended to verify the behaviour of the system in a situation of greater stress. After performing the tests, the reading of the results obtained for each situation **(Figure 44**; raw data presented in the Appendix A) indicated that they were very homogeneous, which proves the coherence of the functioning of the fast-failover system. Regarding the percentage of packet loss, it can be stated that the values obtained were quite encouraging. Achieving a loss of 13,5% when sending 1000 packets with a cadence of 0,01s and only 1,5% when sending with a cadence of 0,1s. This corresponds to the loss of 135 and 15 packets respectively, which translates into 1,35s and 1,5s of total recovery time. Since there were four jumps between links, we can conclude that the

recovery time for each testing was 0,337s and 0,375s respectively[20]. Having in consideration that the switch was running on non-dedicated hardware, with four link changes, there was a clear positive result, which leaves us wondering how much better results could have been achieved by using a high quality hardware switch.

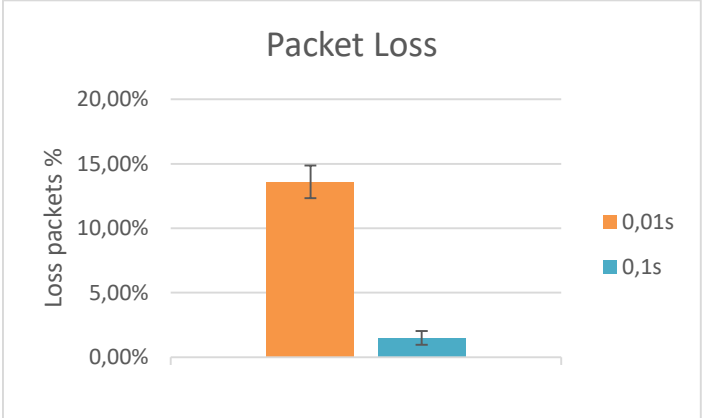


Figure 44 – Packet loss

Thus, it can be concluded that this fast-failover mechanism provided by OF is very efficient and meets the needs of ICS in terms of network recovery time. It would have been interesting to make a comparison between the recovery time of the network using the mechanisms described above and the rapid spanning tree protocol (RSTP), which was not possible to perform in the time conveyed for this research. Nonetheless, the article [93] refers recovery times in the order of 5 to 20ms for the RSTP, which seems to prove the conclusions drawn. On the other side, it must be taken into account that this recovery mechanism may not be the best option to all the network failure scenarios, since it acts in a localized way, without taking into account an overview of the network which would allow it to consider the state of the remaining links beyond the first hop.

This does not mean that it cannot be used in conjunction with centralized mechanisms that have this capability. For example, a centralized algorithm may predict which links are most likely to fail (either based on internal or external network factors) and prevent such failures by dynamically programming in advance this connection backup system.

[20] Assuming the same overhead for each link swap and considering that this was the only factor affecting the packet loss factor.

3.4.2. Security, Configuration and Management

In ICS, security is often neglected due to the difficulty of configuring and managing the network, further magnifying the exposure to malicious attacks already existing due to the use of insecure protocols.

Next, two scenarios are presented which demonstrate some of the capabilities of SDN in securing an ICS by detecting, analysing and reacting to anomalous network situations in a simple and effective way without compromising the ease of configuration and management.

The use of proactive high granular rules could, by itself, nullify or at least hinder some attacks. However, since it is being assumed that every network can be breached, some relaxation has been assumed in the programming of OpenFlow rules, once that the focus of these scenarios was in detection, analysis and reaction rather than prevention.

3.4.2.1. Scenario 2 – Automatic System Reaction

In this scenario (**Figure 45**), an attack[21] is made against the ICS by means of exploiting the vulnerabilities of the PLC and of the Modbus TCP protocol. An attacking machine does a MAC spoof of the physical address of PLC-1 and floods the PLC-2 with a Modbus TCP message which, using the *Write Single Register* function, writes in the same memory register that was being used to store the temperature value sent by PLC-1. The PLC-2 will execute its normal processing, considering the value sent by the attacking machine, because the message cadence sent by the attacking machine is much higher than the one sent by PLC-1 with the real value.

[21] The result of this attack is similar to a man-in-the-middle attack, in which an ARP cache poisoning is done by flooding ARP caches with ARP replies. In this more elaborate attack there would be a total control in the messages that are sent to the PLC-2. However, since the detection principle is the same, it was decided to make an attack, which clearly shows (and relies only on) some of the vulnerabilities specific to ICS's.

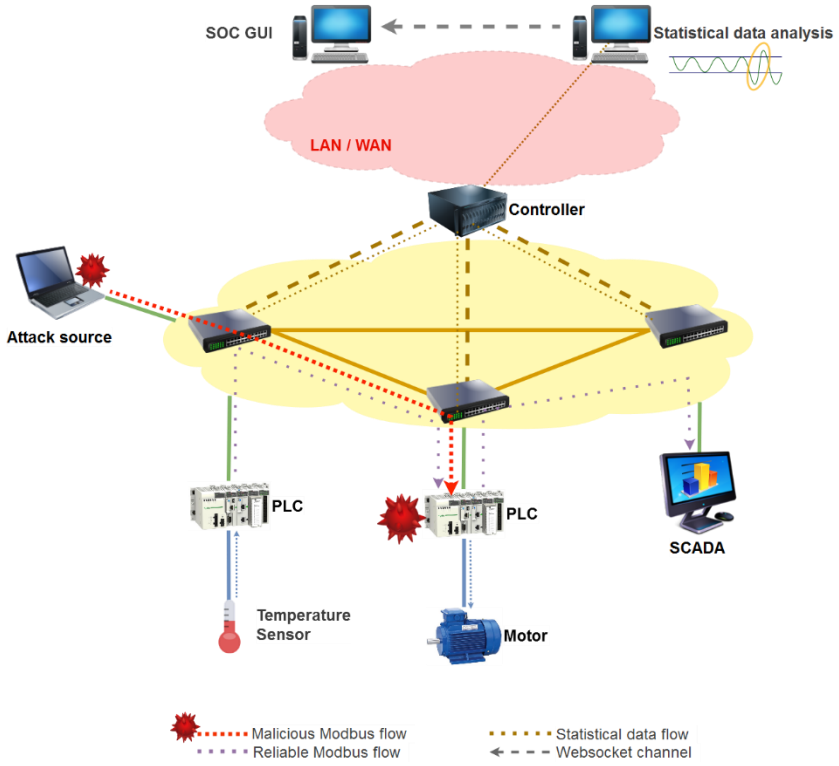


Figure 45 – Attack scenario with automatic reaction

When the management application is active, it detects an anomalous network pattern based on the analysis of the packet rate of the communication between the PLC-1 and PLC-2 in comparison to the previous analysis. A simple and lightweight statistical processing technique were applied to perform anomaly checks on communications traffic, using time series analysis based on a technique derived from the Shewhart [94] control charts. Shewhart control charts provide a low-overhead technique for teletraffic flow analysis - its main shortcoming relates to its low sensitivity regarding the detection of small sustained shifts in the mean. A Shewhart control chart defines the concept of Upper and Lower Control limits based on a statistic obtained from N observations $(y_{(l-1)N+1}, \ldots, y_l)$, in this specific case:

$$\bar{y_t} = \frac{1}{N} \sum_{t=(l-1)N+1}^{lN} y_t \; where \; l = 1,2, \ldots$$

Based on flow information, the SDN app is able to calculate totals for several features, such as bandwidth and transferred bytes as well as the inter-packet arrival time. For each captured interval (5s), the values for these properties are averaged, accordingly with the formula. However, these averages do not constitute an unbiased estimator of the mean, due to the fact the samples are not

independent and also because the measurements are not guaranteed to follow a normal distribution - due to its nature, network captures do not correspond to a stationary random process with uncorrelated values. Working hour schedules may generate non-stationarities, with serial correlation being caused by nature of protocol state machines, whom not change states in an arbitrary fashion - especially in SCADA, communications tend to systematically change accordingly with patterns that can be easily uncovered by a network trace autocorrelation graph.

Traditional Shewhart charts are geared towards Gaussian stationary processes with stable means and standard deviations, therefore, with no autocorrelation, as control limits are based on the standard deviation of the sampled variable. In the presence of a stationary Gaussian process, this would mean that the means and standard deviation would be stable, implying that observations within the $-2*\sigma$ to $2*\sigma$ would correspond to a 95% confidence interval. Since they rely on a fixed interval, calculated from σ, classic Shewhart charts are not suitable for non-stationary or autocorrelated processes, as they would generate too many false alarms.

While time-series analysis allows for removing pattern phenomena and serial correlation, there is the difficulty of fitting auto-regressive models (and the implied computational complexity). As an alternative, the analysis of communications traffic relies on exponential smoothing, using prediction errors (also called residuals) in Shewhart analysis - an approach some authors refer by Exponential Weighted Mean Average (EWMA) charts with residuals [95]. This allows for implementing adaptive control intervals, accordingly with the evolution of the analysed data (the optimal value for α was estimated by trial-and-error, for the testbed scenario).

$$\hat{x}_{t+1} = \alpha x_t + (1-\alpha)\hat{x}_t \text{ , with } \hat{x}_{t+1} \text{ being an estimation of } x_{t+1}$$

While σ could be calculated from the past history of sampled residuals (that is, the time series of prediction errors between readings and forecasted values), [96] refers that this estimation would be very sensitive to outliers, suggesting instead the usage of a moving estimator using the Exponentially Weighted Mean Square Error (EWMS), as a variance estimator (the smoothing constant ρ was established at 0.01, after several tests, while ε_t is the prediction error of the residuals, that is $\varepsilon_t = x_t - \hat{x}_t$). The UCL and LCL calculated from $\hat{\sigma}_t^2$, as suggested by [96].

$$\hat{\sigma}_t^2 = \rho(\varepsilon_t + \mu) + (1-\rho)\hat{\sigma}_{t-1}^2, \mu \approx 0 \text{ under normal conditions}$$

This solution allows for the implementation of an online anomaly detection system that relies on time-series analysis of residual prediction errors to provide semi-supervised anomaly detection. Due to the nature of the SCADA process and field network environments, there is a considerable degree of stability and homogeneity (especially in comparison with ICT networks), regarding the

mix of protocols, hosts and traffic flows involved that makes this approach particularly appropriate, not only for cyber-security monitoring but also for safety monitoring and diagnostics.

After an anomaly is detected and identified as an attack (**Figure 46**), the application will react by sending a REST command to the controller to block the responsible flow and issuing a warning to its own console in order to alert the responsible personal. The controller then sends an OF message to the entrance switch in order for it to drop the respective packets, thus eliminating the exposure to the attack.

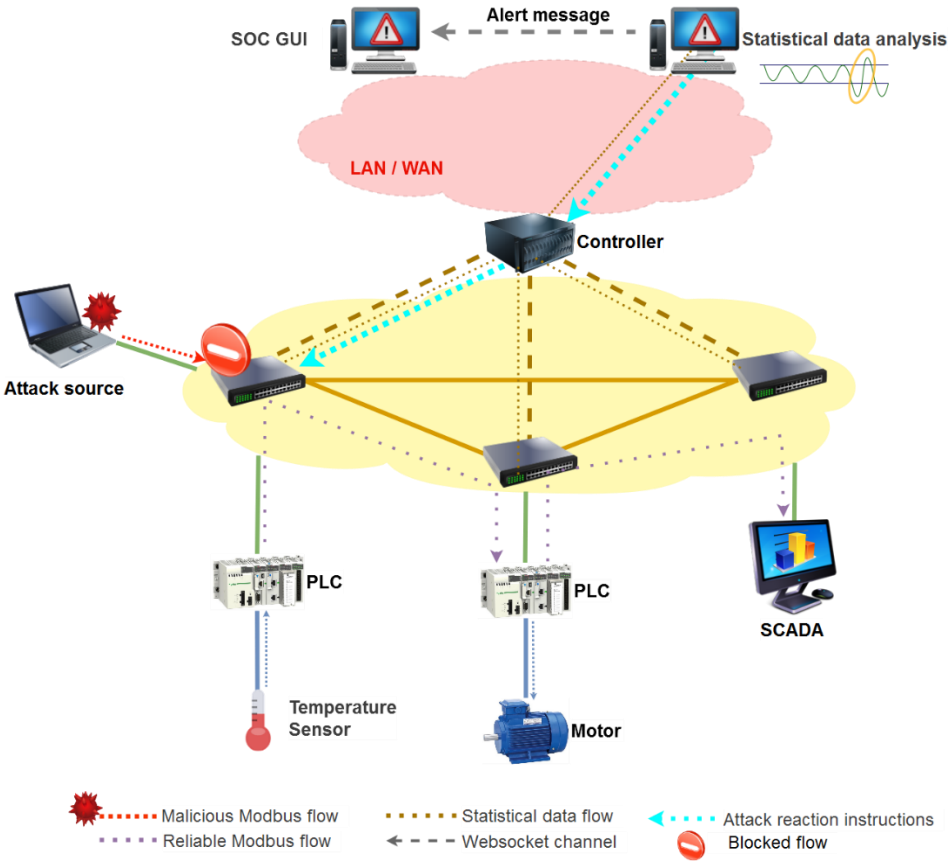


Figure 46 – Automatic attack reaction

This method proved to be quite effective in detecting, identifying and responding to an attack. No human intervention was required in order to accomplish this objective since all the needed actions and configurations were taken by the management application. These results help to prove the advantages in terms of security, configuration and management of SDN.

Although the decision making comes from the layers above the data plane, the OF specifications contemplate *meters* that allow decisions dependent on the rate of packets to be made locally on the switch and before any type of forwarding action. By doing it, in addition to avoiding

communication overhead, it would be possible to immediately block traffic in the periphery of the network. The management application could program and update *meters*, and the respective *meter bands*, in advance according to the average packet rate, thus improving the reaction time of the system and bypassing possible attack vectors that may first try to isolate the switch from the controller and only then perform the attack to the PLC. Nevertheless, this alternative has not been tested since this functionality is still not implemented neither in the OVS [97] nor the Zodiac switch.

However, in ICS, automatic blocking of communication may not be the best solution due to the problems previously identified on the first scenario. In addition, this mode of action is not always well accepted by those responsible for ICS who, in many cases, prefer decision-making to be taken by a responsible technician. An alternative method is then presented in the next scenario.

3.4.2.2. Scenario 3 – Manually Operated Reaction (human-in-the-loop)

In this scenario (**Figure 47**), a similar attack vector is used but the value sent by the attacker along with the *Write Single Register* function is out of a range (0-200º) that was pre-defined having in consideration the logic of this particular industrial process.

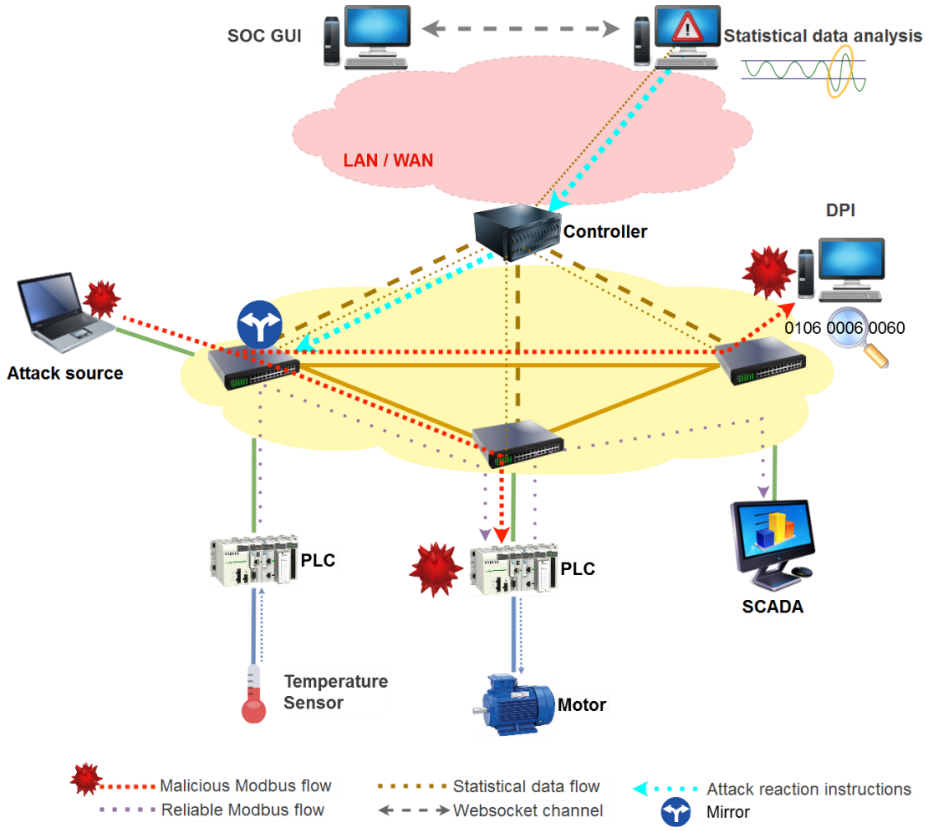


Figure 47 – Attack scenario with manually operated reaction

The management application detects an anomalous network pattern based on the significantly increasing bytes' average rate of the flow responsible for the communication between the PLC-1 and PLC-2 in comparison with the previous analysis. It then sends a REST command to the controller in order to create a mirror of that flow, which is then sent to a deep packet inspection (DPI) system (Snort [98]). The controller then sends an OF message to the switch so as to program the intended mirror.

After the mirror is created, the DPI system starts to receive a copy of the anomalous flow.

The analysis made by the DPI looks inside the Modbus TCP frame so as to check the bytes corresponding to the Modbus function code and data. Below (**Figure 48**) is possible to see the rule used by the DPI.

```
Alert tcp any any -> 10.0.0.200 502 (msg: "Alert IDS…"; content:"|06|"; offset:7; depth:1;
byte_test:2,>,200,10; sid:10000003;)
```

Figure 48 – Snort rule for Modbus analyse

If it detects a write function (in this particular case the *Write Single Registry* function) and a data value out of the pre-defined range, it triggers an alert message that will be sent to the application manager through a network socket. In **Figure 49** is possible to see a Modbus TCP packet and an explanation regarding the above mentioned bytes.

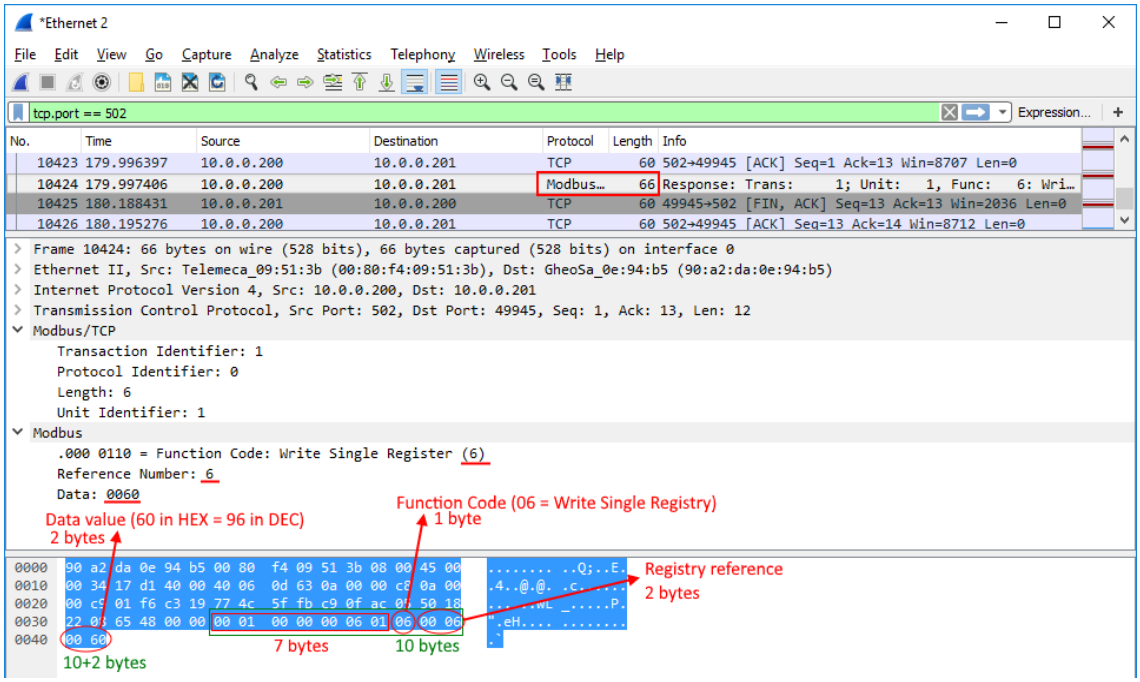


Figure 49 – Modbus TCP packet inspection

The application manager will then issue an alert in its GUI and wait for the responsible technician to decide what behaviour should be adopted. For example, he decides whether to:

- block the communication (as represented in **Figure 50**), which implies that a REST message is sent to the controller in order to block the responsible flow. The controller then sends an OF message to the entrance switch in order for it to drop the respective packets, thus eliminating the exposure to the attack.
- forward the communication to a honey pot for further analysis. REST messages are sent to the controller so as to divert the intended flow and rewrite the packet headers (IP and MAC addresses) making the diverting transparent to the attacker. The controller then sends the correspondent OF messages to the entrance switch in order for it to apply the intended changes.

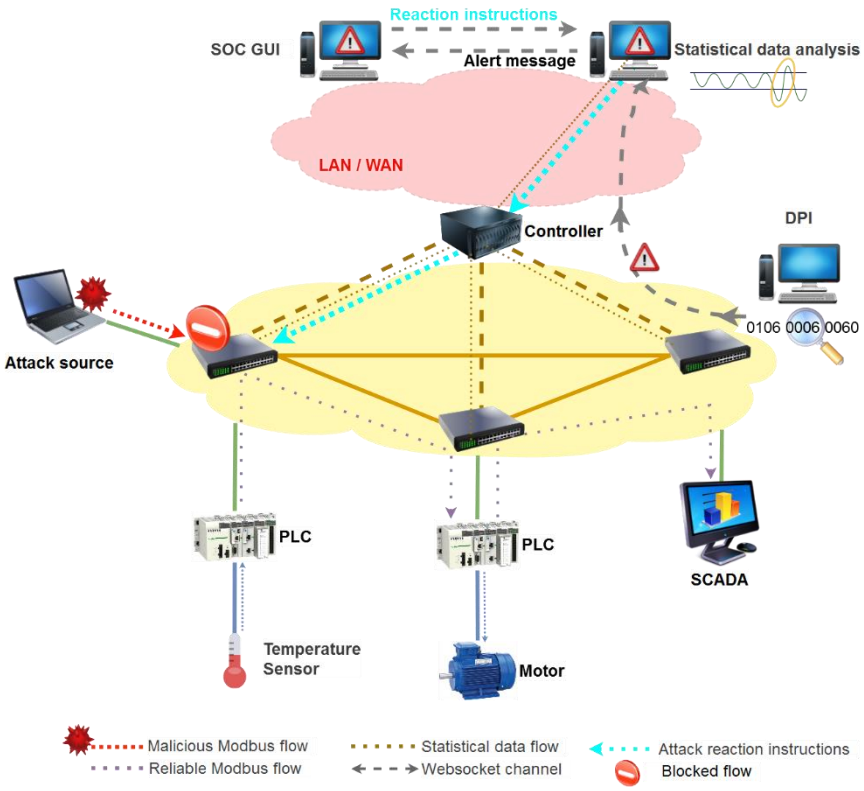


Figure 50 – Attack reaction

As in the second scenario, it was also possible to verify the effectiveness, easiness and versatility that SDN can bring to ICS networks in terms of security, configuration and management. In this third scenario, human intervention is highlighted in the decision-making process, without this implying any increase in complexity in terms of network configuration nor compromising the security of the network.

Chapter 4 – Conclusions and Future Developments

In the course of this investigation, and despite some improvements as of mid-2016, some challenges and constraints, mainly related to its connection to recent and still developing technology, had to be overcome as to reach this research purposes. The several challenges faced were gradually overbore allowing the research to progress positively. Along with all the work done, it was also possible to produce and publish two articles in the scope of this research:

- "Leveraging virtualization technologies to improve SCADA ICS security", published in a special edition of *Journal of Information Warfare* (Volume 15, Issue 3). (Appendix B)

- "Security implications of SCADA ICS virtualization: survey and future trends", published in the 15th European Conference on Cyber Warfare and Security and considered one of the best papers of the conference. (Appendix C)

A third article is already being prepared, to be submitted to a class A conference (NOMS or LCN are being considered).

4.1. Conclusions

From all the research done one can conclude that the proper SDN integration in ICS presents numerous advantages at different levels.

Firstly, it has been shown that it is possible to perform the integration of both technologies in a simple manner, without the need of any type of intervention or change in equipment from the ICS, a factor which frequently delays the adoption of new technologies.

As for configuration and management, some potential of SDN has been found, by being able, in real-time, to display network status as well as to still being able to perform alterations either *via* pre-programming or *via* human interaction with a simple click of a button.

As for availability in cases of breach in communication due to connection failure, it has been proven that the mechanism inherent to OpenFlow allows a quick and effective response, minimizing or even annulling the negative impact on ICS.

In what security is concerned, it is currently highly consensual that communication analysis can prove itself to be quite helpful in the identification of anomalous situations, which are often not detected by other more localized systems. ICS SCADA networks are highly vulnerable due to their equipment and specific communication requirements. In this area, SDN present itself as an asset, since it allows for a quite granular control and analysis of all communication. As proven, the merge of SDN to ICS SCADA permits us to make the entire network safer, not only in prevention, but also in detection, analysis and response to anomalous (and often malicious) situations.

4.2. Future Work

Within the scope of configuration and management, it would be interesting to explore a scenario in which a system created for this purpose received as input configuration files from the industrial control, containing data such as identification of network devices and their programming (e.g. ladder logic with some extra information). According to the analysis of the mentioned files, this system would then be able to identify data (such as the elements intervening in communication, the frequency of this communication, …) and automatically configure the entire network. Such system would greatly facilitate the ground up creation of an ICS as well as the management and integration of new systems or devices in production environment.

As for availability, it would be interesting to explore more complex mechanisms, which would take into consideration the entire network, as to being able to intervene in the network for the optimization of links as well as to respond to broader and more complex connection failure.

In what security is concerned, similarly to what is presented in this project, it would be interesting to develop a centralized security system (based on a SaaS model), that could analyse, monitor and intervene simultaneously in different systems. This would provide two clear benefits: broader knowledge in terms of security, due the centralization of data resulting of the analysis and monitoring of several systems; enhanced expertise in security and its innovation, due to this being a usual gap in ICS environments.

Generally speaking, one of the great advantages of SDN is its easy merge with other technologies.

Other beneficial ideas, apart from those mentioned above, could arise from the merge of other technologies such as machine learning for the analysis of several metrics offered by SDN as to detect patterns and behaviours that might be explored (e.g. relation between behaviour and ICS system status with the several network flow and parameters).

It would also be interesting to merge network function virtualization (NFV), which allows the decoupling of the software implementation of the network functions form the underlying hardware by means of leveraging virtualization techniques, with SDN flexibility, so as to achieve more adaptable networks on demand. A good application for an ICS system that could make use of this interconnection could be a detection and analysis system where virtual security functions, such as DPI or malware filters, could be dynamically deployed on any part of the network when and where needed (e.g. when there is a suspicion of some kind of threat in the network or when the already deployed network functions could need extra processing capabilities so as to be able to process data in real-time – this could be achieved by parallel processing). NFV would be responsible for the deployment, SDN would be responsible for adapting the network to the new deployed elements.

It would also be interesting to be able to perform tests in production environment, or similar, since this would certainly lead to benefits in the research and development of new solutions. During this project an effort was made as to reach this goal, considering the investment constraints presented.

Summing up, I believe that the work produced cleared some of the way for the exploitations of synergies between SND and ICS and, not only did it achieved the intended purposes, as it also resulted in data which opens possibilities for future investigation.

References

[1] Y. J. Reddy and B. R. Mehta, Industrial process automation systems - Design and implementation, Elsevier Inc, 2015.

[2] "Bloodhound on my Trail: Building the Ferranti Argus Process Control Computer," *The International Journal for the History of Engineering & Technology Vol 82, No 1,* January 2012.

[3] Intel®, "The Story of the Intel® 4004," [Online]. Available: http://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html. [Accessed 15 01 2017].

[4] V. Cerf, Y. Dalal and C. Sunshine, "RFC675: SPECIFICATION OF INTERNET TRANSMISSION CONTROL PROGRAM," December 1974. [Online]. Available: https://tools.ietf.org/html/rfc675. [Accessed 15 01 2017].

[5] R. Metcalfe and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM, Vol. 19, No. 5,* pp. 395-404, July 1976.

[6] C. D. Marsan, "Ethernet and IP storm - factory nets," *Network World,* pp. 79-80, 10 July 2010.

[7] J. Slay and M. Miller, "Lessons learned from the Maroochy water breach," in *Goetz E., Shenoi S. (eds) Critical Infrastructure Protection.,* Springer, Boston, MA, International Federation for Information Processing, 2007.

[8] "Malicious Control System Cyber Security Attack Case Study – Maroochy Water Services, Australia," [Online]. Available: http://csrc.nist.gov/groups/SMA/fisma/ics/documents/Maroochy-Water-Services-Case-Study_briefing.pdf. [Accessed 15 01 2017].

[9] D. Kushner, "IEEE Spectrum - The real story of stuxnet," February 2013. [Online]. Available: http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet. [Accessed 15 01 2017].

[10] K. Zetter, Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon, Crown Publishers, 2014.

[11] B. Miller and D. Rowe, "A Survey of SCADA and Critical Infrastructure Incidents," *SIGITE'12,* 10 2012.

[12] "Repository of Industrial Security Incidents," [Online]. Available: http://www.risidata.com/. [Accessed 15 01 2017].

[13] "Dell security annual threat report," 2015. [Online]. Available: https://software.dell.com/docs/2015-dell-security-annual-threat-report-white-paper-15657.pdf. [Accessed 15 01 2017].

[14] McAfee Labs, "Report 2016 Threats Predictions," 2015.

[15] Urs Hoelzle, Google, "Open Networking Summit," April 2012. [Online]. Available: https://www.youtube.com/watch?v=VLHJUfgxEO4. [Accessed 15 01 2017].

[16] Urs Hoelzle, Google, “Open Networking Summit,” April 2012. [Online]. Available: http://opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf. [Accessed 15 01 2017].

[17] K. Greene, “TR10: Software-defined networking,” *MIT Technology Review,* March/April 2009.

[18] N. Feamster, J. Rexford and E. Zegura, “The Road to SDN - An intellectual history of programmable networks,” 2013.

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” 2008.

[20] “OpenFlow Archive,” 2011. [Online]. Available: http://archive.openflow.org. [Accessed 15 01 2017].

[21] O. N. Foundation, “OpenFlow switch specification,” 31 12 2009. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf. [Accessed 15 01 2017].

[22] E. Research, “OpenFlow-MPLS,” [Online]. Available: http://archive.openflow.org/wk/index.php/OpenFlowMPLS. [Accessed 15 01 2017].

[23] Open Networking Foundation, “OpenFlow Switch Specification v1.1,” 28 2 2011. [Online]. Available: http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf. [Accessed 15 01 2017].

[24] “Open Networking Foundation,” [Online]. Available: https://www.opennetworking.org. [Accessed 15 01 2017].

[25] Open Networking Foundation, “OpenFlow Switch Specification v1.2,” 5 12 2011. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf. [Accessed 15 01 2017].

[26] Open Networking Foundation, “OpenFlow Switch Specification v1.3,” 25 6 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf. [Accessed 15 01 2017].

[27] Open Networking Foundation, “OpenFlow Switch Specification v1.4,” 14 10 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf. [Accessed 15 01 2017].

[28] Open Networking Foundation, “OpenFlow Switch Specification v1.5,” 19 12 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf. [Accessed 15 01 2017].

[29] K. Smiler, “Flow Table and Flow Entry Modification Messages,” in *OpenFlow Cookbook*, Birmingham, Packt Publishing, 2015, pp. 81-87.

[30] “Ethane: A Security Management Architecture,” [Online]. Available: http://yuba.stanford.edu/ethane/index.html. [Accessed 15 01 2017].

[31] “Ternary Content-Addressable Memory,” [Online]. Available: https://www.pagiamtzis.com/cam/camintro/. [Accessed 15 01 2017].

[32] “Scott Shenker,” [Online]. Available: http://www.icsi.berkeley.edu/icsi/about/board/shenker. [Accessed 15 01 2017].

[33] S. Shenker, “Colloquium on Computer Systems Seminar Series,” [Online]. Available: https://www.youtube.com/watch?v=WabdXYzCAOU. [Accessed 15 01 2017].

[34] B. Pfaff, B. Davie, “The Open vSwitch Database Management, Internet Engineering Task Force,” [Online]. Available: http://www.ietf.org/rfc/rfc7047.txt. [Accessed 15 01 2017].

[35] Internet Engineering Task Force, “MPLS Transport Profile (MPLS-TP) Linear Protection,” [Online]. Available: https://tools.ietf.org/html/rfc6378. [Accessed 15 01 2017].

[36] Network Working Group, “Border Gateway Protocol,” [Online]. Available: https://tools.ietf.org/html/rfc4271. [Accessed 15 01 2017].

[37] CISCO, “OpFlex: An Open Policy Protocol White Paper,” [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html. [Accessed 15 01 2017].

[38] Internet Engineering Task Force, “Network Configuration Protocol,” [Online]. Available: https://tools.ietf.org/html/rfc6241. [Accessed 15 01 2017].

[39] Internet Engineering Task Force, “Locator/ID Separation Protocol (LISP),” [Online]. Available: https://tools.ietf.org/html/rfc7834. [Accessed 15 01 2017].

[40] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, Internet Engineering Task Force, “Forwarding and Control Element Separation (ForCES),” [Online]. Available: https://tools.ietf.org/html/rfc5810. [Accessed 15 01 2017].

[41] “Open vSwitch,” [Online]. Available: http://openvswitch.org. [Accessed 15 01 2017].

[42] Open Networking, “Open Networking - Products Listing,” [Online]. Available: https://www.opennetworking.org/products-listing. [Accessed 15 01 2017].

[43] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown and S. Shenker, “NOX: Towards an Operating System for Networks,” 2008.

[44] “NOX,” [Online]. Available: http://www.noxrepo.org/. [Accessed 15 01 2017].

[45] “POX,” [Online]. Available: https://www.noxrepo.org/pox. [Accessed 15 01 2017].

[46] “Beacon,” [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home. [Accessed 15 01 2017].

[47] “Floodlight,” [Online]. Available: http://www.projectfloodlight.org/.

[48] “ONOS,” [Online]. Available: http://onosproject.org/. [Accessed 15 01 2017].

[49] “Open MuL,” [Online]. Available: http://www.openmul.org/. [Accessed 15 01 2017].

[50] “OpenDaylight,” [Online]. Available: https://www.opendaylight.org/. [Accessed 15 01 2017].

[51] “Ryu,” [Online]. Available: http://osrg.github.io/ryu/. [Accessed 15 01 2017].

[52] “Trema,” [Online]. Available: https://trema.github.io/trema/. [Accessed 15 01 2017].

[53] D. Kreutz, F. Ramos, P. Verissimo, C. Rothenberg, S. Azodolmolky and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” 2014.

[54] “ISA-95,” [Online]. Available: https://www.isa.org/isa95/. [Accessed 15 01 2017].

[55] “Modbus,” [Online]. Available: http://www.modbus.org/. [Accessed 15 01 2017].

[56] “Ethernet/IP,” [Online]. Available: https://www.odva.org/Technology-Standards/EtherNet-IP/Overview. [Accessed 15 01 2017].

[57] “Fieldbus,” [Online]. Available: https://en.wikipedia.org/wiki/Fieldbus. [Accessed 15 01 2017].

[58] “Automation - Profibus and Modbus: a comparison,” [Online]. Available: http://www.automation.com/automation-news/article/profibus-and-modbus-a-comparison. [Accessed 15 01 2017].

[59] “Fieldbus Foundation,” [Online]. Available: http://www.fieldbus.org. [Accessed 15 01 2017].

[60] “Wikipedia - List of automation protocols,” [Online]. Available: https://en.wikipedia.org/wiki/List_of_automation_protocols. [Accessed 15 01 2017].

[61] “Profibus,” [Online]. Available: http://www.profibus.com. [Accessed 15 01 2017].

[62] “International Society of Automation - IEC 61131-3,” [Online]. Available: https://www.isa.org/standards-publications/isa-publications/intech-magazine/2012/october/system-integration-iec-61131-3-industrial-control-programming-standard-advancements/. [Accessed 15 01 2017].

[63] C. H. Gresser, “Hacking SCADA/SAS Systems,” in *Seminar at Petroleum Safety Authority Norway*, 2006.

[64] European Union Agency for Network and Information Security (ENISA), “Window of exposure... a real problem for SCADA systems?,” 2013.

[65] “ISA99, Industrial Automation and Control Systems Security,” [Online]. Available: https://www.isa.org/isa99/. [Accessed 15 01 2017].

[66] “7th Framework Programme for Research and Technological Development,” [Online]. Available: https://ec.europa.eu/research/fp7/understanding/fp7inbrief/home_en.html. [Accessed 15 01 2017].

[67] “MICIE,” [Online]. Available: http://www.micie.eu/. [Accessed 15 01 2017].

[68] “CockpitCI,” [Online]. Available: http://www.cockpitci.eu/. [Accessed 15 01 2017].

[69] “ATENA,” [Online]. Available: https://www.atena-h2020.eu/. [Accessed 15 01 2017].

[70] “Horizon 2020,” [Online]. Available: https://ec.europa.eu/programmes/horizon2020/. [Accessed 15 01 2017].

[71] X. Dong, H. Lin, R. Tan, R. K. Iyer and Z. Kalbarczyk, “Software-Defined Networking for Smart Grid Resilience: Opportunities and Challenges,” in *Cyber Physical System Security (CPSS'15)*, Singapore, 2015.

[72] N. Irfan and A. Mahmud, “A novel secure SDN/LTE-based Architecture for Smart Grid,” *Proceedings of the 2015 IEEE International Conference on Computer and Information Technology,* pp. 762-69, 2015.

[73] W. Machii, I. Kato, M. Koike, M. Matta, T. Aoyama, H. Naruoka, K. I and Hashimoto, “Dynamic zoning based on situational activities for ICS security,” *Proceedings of the,* pp. 1-5, 2015.

[74] Chavez, A, Hamlet, J, Lee, E, Martin, M & Stout, W, Sandia National Laboratories, “Network randomization and dynamic defense for critical infrastructure systems,” Albuquerque, New Mexico and Livermore, California, U.S.A., 2015.

[75] E. Silva, L. Knob, J. Wickboldt, L. Gaspary, L. Granville and A. Schaeffer-Filho, “Capitalizing on SDN-based SCADA systems: an anti-eavesdropping case-study,” in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, Ottawa, ON, 2015.

[76] B. Genge, P. Haller, A. Beres, H. Sándor and I. Kiss, “Using software-defined networking,” *Securing Cyber-Physical Systems,* pp. 305-29, 2016.

[77] P. Simões, T. Cruz, J. Proença and E. Monteiro, “On the use of honeypots for detecting cyber attacks on industrial control networks,” *Proceedings of the 12th European Conference on Information Warfare and Security,* pp. 263-70, 2013.

[78] Y. Song, S. Shin and Choi, “Network Iron Curtain: hide enterprise networks with,” *Information security applications, lecture notes in computer science,* pp. 218-30.

[79] N. Adrichem, B. J. Asten and F. A. Kuipers, “Fast Recovery in Software-Defined Networks,” in *EWSDN '14 Proceedings of the 2014 Third European Workshop on Software*, Washington, 2014.

[80] N. Dorsch, F. Kurtz, H. Georg, C. Hagerling and C. Wietfeld, “Software-Defined Networking for Smart Grid Communications: Applications, Challenges and Advantages,” in *IEEE International Conference on Smart Grid Communications*, 2014.

[81] “Mininet,” [Online]. Available: http://mininet.org/. [Accessed 15 01 2017].

[82] “KICKSTARTER - Zodiac FX: The world's smallest OpenFlow SDN switch,” [Online]. Available: https://www.kickstarter.com/projects/northboundnetworks/zodiac-fx-the-worlds-smallest-openflow-sdn-switch. [Accessed 15 01 2017].

[83] “OpenDaylight membership,” [Online]. Available: https://www.opendaylight.org/membership/. [Accessed 15 01 2017].

[84] SDX Central, “The Future of Network Virtualization and SDN Controllers,” 9 2016. [Online]. Available: https://www.sdxcentral.com/reports/network-virtualization-sdn-controllers-2016/chapter-5-sdn-controller-market-landscape/. [Accessed 15 01 2017].

[85] “Rapid SCADA,” [Online]. Available: http://rapidscada.org/. [Accessed 15 01 2017].

[86] “iPerf,” [Online]. Available: https://iperf.fr/. [Accessed 15 01 2017].

[87] “Wireshark,” [Online]. Available: https://www.wireshark.org/. [Accessed 15 01 2017].

[88] “Apache - Karaf,” [Online]. Available: http://karaf.apache.org/. [Accessed 15 01 2017].

[89] “Schneider Electric,” [Online]. Available: http://www.schneider-electric.com. [Accessed 15 01 2017].

[90] “ModBus Tools,” [Online]. Available: http://www.modbustools.com. [Accessed 18 01 2017].

[91] M. Mantere, M. Sailio and S. Noponen, “Network Traffic Features for Anomaly Detection in Specific Industrial Control System Network,” *Future Internet*, pp. 460-473, 25 09 2013.

[92] “ABB Review Special Report: IEC 61850,” 2010.

[93] “Industrial Ethernet Book Issue 96 / 8,” [Online]. Available: http://www.iebmedia.com/?id=11813&parentid=74&themeid=275&showdetail=true&bb=true. [Accessed 15 01 2017].

[94] W. A. Shewhart, Economic Control of Quality Manufactured Product . D.Van Nostrand Reinhold, Princeton, NJ,, New York: D. Van Nostrand Company, Inc., 1931.

[95] C. Chatfield, The Analysis of Time Series: An Introduction, 6th edition, Chapman and Hall/CRC , 2003.

[96] G. Münz and G. Carle, “Application of Forecasting Techniques and Control Charts for Traffic Anomaly Detection,” in *Proceedings of the 19th ITC Specialist Seminar on Network Usage and Traffic*, Berlin, Germany, 2008.

[97] “Open vSwitch v2.5,” [Online]. Available: http://openvswitch.org/support/dist-docs-2.5/TODO.md.txt. [Accessed 15 01 2017].

[98] “Snort,” [Online]. Available: https://www.snort.org/. [Accessed 15 01 2017].

[99] M. Abrams and J. Weiss, “Malicious Control System Cyber Security Attack Case Study - Maroochy Water Services, Australia,” 2008.

[100] M. Abrams and J. Weiss, “Bellingham, Washington, Control system cyber security case study,” 2007.

[101] “OpenFlow: Enabling Innovation in Campus Networks,” 14 March 2008.

[102] M. Dhawan, R. Poddar, K. Mahajan and V. Mann, “SPHINX: Detecting Security Attacks in Software-Defined Networks,” in *The Network and Distributed System Security Symposium*, 2015.

[103] J. Stirland, K. Jones, H. Janicke and T. Wu, “Developing Cyber Forensics for SCADA Industrial Control Systems,” in *Proceedings of the International Conference on Information Security and Cyber Forensics*, Kuala Terengganu, Malaysia, 2014.

[104] “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,” in *Hotnets*, Monterey, CA, USA, 2010.

[105] P. Pupatwibul, A. Banjar, A. A. Sabbagh and R. Braun, “A Comparative Review: Accurate OpenFlow Simulation Tools for Prototyping,” *JOURNAL OF NETWORKS, VOL. 10, NO. 5, MAY 2015*, vol. 10, 5 2015.

[106] V. Antonenko and R. Smelyanskiy, “Global Network Modelling Based on Mininet Approach,” in *HotSDN*, Hong Kong, China., 2013.

[107] B. Heller, “REPRODUCIBLE NETWORK RESEARCH WITH HIGH-FIDELITY EMULATION,” 2013.

[108] C. Doom, An introduction to business information management, Brussels: ASP, 2009.

[109] D. Baley and E. Wright, Practical SCADA for Industry, Newnes, 2003.

[110] “International Electrotechnical Commission - Core Standards,” [Online]. Available: http://www.iec.ch/smartgrid/standards/. [Accessed 15 01 2017].

[111] Flowgrammable, “OpenFlow,” [Online]. Available: http://flowgrammable.org/sdn/openflow/. [Accessed 15 01 2017].

[112] Hewlett-Packard, “Data center security redefined,” 2014. [Online]. Available: http://h20195.www2.hp.com/v2/GetPDF.aspx%2F4AA5-1629ENW.pdf. [Accessed 12 06 2016].

[113] “Northbound Networks,” [Online]. Available: https://northboundnetworks.com/. [Accessed 15 01 2017].

[114] “PLC Manual,” [Online]. Available: www.plcmanual.com. [Accessed 15 01 2017].

[115] “Project Floodlight,” [Online]. Available: https://floodlight.atlassian.net/wiki/. [Accessed 15 01 2017].

[116] “Open Networking Foundation - products listing,” [Online]. Available: https://www.opennetworking.org/products-listing. [Accessed 15 01 2017].

[117] R. Fontes, “Visual Network Description,” [Online]. Available: http://www.ramonfontes.com/visual-network-description/. [Accessed 15 01 2017].

[118] J. L. Chen, W. Ma, H. Kuo and W. C. Hung, “EnterpriseVisor: A Software - Defined Enterprise Network Resource Management Engine,” in *Proceedings of the 2014 IEEE/SICE International Symposium on System Integration*, Tokyo, Japan, 2013.

Appendix A

Network Equipment Testing: Wait interval between packet sending: 0,100s									
Switch	OVS			Zodiac			3COM		
Packet size (bytes)	**66**	**135**	**256**	**66**	**135**	**256**	**66**	**135**	**256**
Collected data	0,779	0,707	0,719	0,710	0,742	0,841	0,638	0,609	0,646
	0,710	0,737	0.709	0,710	1,844	1,031	0,574	0,586	0,620
	0,719	0,740	0,728	1,041	1,706	0,855	0,559	0,598	0,787
	0,724	0,748	0,834	1,199	0,777	0,840	0,604	0,597	0,639
	0,703	0,730	0,717	0,706	0,740	0,854	0,556	0,622	0,697
	0,712	0,744	0,772	0,691	0,759	0,867	0,562	0,587	0,654
	0,722	0,739	0,794	0,780	0,821	0,838	0,596	0,596	0,732
	0,718	0,709	0,786	1,754	2,023	0,832	0,584	0,606	0,748
	0,719	0,721	0,727	0,731	2,411	2,420	0,558	0,599	0,699
	0,714	0,735	0,739	0,705	0,813	0,968	0,576	0,598	0,703
Arithmetic mean	**0,722**	**0,731**	**0,757**	**0,903**	**1,264**	**1,035**	**0,581**	**0,600**	**0,693**
Standard deviation	0,021	0,014	0,041	0,345	0,655	0,491	0,026	0,011	0,053

Network Equipment Testing: Wait interval between packet sending: 0,250s									
Switch	OVS			Zodiac			3COM		
Packet size (bytes)	**66**	**135**	**256**	**66**	**135**	**256**	**66**	**135**	**256**
Collected data	0,697	0,754	0,762	0,784	0,751	1,345	0,561	0,588	0.634
	0,699	0,724	0,729	0,711	0,762	1,785	0,586	0,644	0,658
	0,695	0,758	0,734	0,750	0,888	1,645	0,595	0,569	0,616
	0,724	0,747	0,755	0,712	0,797	0,850	0,574	0,573	0,617
	0,717	0,744	0,743	0,743	0,744	1,841	0,554	0,584	0,621
	0,722	0,736	0,728	0,764	0,790	1,606	0,563	0,575	0,619
	0,703	0,740	0,741	1,387	0,744	1,367	0,547	0,582	0,622
	0,693	0,741	0,744	0,681	0,762	0,834	0,552	0,586	0,632
	0,698	0,744	0,738	1,056	1,820	0,831	0,548	0,601	0,652
	0,717	0,739	0,735	1,394	0,794	0,848	0,569	0,598	0,643
Arithmetic mean	**0,707**	**0,743**	**0,741**	**0,898**	**0,885**	**1,295**	**0,565**	**0,590**	**0,631**
Standard deviation	0,012	0,009	0,011	0,280	0,331	0,420	0,016	0,022	0,016

Network Equipment Testing: Wait interval between packet sending: 0,500s									
Switch	OVS			Zodiac			3COM		
Packet size (bytes)	**66**	**135**	**256**	**66**	**135**	**256**	**66**	**135**	**256**
Collected data	0,727	0,744	0,766	0,713	2,234	0,838	0,585	0,599	0,643
	0,735	0,739	0,756	1,351	0,764	0,705	0,573	0,597	0,690
	0,731	0,744	0,741	0,735	0,891	0,850	0,593	0,643	0,620
	0,713	0,751	0,762	0,714	0,752	0,826	0,567	0,588	0,615
	0,724	0,734	0,833	0,698	0,761	0,821	0,594	0,607	0,634
	0,834	0,741	0,729	0,716	0,756	0,833	0,542	0,599	0,616
	0,750	0,746	0,718	0,714	0,747	0,829	0,587	0,593	0,612
	0,744	0,731	0,729	0,700	2,194	1,402	0,548	0,667	0,648
	0,731	0,758	0,764	0,705	0,761	0,825	0,601	0,592	0,618
	0,729	0,753	0,724	0,705	0,772	0,856	0,561	0,632	0,631
Arithmetic mean	**0,742**	**0,744**	**0,752**	**0,775**	**1,063**	**0,879**	**0,575**	**0,612**	**0,633**
Standard deviation	0,034	0,008	0,034	0,203	0,608	0,189	0,020	0,026	0,024

Network Equipment Testing: Throughput				
Packet size (bytes)	128 Kbps			
Switch	**OVS**	**Zodiac**	**3COM**	**TP-Link**
Collected data	462,000	5,750	93,900	464,000
	462,000	5,610	93,900	465,000
	464,000	4,040	93,900	465,000
	461,000	4,080	93,900	464,000
	463,000	5,880	93,800	465,000
	463,000	6,310	93,800	467,000
	464,000	5,880	93,900	465,000
	462,000	4,980	93,900	466,000
	463,000	4,900	93,800	464,000
	463,000	4,070	93,800	464,000
Arithmetic mean	**462,700**	**5,150**	**93,860**	**464,900**
Standard deviation	0,949	0,858	0,052	0,994

Scenario 1 - Packet loss – Packet size: 135 bytes			
Interval	**0,01s**	**0,1s**	**0,5**
Collected data	14,000	2,000	1,000
	12,000	1,000	0,000
	14,000	2,000	0,000
	13,000	1,000	0,000
	15,000	1,000	0,000
	12,000	2,000	0,000
	13,000	1,000	1,000
	14,000	1,000	0,000
	14,000	2,000	0,000
	14,000	2,000	0,000
Arithmetic mean	**13,500**	**1,500**	**0,200**
Standard deviation	0,972	0,527	0,422

Appendix B

Leveraging Virtualization Technologies to Improve SCADA ICS Security

T Cruz, R Queiroz, J Proença, P Simões, E Monteiro

Department of Informatics Engineering
University of Coimbra, Portugal
E-mail: tjcruz@dei.uc.pt; rqueiroz@student.dei.uc.pt; jdgomes@dei.uc.pt; psimoes@dei.uc.pt; edmundo@dei.uc.pt

Abstract: *In recent years, Supervisory Control and Data Acquisition (SCADA) Industrial Control Systems (ICS)–systems used for controlling industrial processes, power plants, or assembly lines–have become a serious concern because of security and manageability issues. While the introduction of virtualization technologies has been instrumental in helping ICT infrastructures deal with such problems, their adoption in the ICS domain has been slow, despite recent developments such as the introduction of hypervisors or software-defined networking. This paper provides an overview of the usage of such technologies to improve SCADA ICS security and reliability; it also proposes advanced use cases.*

Keywords: *Virtualization, Critical Infrastructure Protection, Industrial Control Systems*

Introduction

In recent years, SCADA ICS–systems used for controlling power plants, assembly lines, or industrial processes, often part of critical and/or strategic infrastructures–have become a serious concern because of security and manageability issues. After years of air-gaped isolation, the increased coupling of ICS and ICT systems, together with the absence of proper management and security policies (Krutz 2006), disclosed several weaknesses in SCADA ICS, which were left exposed to attacks and potentially catastrophic consequences. These problems hardly constitute any novelty within the ICT domain, which has dealt with them for decades, prompting the development of specific tools and protocols, as well as for the establishment of management frameworks, such as Information Technology Infrastructure Library (ITIL) change management (Galup *et al.* 2009) or security-oriented policies.

However, ICT-specific practices cannot be easily ported to the ICS domain. For ICS operators, equipment manufacturers, and software developers alike, reliability is the top priority. Continuous operation and operational safety targets make it difficult to deploy several ICT-specific strategies and tools because of the potential impact on the ICS. This has pushed the industry, researchers, and standardization organizations to conceive ICS-specific security and management solutions and frameworks, as well as to publish guidelines documenting best practices. New product lines have also been introduced, with added security features and management capabilities.

Still, the ICS paradigm itself remained relatively unchanged, as proposed solutions try to fix what is wrong without attempting to introduce significant change into existing systems. This solution is far from optimal, as typical lifecycle-management operations, such as security patch deployment, are still an issue in modern SCADA ICS, the same being true for change management. In contrast, these issues have been addressed in the ICT domain for years through the continuous development of technologies, tools, and practices designed to address such needs. Virtualization technologies, which influence ICT computing and communications infrastructures, are among these developments. Developments such as hypervisors, Software-Defined Networking (SDN), or Network Function Virtualization (NFV) are reshaping the ICT ecosystem, providing the means to rationalize the use of computing and communications resources, also being instrumental to optimize and/or to improve aspects such as lifecycle management, energy efficiency, reliability, or security, among others.

From an ICS-security and -reliability perspective, device and infrastructure virtualization may have a similar impact as they had for ICT, as the industry slowly starts to absorb some of the technologies customized and fine-tuned for critical infrastructure environments. However, this process is still in early stages, not only because the specific ICS use cases for several virtualization technologies have yet to be developed, but also because extensive testing is required for its certification in such environments. In this scope, this paper consists of an extended version of an earlier article (Cruz *et al.* 2016)– analysing the application of virtualization technologies for communications and

computing resources in ICS contexts, with a focus on recent developments, open challenges, and benefits, from a security and reliability-oriented perspective.

The rest of this paper is structured as follows. The next section discusses the problem of security in ICS/SCADA, also explaining the potential benefits of introducing domain-aware virtualization technologies in such environments. Immediately following is a discussion of the introduction of network virtualization technologies in SCADA ICS and its security benefits. Next, the advantages of introducing partitioning hypervisors in ICS are addressed by describing a virtualized Programmable Logic Controller (PLC-) -use case. Finally, the authors present conclusions and insights about future developments.

Virtualization and SCADA ICS Security

As their scope was originally restricted to isolated environments, SCADA systems were considered relatively safe from external intrusion. However, as architectures evolved, these systems started to assimilate technologies from the ICT world, such as TCP/IP and Ethernet networking. This trend, together with the increasing adoption of open, documented protocols, exposed serious weaknesses in SCADA architectures, a situation that was aggravated by factors such as the use of insecure protocols, including Modbus (Triangle 2002) and inadequate product lifecycle-management procedures (Igure, Laughter & Williams 2006), the latter being responsible for the proliferation of devices and components beyond their end-of-life-support status. Also, the interconnection of the ICS network with organizational ICT network infrastructures, and even with the exterior (for remote management), brought a new wave of security incidents, with externally initiated attacks on ICS systems increasing significantly, especially when compared with internal attacks (Kang *et al.* 2011). Overall, this situation has become the root cause of many well-known ICS security incidents, such as the Stuxnet Trojan (O'Murchu & Falliere 2011).

In fact, ICS security cannot be approached in the same way as its ICT counterpart, as both domains differ significantly in terms of their fundamental design principles. Due to their

critical nature, ICS-operation and -design practices frequently privilege availability and reliability over confidentiality and data integrity—a perspective that is quite opposite from the ICT philosophy, which follows an inverse order of priorities (ISA-99.00.01).

The differences between the ICT and ICS domains also mean that there is no 'one-size-fits-all' solution when it comes to choosing and implementing security mechanisms. The fundamental premises for ICT security tools and commonplace lifecycle-management procedures, such as patching and updating a system, can become troublesome in an ICS, especially with situations such as the impediment/high cost of stopping production (Zhu *et al.* 2011), or even the explicit prohibition by the system's manufacturer, as any software release has to be certified before being released. Also, several security mechanisms, such as anti-virus software, are frequently ill advised by SCADA software providers, as they might interfere with the response latency of the host. The same rationale applies to anything deployed in the middle of the critical communications path (for example, an inline network Intrusion Detection System), as it may induce latency or some other sort of reliability issue.

Ironically, much of the problems faced by ICS are not entirely new, as they were known well before in the ICT domain, which has undergone several paradigm shifts and undertaken major technological steps to deal with them. More recently, the rise of the virtualization paradigm has become instrumental in changing the ICT computing landscape and providing the means to leverage computing and communications resources through consolidation and efficient management. Technologies such as hypervisors, SDN, or NFV are contributing to rationalizing, streamlining, and reshaping of infrastructures and devices, up to the point of changing the way communications and computing resources are consumed by end-users.

In terms of security and reliability, the impact is manifold. For instance, by creating a virtual machine (VM) snapshot, it is possible to rollback changes in case of failure or corruption caused by a failed OS patch or malicious tampering; VMs can be cloned for

sandboxed testing, prior to deployment into production; hypervisors can perform in-place behaviour monitoring of instances for security and safety purposes. Similarly, technologies such as SDN, which constitute a flow-oriented virtualization mechanism for networks, allow for the flexible creation and management of network overlays on top of existing physical infrastructures, while also enabling significant security and reliability benefits (Proença *et al.* 2015). NFV, in its turn, can work together with SDN to virtualize network equipment functionality, spreading it across the communications and computing infrastructure in an efficient and rational way, and also enabling the creation of innovative security solutions designed to better couple with the increasingly distributed nature of modern ICS and associated threats (Cruz *et al.* 2015).

But the introduction of ICT-like virtualization techniques in ICS is not a straightforward process. For operators, equipment manufacturers, and software developers alike, reliability, operational safety, and continuous operation are top priorities, which make it difficult to deploy several IT-specific strategies and tools, because of the potential impact on the ICS. For example, the latency overhead of certain mechanisms may not be compatible with real-time operation requirements. Hypervisors must cope with the (soft) real-time requirements of ICS applications; any attempt to introduce SDN or NFV must account for the potential impact in terms of ICS reliability or latency.

Despite the constraints, the potential efficiency, security, and reliability benefits for ICS are enough to justify the progressive development and introduction of domain-aware virtualization technologies. For instance, real-time hypervisors can provide safe partitioning and isolation, which will enable the creation of managed execution environments for real-time workloads, with continuous assessment of partition behaviour, and also provide rollback capabilities for potentially compromised systems. Use of SDN technologies can provide the ICS operator with the means to monitor the ICS communications infrastructure behaviour, while easing the implementation of countermeasures and deployment of security mechanisms. As ICS become increasingly distributed, NFV can provide the means to efficiently spread functional security components across the ICS communications and computing infrastructure in order to

better couple with the dispersed nature of the protected systems. The next section of this article will discuss how domain-aware virtualization can provide effective security benefits for ICS, with a focus on two major scopes: communications and computing.

Virtualization of SCADA ICS Communications Infrastructures

This section is specifically concerned with the introduction of SDN and NFV technologies within the SCADA ICS scope. For this purpose, the security benefits of the technologies hereby discussed will be analysed from a broad perspective, both in terms of the physical ICS dimension and dispersion of its scope, ranging from plant-level to distributed Industrial Automation and Control Systems (IACS) use cases. All sections will start with a brief introduction of their respective cornerstone concepts, namely SDN and NFV, in order to ease their introduction in the context of SCADA ICS security.

SDN and SCADA ICS

In conceptual terms, network architectures encompass three planes, which represent different areas of operation (Kreutz *et al.* 2014; Ellanti *et al.* 2005), as illustrated in **Figure 1**, below: management, control, and data. In this model (there are other variations), each plane has a specific function in terms of data transmission and network operations.

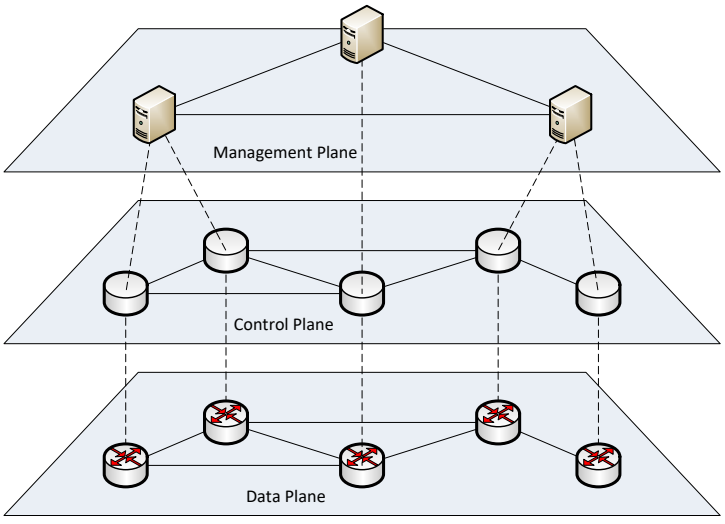


Figure 1. Network planes (adapted from Kreutz *et al.* 2014)

In this model, each plane plays a well-defined role, each one with its own characteristics:

- The **management plane** corresponds to traffic generated by services used for network infrastructure provision, maintenance, and monitoring. Such traffic can be transported through in-band (sharing the same link as user/normal traffic) or out-of-band (OOB) connections (a separate link/connection dedicated for management operations) (Schudel & Smith 2007).
- The purpose of the **control plane** (or signalling plane) is to support the setup of the data plane, including traffic between network elements related with policy or routing information exchanges. This is the case with switches, which may use specific protocols to exchange bridge information among them in order to infer topology information and to avoid loops. Control-plane traffic includes signalling, routing information, and link-state protocols, among other types of traffic (Schudel & Smith 2007).
- The **data plane** (also referred as the user plane, forwarding plane, carrier plane, or bearer plane) is responsible for carrying user data. Traffic belonging to this plane does not involve source or destination IP addresses belonging to network elements, such as routers or switches,, as it is expected to involve only end devices, such as computers and servers (Schudel & Smith 2007), which use the network for transport purposes.

SDN departs from the vertical integration that is characteristic of the traditional networking model, proposing an architecture that decouples forwarding functions (data plane) and network control (control plane), with the aim of introducing direct programmability into the network, to applications and policy engines alike (Kreutz *et al.* 2014). The control plane is moved outside the forwarding network elements and placed in a logically centralized controller (whose functionality may be spread among several instances, to improve scalability and resilience (Yeganeh, Tootoonchian & Ganjali 2013)), with the data plane remaining in place. The term SDN (for 'Software Defined Networking') was first introduced in an article (Greene 2009) referring to the Openflow project (ONF 2012) at the time being developed at the University of Stanford, which eventually became one of the first SDN-enabling standards.

With SDN, packet forwarding is flow oriented, meaning both origin and destinations are taken into account, instead of just packet destination, as in traditional networking. The SDN controller manages flow policies for a range of forwarding elements, effectively

moving such functions out of the devices. Thus, SDN-capable elements can be dynamically reconfigured over the network accordingly with the needs of network services and applications. For this reason, the controller will have a broader view of the domain, contrasting with the narrow view that an individual forwarding element has in a traditional IP network. **Figure 2** illustrates the flow-rule table of the OpenFlow protocol (one of the most popular SDN protocols).

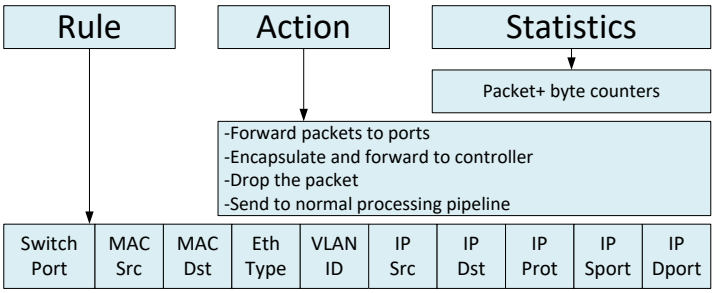


Figure 2: Openflow flow-rule table (adapted from SDX Central 2014)

Leveraging SDN for SCADA/ICS security

SDN allows for increased network flexibility and programmability, in particular for complex scenarios, which benefit from the reduced overhead for management operations such as topology changes for implementing overlay networks. Besides these benefits, SDN can also provide an effective mechanism for security applications (Proença *et al.* 2015). This is due to the fact that a centralized element with a global view of all the network entities (such as devices, flows, and network elements) is able to provide more efficient information-gathering and security-reaction mechanisms, especially when compared with the narrow local view individually provided by each forwarding element in traditional IP networks. For instance, an Openflow controller can provide information useful for online analysis and detection of security issues, as suggested by Braga, Mota, and Passito (2010):

- **Packets per flow:** this counter can be used for slow rate DDoS detection, as such attacks usually rely on the transmission of a reduced number of packets from a large amount of sources;
- **Average bytes per flow:** this can be used to detect small payload sizes, which are frequent in DDoS attack flows, in order to increase the attack efficiency;
- **Average duration per flow:** an SDN flow is deleted from its flow table if left inactive (no packets received) for a period of time, a feature which can be used to detect short flows characteristic of DoS attacks (Sadre, Sperotto & Pras 2012);

- **Percentage of pair-flows:** an asymmetry between flows coming into and out of the network can be an indicator of an ongoing DDoS attack (Kreibich 2005);
- **Number of single-flows:** it is possible that the number of unpaired flows increases dramatically in the beginning of a flood attack. This can be calculated on a per interval basis after subtracting the paired flows from the total;
- **Number of used TCP/IP ports and addresses:** DDoS frequently involve random spoofing of IP and ports, whose rate of increase may reveal ongoing issues.

Moreover, flow-based forwarding can be used to increase the efficiency of a reaction, being used to isolate or divert flows, instead of simply blocking an attack. This is useful to improve existing security techniques—for example, dynamically diverting attackers to honeypot systems as soon they are detected. SDN can also help handling Denial of Service (DoS) and Distributed DoS (DDoS) attacks by improving detection and reaction mechanisms.

Besides the generic security application scenarios, there have been several developments regarding SDN-based security mechanisms for ICS. For instance, Dong *et al.* (2015) propose reinforcing the resilience of SCADA networks used for smart grid applications using a solution relying on three elements (SCADA master, SDN controller, Intrusion Detection System—IDS), which coordinate with each other in order to detect attacks and reconfigure the network so as to mitigate and overcome identified problems. Suggested use cases include the dynamic establishment of routes to transmit control commands only when necessary (to shorten the time window for tampering attempts), automatic rerouting or dropping of suspicious packets to avoid spoofing or flooding attacks from compromised SCADA elements, or implementation of network monitors to deal with delay attacks.

Irfan & Mahmud (2015) propose using SDN for dynamic creation of virtual networks in order to isolate distinct traffic and hosts, and to enable traffic prioritization and secure partitioning. The concept is demonstrated using an SDN-controller proxy to create three isolated networks, which share the same physical infrastructure but have their own SDN controllers. Authors discuss the use of this architecture to improve aspects such as

authentication, confidentiality, integrity, non-repudiation, and availability. A similar approach is also suggested by Machii *et al.* (2015) as a way to minimize the attack surface by using SDN to dynamically segregate fixed functional groups within the ICS. A dynamic zone-based approach is also proposed, taking advantage of the information obtained from field devices to estimate the operation phase of the ICS (as each phase—such as start-up, normal operation, or load-change—exhibits different behaviour and communications profiles) and to calculate the optimal zone topology, deploying the needed SDN configuration in runtime. This strategy reduces the time and spatial exposure to attacks (effectively creating a moving target) and also provides the means to isolate compromised devices.

Also related to dynamic configuration techniques, Chavez *et al.* (2015) present a security solution based on network randomization, which also encompasses an IDS with near real time reaction capabilities. This network randomization approach assigns new addresses to network devices in a periodic basis or by request, in order to protect them against attacks that rely on knowledge about the ICS topology (such as static device addresses). The responsible controller application keeps an updated database of all the network specifications (mostly devices and real addresses), generating overlay IP addresses for the same devices and for each flow, which are used to define the OpenFlow rules on flow tables. This way, all the traffic flowing on the network uses 'fake' overlay addresses that are periodically randomized, reducing their useful lifetime and, consequently, the time window available for any attacker to take advantage of that knowledge. The proposed IDS takes advantage of the predictable, auto-similar, traffic patterns of ICS networks for identifying attacks and triggering defence reactions (a network randomization request, which will render useless any ongoing attack using old overlay addresses). Attack detection makes use of machine learning algorithms and mathematical methods, fed and trained using OpenFlow's statistical counters.

Silva *et al.* (2015) also describe a dynamic technique that makes use of SDN to prevent eavesdropping on SCADA networks. The intended goal is to deter attackers from collecting sequential data, which is essential for breaking encryption, identifying patterns,

and retrieving useful information from the payload. By taking advantage of redundant network connectivity, a multi-path routing mechanism enables a flow to be transmitted and split over different paths (see **Figure 3**, below) by resorting to an algorithm that calculates the shortest path between two devices, dynamically assigns a cost to each one, and uses an OpenFlow timer (hard timeout) to periodically reinstall new flow rules.

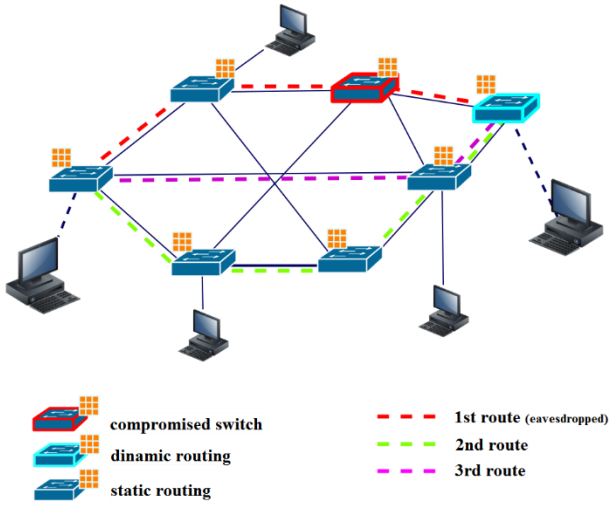


Figure 3: Multi-flow, redundant routing for flow splitting (adapted from Silva *et al.* 2015)

Genge *et al.* (2016) propose two distinct SDN-based techniques to mitigate and block ICS cyber attacks. The first technique (see **Figure 4**, below), designed for single-domain networks, attempts to mitigate DoS attacks by rerouting traffic, using information from the SDN controller. SDN controllers feed an application that continuously monitors the state of the network links and communicates with the controller to issue flow reconfiguration operations. Once an attack is detected (few details are provided about this, though), the corresponding data flows are rerouted, in order to protect the ICS.

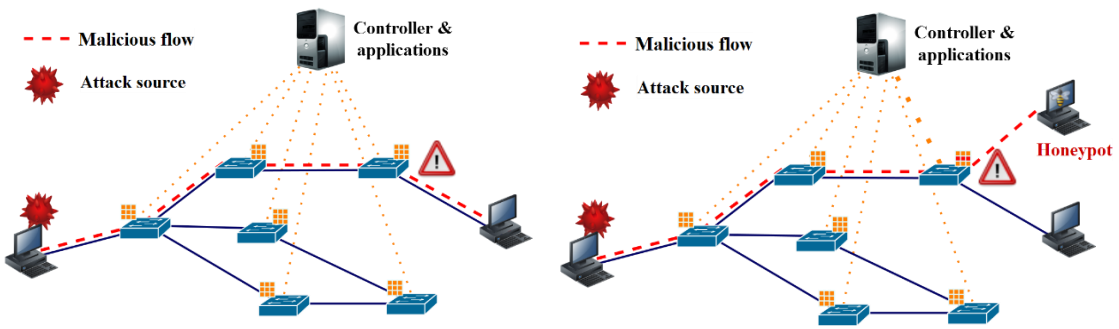


Figure 4: A single-domain SDN-based security solution (adapted from Genge *et al.* 2016)

The second technique (see **Figure 5**) targets multi-domain networks, with the goal of blocking the attack as close as possible to the entry point in the network.

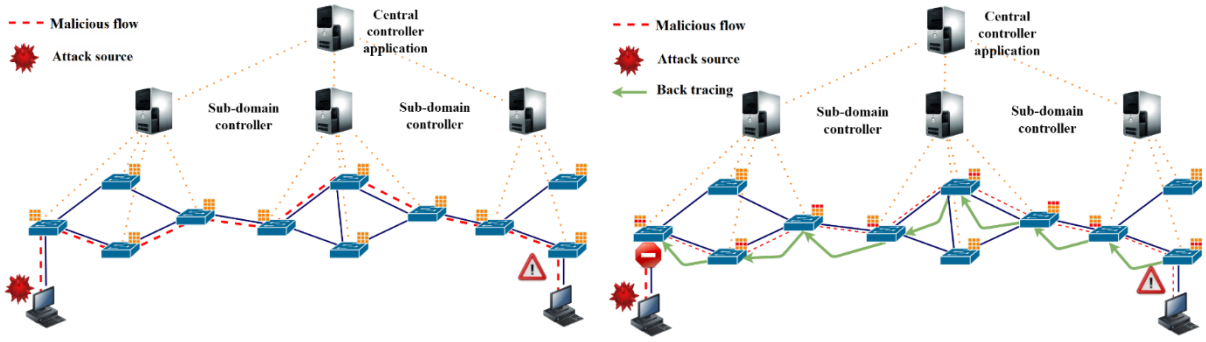


Figure 5: A multiple-domain SDN-based security solution (adapted from Genge *et al.* 2016)

For such a multi-domain network, each domain has its own OpenFlow controller, connected to a centralized security application. This application receives information from the SDN controllers, which have access to a global perspective about the network. Once an attack is detected, the security application will backtrack towards its origin by recursively issuing queries about the related flows to identify the previously paired nodes until the original network entrance point is found.

ICS-specific honeypots and honeynets can also benefit from the introduction of SDN technologies. Honeypots are decoy or dummy targets set up to attract and detect/profile attacks. Exposed to probing and attack, these targets are used to lure and track intruders as they advance (Simões *et al.* 2013), revealing any scouting activities. Traditionally, honeypot systems live in unused address space in the system, waiting for attackers to find them (Spitzner 2003), but their operation can be greatly improved by SDN, which has the possibility of turning them into a more proactive defence.

Using SDN network-flow manipulation capabilities, it is possible to improve honeypot operation and transform it into an active security component by working together with

other mechanisms, such as network intrusion detection systems (NIDS). When an unauthorized activity is detected by a NIDS, the SDN controller can divert the anomalous traffic flows to an ICS-specific honeypot, such as the one proposed by Simões *et al.* (2013). The attacker would not be aware of this diversion and would continue the attack. Meanwhile the honeypot will log its activity for forensics analysis. **Figure 6**, below, illustrates an example of this approach.

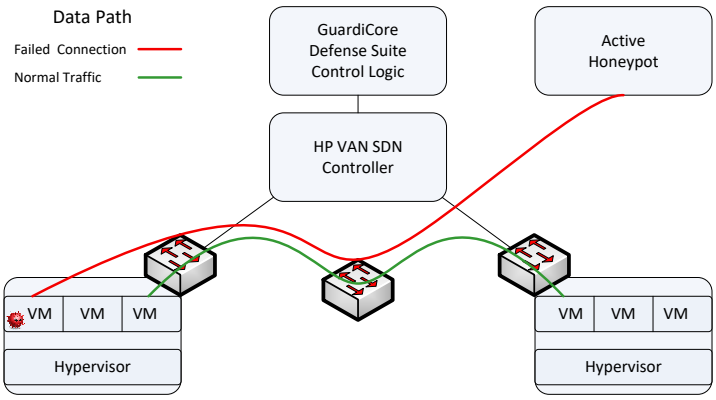


Figure 6: Active honeypot (reproduced from Hewlett-Packard 2014)

Also, Song, Shin, and Choy (2014) suggested using honeynets (networks set up with several honeypot devices) together with SDN technologies to detect scouting procedures and collect profiling information about attackers. This is achieved by providing the attacker with false information from the honeynet, using OpenFlow to detect the scan attacks by inspecting packets coming towards closed or unused ports, or to detect corrupt packets or sessions. After a successful detection, the infringing packet and the subsequent ones in the same flow will be redirected to the honeynet. Despite being a generic proposal, this solution can be easily ported to most ICS infrastructures.

Network Function Virtualization and distributed ICS

NFV is the result of the convergence between telecommunications infrastructures and infrastructure virtualization. As network applications and services scale and evolve (not only in sheer capacity requirements, but also in complexity), they impose an added burden to the supporting telecommunications provider infrastructure, requiring the use of specific

network management and traffic policies that cannot be provided by the network. As Chiosi *et al.* (2012) have noted, from this perspective, NFV is a significant development as it enables the creation of flexible and on-demand network services through a service chain-based composition mechanism that uses network functions implemented in VNF (Virtualized Network Functions) components comprising functionality such as NAT, IDS, Firewalls or other service modules implemented as VM appliances.

The NFV vision attempts to decouple network capacity from functionality, by conceiving an end-to-end service as an entity that can be modelled and described by means of network function forwarding graphs (**Figure 7**) involving interconnected VNFs and endpoints (also known as service chaining).

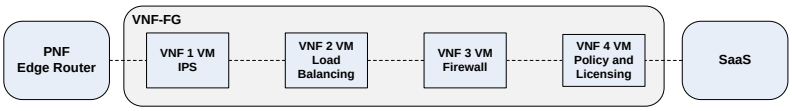


Figure 7: NFV Forwarding Graph example

This approach allows for creation of differentiated end-to-end services that can be provided by the (ordered) combination of elementary VNF or physical functions, chained together by a Forwarding Graph, which models the service flows (see **Figure 8**, below). Furthermore, VNF FGs can be nested to define complex functions. VNFs are implemented in software, being interconnected through the logical links that are part of a virtualized network overlay, which can be implemented using SDN.

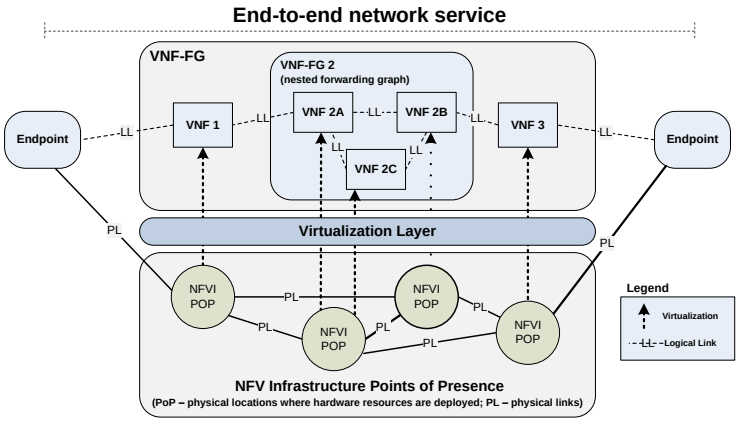


Figure 8: NFV end-to-end service with VNFs (adapted from Ersue 2013)

Eventually, even Physical Network Functions (conventional network devices with close coupled software and hardware that perform network functions) can be involved in a Network Forwarding Graph service chain (the concept of service chain is not exclusive of NFV). A virtualization layer abstracts the physical resources (computing, storage, and networking) on top of which the VNFs are deployed and implemented, with the supporting NFV Infrastructure (NFVI) being spread across different physical locations, called Points of Presence (NFVI PoPs), as shown in **Figure 5**, above.

NFV as an enabler for a new generation of distributed IACS

Use cases such as Internet of Things (IoT), wire-to-water generation, micro generation, smart metering or smart water management constitute a new generation of distributed IACS that can only be supported with the help of a complex distributed software stack, potentially also requiring the involvement of third-parties, such as telecommunications and cloud operator infrastructures—for this reason, the introduction of Network Function Virtualization component appliances, distributed across geographically dispersed infrastructure PoPs, makes entire sense.

As the IACS enters the customer premises, the NFV service abstraction model (services as composition of VNFs) provides an effective way to introduce support components along the service path. For instance, a data collection and analysis VNF can be added to the customer service chain (eventually within a virtual Business Gateway service

abstraction) to provide data collection for smart metering scenarios. The same rationale applies for security purposes, as cyber-physical protection (for example, to implement bump-in-the-wire encryption) or security anomaly detection VNFs can be integrated within service chains, also using SDN to create flexible security monitoring and reaction capabilities. Moreover, Distributed IDS (DIDS) components may be consolidated in the form of VNFs optimally deployed in order to reduce service overhead and rationalize resources. For instance, the DIDS components might be deployed in the form of VNFs, either shared among several Business Gateway FGs or used exclusively by a service instance (Cruz *et al.* 2015). Some manufacturers (RAD 2015) (ECI 2015) are starting to propose NFV products for ICS applications that implement this philosophy, incorporating NFV capabilities in access nodes for optical transport or packet switched networks, for hosting firewall, encryption or traffic monitoring VNFs.

NFV is also an enabler for fog computing scenarios. The term ‘fog computing’, frequently also referred as ‘edge computing’, is based on the idea that, rather than hosting and working from a centralized cloud, some parts of the infrastructure may be deployed on network ends, using virtualized platforms located between end-user devices and the cloud data centres. It attempts to provide better quality of service in terms of delay, power consumption, and reduced data traffic over the Internet, among other benefits. Fog computing tries to address the need to process large data streams in real time while working within the limits of available bandwidth, by placing some of transactions and resources at the edge of the cloud, thus improving the efficiency of the infrastructure by offloading processing tasks before passing them to the cloud.

The NFV paradigm is naturally compatible with fundamental premises for implementation of fog-computing distributed topologies. As such, it is envisioned that distributed awareness and IACS cyber-security detection capabilities will take advantage of the NFV paradigm to support their underlying deployment model, departing from the conventional, self-contained model and moving towards an architecture capable of keeping up with the geographically dispersed nature of IoT IACS. Also, the VNF deployment criteria may consider the availability of specific capabilities (such as raw

processing capacity) in a specific NFVI POP. For instance, per-subscriber security-event processing components may be hosted in a different NFVI POP from the one(s) hosting other VNFs for the DIDS service.

Real-time Hypervisors + SDN = Towards a Virtualized PLC

Born in the mainframe era, Virtual Machine Monitors (also called Hypervisors) have ultimately evolved towards being supported in open, Commercial Off-The-Shelf (COTS) hardware, bringing a significant improvement for the ICT ecosystem, allowing for co-hosting of several VMs within a host machine, sharing resources, and providing a managed execution environment. Specifically, type-1 (bare metal) hypervisors have become popular in large-scale virtualization scenarios such as data centres, bringing several benefits in terms of resource consolidation, business continuity, scalability, management, and security.

However, most type-1 hypervisors are optimized for ICT loads, and, thus, are unsuitable for several ICS application use cases, mostly due to the overhead of the mediation and translation mechanisms abstracting the host hardware from the VM. This situation gradually began to change, as some operators started virtualizing hosts with services deployed on general-purpose OS, such as SCADA Master Stations (MS), Human-Machine Interfaces (HMI) or Historian Database servers (HDB), using conventional type-1 hypervisors. This was possible due to developments that allowed such hypervisors to benefit from hardware-assisted memory management and I/O mechanisms to implement robust resource affinity and reservation (such as VT-d and PCI SRV-IO; see Garcia-Valls, Cucinotta & Lu 2014), thus, providing performance guarantees while avoiding the effect of resource overprovisioning. Also, real-time clock integrity issues, one of the main concerns in hypervisor environments, were mostly solved using para-virtualized interfaces (KVM 2015) and/or adequate clock synchronization policies.

Other ICS elements, such as process control devices, can also potentially benefit from virtualization technologies. For instance, (Cahn *et al.* 2013) proposed the virtualization

of Intelligent Electronic Devices (IEDs) used to collect information from sensors and power equipment, with the purpose of optimizing the maintenance and cost overheads, while increasing reliability. The same rationale could be applied to Programmable Logic Controller (PLC) devices, which constitute the focus of this section.

PLCs are pervasive components in ICS, such as SCADA systems, being designed to control industrial processes autonomously or as part of a distributed-control system topology. While the success of the PLC may be explained by its robustness and reliability, it is one of the most enduring legacies in modern ICS, having evolved very little over the last years. Modern PLCs are the outcome of an evolutionary process that started with the first generation of relay-based devices, progressively incorporating technologies such as microprocessors and microcontrollers, Real-Time Operating Systems (RTOS) and communications capabilities ranging from serial point-to-point or bus topologies to Ethernet and TCP/IP. Although modern PLCs are often embedded devices running Real-Time Operating Systems (RTOS), equipped with System-on-Chip or CPUs (PowerPC, x86 or ARM) based on commodity Instruction Set Architectures (ISA), their virtualization was not deemed feasible until recently, due to the lack of specific hardware, software, and infrastructure support.

Towards the virtual PLC

PLCs are designed for reduced and deterministic latency, operating under strict timing constraints that are dependent on factors such as the end-to-end and event response latencies across components on interconnected buses, or signal and message propagation delays. These requirements are incompatible with the use of several virtualization technologies, such as conventional type-1 hypervisors, due to overhead issues and the lack of support for real-time payloads.

However, recent developments, such as the implementation of low-latency deterministic network connectivity for converged Ethernet and the availability of real-time hypervisors, have made it possible to virtualize components of the PLC architecture. The vPLC architecture described by Cruz, Simões & Monteiro *(*2016) takes advantage of these

capabilities by decoupling the PLC execution environment from I/O modules using an SDN-enabled Ethernet fabric to provide connectivity to the I/O subsystem (**Figure 9**, below). This architecture departs from the SoftPLC concept, as proposed by products such as (Codesys) or (ISaGRAF), by adopting an approach in line with (Intel 2013) and (IntervalZero 2010), with the added benefit of a convergent fabric scenario with SDN capabilities.

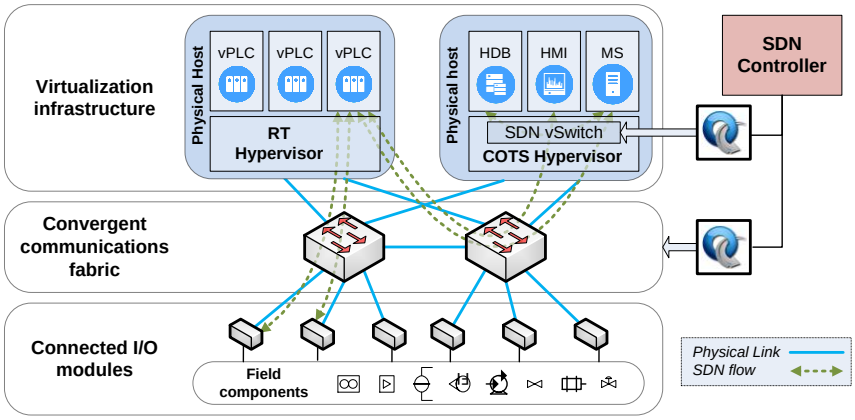


Figure 9: The vPLC architecture

In the vPLC, the PLC I/O bus is replaced by high-speed networking capabilities, with SDN allowing for the creation of flexible virtual channels on the I/O fabric, accommodating the connectivity flows between the vPLC instances and the I/O modules (such as sensor interfaces or motion controllers), and providing traffic isolation. Moreover, such I/O modules can be built with reduced complexity, thanks to recent progress in terms of Field-Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuit (ASIC) technology. SDN reconfiguration is managed by means of an SDN controller, via a High-Availability (HA) server (not depicted in the figure), which interacts with its northbound interface. The HA server continuously monitors the SDN switch statistics and path reachability, triggering reconfiguration procedures in case of performance degradation or failure.

This decentralized model shares similarities with remote or distributed I/O PLC topologies, with networked I/O modules acting as extensions of the PLC rack. This

architecture shares similarities with the Converged Plantwide Ethernet (CWpE) (Didier *et al.* 2011) proposal, or even critical avionics systems, which replace legacy interconnects with Ethernet-based technologies, such as Avionics Full-Duplex Switched Ethernet (AFDX) (Fuchs 2012).

Advances in cut-through switching, together with Remote Direct Memory Access techniques (RDMA), particularly in converged Ethernet scenarios, have allowed for port-to-port latencies of the order of the hundredths of nanoseconds in 10G Ethernet switch fabrics and application latencies in the order of microseconds (Beck & Kagan 2011). Additionally, resources such as Intel's Data Plane Development Kit (DPDK) (Zhang *et al.* 2014) allow for the implementation of low-latency, high-throughput packet processing mechanisms that bypass kernels, thus, bringing the network stack into user space and enabling adapters to perform Direct Memory Access operations to application memory. This enables satisfying requirements for single-digit microsecond jitter and restricted determinism, allowing for bare-metal performance on commodity server hardware. On top of this, proposals such as the 802.1Qbv Time Sensitive Networking (IEEE TSN) standard provide compliance with real-time requirements in the microsecond range on conventional Ethernet.

As for computing resources, there are two factors that must be considered. First, modern x86 or ARM processors have become capable of replacing microcontrollers in standalone PLC applications (Kean 2010) because of improvements in terms of raw performance, low latency I/O mechanisms, or the availability of ISA extensions suitable for Digital Signal Processing tasks. Second, the availability of real-time static partitioning hypervisors, such as Jailhouse (Siemens), Xtratum (Crespo, Ripoll & Masmano 2010), X-Hyp (X-HYP) or PikeOS (Baumann *et al.* 2011), enables hosting RTOS guest VMs for real-time workloads. Some hypervisors, such as Xtratum and PikeOS, even replicate the ARINC 653 (Fuchs 2012) partitioning model for safety-critical avionics RTOS, with a Multiple Independent Levels of Security/Safety (MILS) (Alves-Foss *et al.* 2006) architecture.

The benefits of this approach are manifold. The price tag for entry-level PLCs is comparable to a COTS server that can host several vPLC instances, being kept out of the factory floor or industrial environment. Distributed I/O on converged Ethernet also provides cost-effective performance and reliability benefits, as communications between different vPLC instances can take place across the convergent fabric or even locally, if co-located on the same host, with SDN allowing for flexible creation of communications channels for differentiated requirements. Moreover, I/O modules—the components with highest failure rate in PLCs—can be easily and quickly replaced in case of failure.

Particularly, the potential advantages of the vPLC in terms of reliability, safety, and security are considerable, as it can take advantage of datacentre-like redundant power, computing, and communications resources. Other benefits are also envisioned, namely:

- Hypervisors allow for migration of virtualized ICS components, as well as instance cloning for pre-deployment tests;
- PLC watchdogs and system-level debugging and tracing mechanisms can be implemented at the hypervisor level, which is able to oversee and control the vPLC partition behavior;
- vPLCs benefit from partitioning isolation, with VMs being easy to restore in a fresh state in case of tampering or other malicious activity;
- SDN-managed isolated I/O paths ease the implementation of flexible, on-demand protection mechanisms at the I/O level, thereby paving the way for the introduction of NFV components at the ICS level.

Overall, these benefits constitute strong arguments in support of the vPLC proposal. Moreover, most of them suggest that the vPLC could be feasible even for a single instance per device, using Industrial-grade Single Board Computers, instead of COTS servers.

Conclusion

This paper discusses the implications of the progressive introduction of virtualization technologies in ICS, with a special focus on security and reliability aspects. The virtualization of both network and computing virtualization was analysed from an ICS-

centric standpoint, covering recent developments as well as proposing new use cases and approaches to improve network and systems security.

Starting with an overview of network virtualization technologies, such as SDN and NFV and their application within ICS and distributed IACS, the paper next addressed the issue of using hypervisor technologies for real-time workloads. In this latter perspective, a virtual PLC (vPLC) architecture was discussed, which transcends the simple virtualization of the PLC device, constituting an integrated approach in which the device merges with the infrastructure in a seamless way. The vPLC takes advantage of network and computing virtualization technologies to propose a converged approach for plan-wide consolidation of the ICS infrastructure, with performance, cost, and security benefits. This proposal is presently under development by a team that includes the authors of this paper.

Appendix C

Security implications of SCADA ICS virtualization: survey and future trends

T. Cruz, R. Queiroz, P. Simões, E. Monteiro
University of Coimbra, Portugal

{tjcruz, rqueiroz, psimoes, edmundo}@dei.uc.pt

Abstract: In recent years, Supervisory Control and Data Acquisition (SCADA) Industrial Control Systems (ICS) – a kind of systems used for controlling industrial processes, power plants or assembly lines – have become a serious concern because of security and manageability issues. Years of air-gaped isolation, the increased coupling of ICS and Information and Communication Technology (ICT) systems, together with the absence of proper management and security policies, disclosed several weaknesses in SCADA ICS. Suddenly, these systems were faced with a reality that was familiar for ICT infrastructure managers for decades, which has driven the need for the development of specific technologies, as well as the establishment of management frameworks and the adoption of security-oriented policies. Virtualization was one of such developments, whose influence spawns several domains, from networking and communications to mass storage and computing resources.

For ICT, the rise of virtualization constituted a paradigm shift, with significant gains in terms of resource consolidation, manageability or even security. These benefits are yet to fully reach the ICS domain, despite recent developments geared towards the introduction of hypervisors or software-defined networking within such systems. This paper provides an overview on the usage of such technologies to improve SCADA ICS security and reliability also proposing advanced use cases.

Introduction

In recent years, SCADA ICS – a kind of systems used for controlling power plants, assembly lines or industrial processes, often part of critical and/or strategic infrastructures – have become a serious concern because of security and manageability issues. After years of air-gaped isolation, the increased coupling of ICS and ICT systems, together with the absence of proper management and security policies (Krutz 2006), disclosed several weaknesses in SCADA ICS, which were left exposed to attacks, with potentially catastrophic consequences. Nevertheless, these problems hardly constitute any novelty within the ICT domain, which has dealt with them for decades, driving the need for the development of specific tools and protocols, as well as the establishment

of management frameworks, such as Information Technology Infrastructure Library (ITIL) change management (Gallup 2009) or security oriented policies.

However, ICT-specific practices cannot be easily ported to the ICS domain. For ICS operators, equipment manufacturers and software developers alike, reliability is top priority. Continuous operation and operational safety targets make it difficult to deploy several ICT-specific strategies and tools, because of the potential impact on the ICS. This has pushed the industry, researchers and standardization organizations to conceive ICS-specific security and management solutions and frameworks, as well as publishing guidelines and guides documenting best practices. New product lines were also introduced, with added security features and management capabilities.

Still, the ICS paradigm itself remained relatively unchanged, as proposed solutions try to fix what is wrong without attempting to introduce significant change into existing systems. This solution is far from optimal, as typical lifecycle management operations such as security patch deployment are still an issue in modern SCADA ICS, the same being true for change management. In contrast, these issues have been addressed in the ICT domain for years, through the continuous development of technologies, tools and practices, designed to address such needs. Virtualization technologies are among these developments, which influence ICT computing and communications infrastructures. Developments such as hypervisors, Software-Defined Networking (SDN) or Network Function Virtualization (NFV) are reshaping the ICT ecosystem, providing the means to rationalize the use of computing and communications resources, also being instrumental to optimize and/or improve aspects such as lifecycle management, energy efficiency, reliability or security, among others.

From an ICS security and reliability perspective, device and infrastructure virtualization may have a similar impact as they had for ICT, as the industry slowly starts to absorb some of the technologies, customized and fine-tuned for critical infrastructure environments. However, this is a process undergoing its early stages, not only because the specific ICS use cases for several virtualization technologies have yet to be developed, but also because extensive testing is required for its certification in such environments. In this scope, this paper analyses the application of virtualization technologies for communications and computing resources in ICS contexts, with a focus on recent developments, open challenges and benefits, from a security and reliability-oriented perspective.

The rest of this paper is structured as follows. Section 2 discusses the problem of security in ICS/SCADA, also explaining the potential benefits of introducing domain-aware virtualization technologies in such environments. Section 3 discusses the introduction of network virtualization technologies in SCADA ICS and its security benefits. Section 4 addresses the advantages of

introducing partitioning hypervisors in ICS, describing a virtualized Programmable Logic Controller (PLC) use case. Finally, section 5 presents conclusions insights about future developments.

Virtualization and SCADA ICS security

As their scope was originally restricted to isolated environments, SCADA systems were considered relatively safe from external intrusion. However, as architectures evolved, these systems started to assimilate technologies from the ICT world, such as TCP/IP and Ethernet networking. This trend, together with the increasing adoption of open, documented protocols, exposed serious weaknesses in SCADA architectures, a situation that was aggravated by factors like the use of insecure protocols, such as Modbus (Triangle 2002) or inadequate product lifecycle management procedures (Igure 2006), the latter being responsible for the proliferation of devices and components beyond their end-of-life support status. Also, the interconnection of the ICS network with organizational ICT network infrastructures, and even with the exterior (for example, for remote management) brought a new wave of security incidents, with externally initiated attacks on ICS systems increasing significantly, especially when compared with internal attacks (Kang 2011). Overall, this situation has become the root cause of many well-known ICS security incidents, such as the Stuxnet Trojan (O'Murchu 2011).

In fact, ICS security cannot be approached in the same way as its ICT counterpart, as both domains differ significantly on their fundamental design principles. Due to its critical nature, ICS operation and design practices frequently privilege availability and reliability over confidentiality and data integrity – a perspective that is quite the opposite from the ICT philosophy, which follows an inverse order of priorities (ISA-99.00.01).

The differences between the ICT and ICS domains also mean that there is no "one size fits all" solution when it comes to choose and implement security mechanisms. The fundamental premises for ICT security tools and commonplace lifecycle management procedures, such as patching and updating a system, can become troublesome in an ICS, when faced with situations such as the impediment / high cost of stopping production (Zhu 2011), or even the explicit prohibition by the system's manufacturer, as any software release has to be certified before being released. Also, several security mechanisms, such as anti-virus software are frequently unadvised by SCADA software providers, as they might interfere with the response latency of the host. The same rationale applies to anything deployed in the middle of the critical communications path (e.g., an inline network Intrusion Detection System), as it may induce latency or some other sort of reliability issue.

Ironically, much of the problems faced by ICS are not entirely new, as they were known well before in the ICT domain, which has undergone several paradigm shifts and major technological steps to deal with them. More recently, the rise the virtualization paradigm has become instrumental in changing the ICT computing landscape, providing the means to leverage computing and communications resources, through consolidation and efficient management. Technologies such as hypervisors, SDN or NFV are contributing to rationalize, streamline and reshape infrastructures and devices, up to the point of changing the way communications and computing resources are consumed by end-users.

In terms of security and reliability, the impact is manifold. For instance, by creating a virtual machine (VM) snapshot it is possible to rollback changes in case of failure or corruption caused by a failed OS patch or malicious tampering; VMs can be cloned for sandboxed testing, prior to deployment into production; hypervisors can perform in-place behavior monitoring of instances for security and safety purposes. Similarly, technologies such as SDN, which constitute a flow-oriented virtualization mechanism for networks, allow for the flexible creation and management of network overlays on top of existing physical infrastructures, while also enabling significant security and reliability benefits (Proença 2015). NFV, in its turn, can work together with SDN to virtualize network equipment functionality, spreading it across the communications and computing infrastructure in an efficient and rational way, also enabling the creation of innovative security solutions designed to better couple with the increasingly distributed nature of modern ICS and associated threats (Cruz 2015).

But the introduction of ICT-like virtualization techniques in ICS is not a straightforward process. For operators, equipment manufacturers and software developers alike, reliability, operational safety and continuous operation are top priorities, a situation that makes it difficult to deploy several IT-specific strategies and tools, because of the potential impact on the ICS. For example, the latency overhead of certain mechanisms may not be compatible with real-time operation requirements. Hypervisors must cope with the (soft) real-time requirements of ICS applications; any attempt to introduce SDN or NFV must account for the potential impact in terms of ICS reliability or latency.

Despite the constraints, the potential efficiency, security and reliability benefits for ICS are enough to justify the progressive development and introduction of domain-aware virtualization technologies. For instance, real-time hypervisors can provide safe partitioning and isolation, enabling the creation of managed execution environments for real-time workloads, with continuous assessment of partition behavior, also providing rollback capabilities for potentially compromised systems. Use of SDN technologies can provide the ICS operator with the means to monitor the ICS communications infrastructure behavior, while easing the implementation of

countermeasures and deployment of security mechanisms. As ICS become increasingly distributed, NFV can provide the means to efficiently spread functional security components across the ICS communications and computing infrastructure, in order to better couple with the dispersed nature of the protected systems. The next two chapters will discuss how domain-aware virtualization can provide effective security benefits for ICS, with a focus on two major scopes: communications and computing.

Virtualization of SCADA ICS communications infrastructures

This chapter is specifically concerned with the introduction of SDN and NFV technologies within the SCADA ICS scope. For this purpose, the security benefits of the technologies hereby discussed will be analyzed from a broad perspective, both in terms of the physical ICS dimension and dispersion of its scope, ranging from plant-level to distributed Industrial Automation and Control Systems (IACS) use cases. All sections will start with a brief introduction of its respective cornerstone concepts, namely SDN and NFV, in order to ease its introduction in the context of SCADA ICS security.

SDN and SCADA ICS

SDN is an architecture that decouples forwarding functions (data plane) and network control (control plane), with the aim of introducing direct programmability into the network, to applications and policy engines alike. With SDN, packet forwarding is flow oriented, meaning both origin and destination are taken into account, instead of just packet destination, as in traditional networking. Flow policies are granted by an SDN controller, which manages the policies for a range of forwarding elements in a given network, effectively moving control plane functions outside of the devices. Thus, SDN-capable elements can be dynamically reconfigured over the network accordingly with the needs of network services and applications. For this reason, the controller will have a broader view of the domain, contrasting with the narrow view that an individual forwarding element has in a traditional IP network. There are several SDN protocols, among which OpenFlow (ONF) is one of the most popular.

SDN allows for increased network flexibility and programmability, in particular for complex scenarios, which benefit from the reduced overhead for management operations such as topology changes for implementing overlay networks. Besides these benefits, SDN can also provide an effective mechanism for security applications (Proença 2015). This is due to the fact that a centralized element with a global view of all the network entities – such as devices, flows and network elements – is able to provide more efficient information gathering and security reaction mechanisms, especially when compared with the narrow local view individually provided by each

forwarding element in traditional IP networks. Moreover, flow-based forwarding can be used to increase the efficiency of a reaction, being used to isolate or divert flows, instead of simply blocking an attack. This is useful to improve existing security techniques – for example, allowing to dynamically divert attackers to honeypot systems, as soon they are detected. SDN can also help handling Denial of Service (DoS) and Distributed DoS (DDoS) attacks, by improving detection and reaction mechanisms.

Besides the generic security application scenarios, there have been several developments regarding SDN-based security mechanisms for ICS. For instance, (Dong 2015) proposes reinforcing the resilience of SCADA networks used for smart grid applications using a solution relying on three elements (SCADA master, SDN controller, Intrusion Detection System – IDS), which coordinate with each other in order to detect attacks and reconfigure the network so as to mitigate and overcome identified problems. Suggested use cases include the dynamic establishment of routes to transmit control commands only when necessary (to shorten the time window for tampering attempts); automatic rerouting or dropping of suspicious packets to avoid spoofing or flooding attacks from compromised SCADA elements; or the implementation of network monitors to deal with delay attacks.

(Irfan 2015), proposes using SDN for dynamic creation of virtual networks in order to isolate distinct traffic and hosts, enabling traffic prioritization and secure partitioning. The concept is demonstrated using an SDN controller proxy to create three isolated networks, which share the same physical infrastructure, but have their own SDN controllers. Authors discuss the use of this architecture to improve aspects such as authentication, confidentiality, integrity, non-repudiation and availability. A similar approach is also suggested by (Machii 2015) as a way to minimize the attack surface, by using SDN to dynamically segregate fixed functional groups within the ICS. A dynamic zone-based approach is also proposed, taking advantage of the information obtained from field devices to estimate the operation phase of the ICS (as each phase, such as start-up, normal operation or load-change exhibit different behavior and communications profiles) and calculate the optimal zone topology, deploying the needed SDN configuration in runtime. This strategy reduces the time and spatial exposure to attacks, also providing the means to isolate compromised devices.

Also related to dynamic configuration techniques, (Chavez 2015) presents a security solution based on network randomization, complemented with an IDS capable with near real time reaction capabilities. This network randomization approach assigns new addresses to network devices in a periodic basis or by request, in order to protect them against attacks that rely on knowledge about the ICS topology (such as static device addresses). The responsible controller application keeps an updated database of all the network specifications (mostly devices and real addresses),

generating overlay IP addresses for the same devices and for each flow, which are used to define the OpenFlow rules on flow tables. This way, all the traffic flowing on the network uses "fake" overlay addresses that are periodically randomized, reducing their useful lifetime and, consequently, the time window available for any attacker to take advantage of that knowledge. The proposed IDS takes advantage of the predictable, auto-similar, traffic patterns of ICS networks for identify attacks and trigger defense reactions (a network randomization request, which will render useless any ongoing attack using old overlay addresses). Attack detection makes use of machine learning algorithms and mathematical methods, fed and trained using OpenFlow's statistical counters.

(Silva 2015) also describes a dynamic technique that makes use of SDN to prevent eavesdropping on SCADA networks. The intended goal is to deter attackers from collecting sequential data, which is essential for breaking encryption, identify patterns and retrieve useful information from the payload. By taking advantage of redundant network connectivity, a multi-path routing mechanism enables a flow to be transmitted and split over different paths (see Figure 1) by resorting to an algorithm that calculates the shortest path between two devices, dynamically assigns a cost to each one and uses an OpenFlow timer (hard timeout) to periodically reinstall new flow rules.

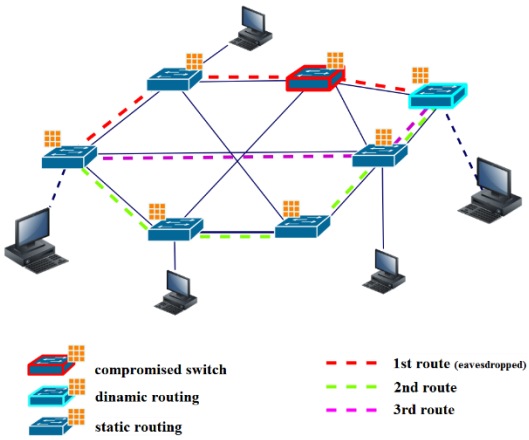


Figure 1: Multi-flow, redundant routing for flow splitting (adapted from (Silva 2015))

(Genge 2016) proposes two distinct SDN-based techniques to mitigate and block ICS cyber attacks. The first technique (see Figure 2), designed for single-domain networks, attempts to mitigate DoS attacks by rerouting traffic, using information from the SDN controller. SDN controllers feed an application that continuously monitors the state of the network links and communicates with the controller to issue flow reconfiguration operations. Once an attack is detected (few details are provided about this, though), the corresponding data flows are rerouted, in order to protect the ICS.

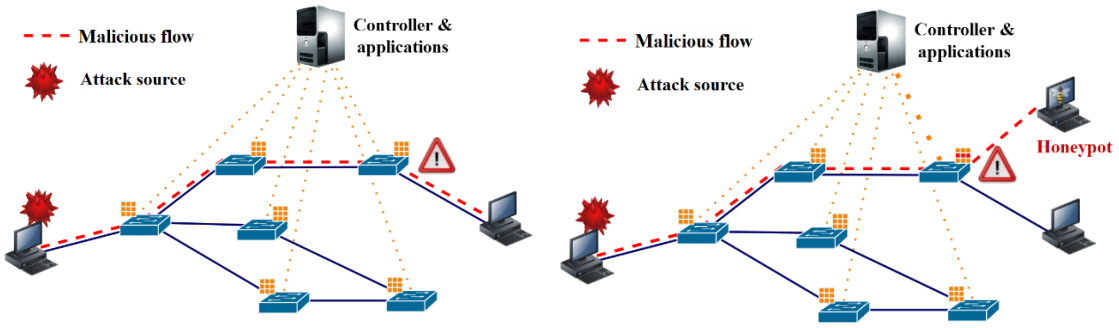


Figure 2: A single-domain SDN-based security solution (adapted from (Genge 2016))

The second technique (see Figure 3) targets multi-domain networks, with the goal of blocking the attack as close as possible to the entry point in the network.

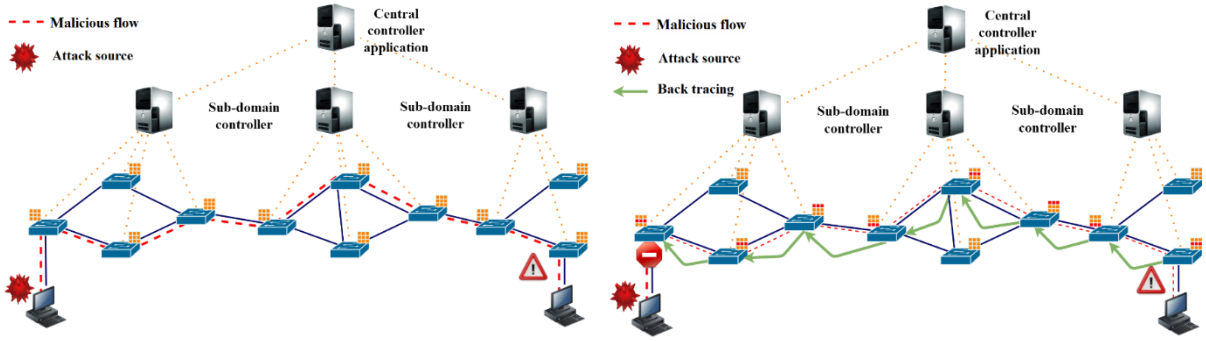


Figure 3: A multiple-domain SDN-based security solution (adapted from (Genge 2016))

For such a multi-domain network, each domain has its own OpenFlow controller, connected to a centralized security application. This application receives information from the SDN controllers, having access to a global perspective about the network – once an attack is detected, it will backtrack towards its origin, by recursively issuing queries about the related flows to identify the previously paired nodes, until the original network entrance point is found.

Network Function Virtualization and Distributed ICS

NFV is the result of the convergence between telecommunications infrastructures and infrastructure virtualization. As network applications and services scale and evolve (not only in sheer capacity requirements, but also in complexity), they imposed an added burden to the supporting telecommunications provider infrastructure, requiring the use of specific network management and traffic policies that cannot be provided by the network. In this perspective, NFV (Chiosi 2012) is a significant development as it enables the creation of flexible and on-demand network services through a service chain-based composition mechanism that uses network functions implemented in VNF (Virtualized Network Functions) components comprising

functionality such as NAT, IDS, Firewalls or other service modules, implemented as VM appliances.

The NFV vision attempts to decouple network capacity from functionality, by conceiving an end-to-end service as an entity that can be modeled and described by means of network function forwarding graphs (Figure 4) involving interconnected VNFs and endpoints (also known as service chaining.

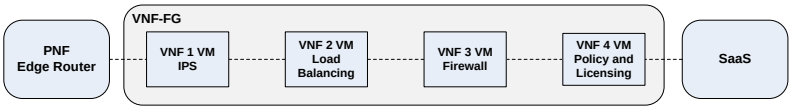


Figure 4: NFV Forwarding Graph example

This approach allows for creation of differentiated end-to-end services that can be provided by the (ordered) combination of elementary VNF or physical functions, chained together by a Forwarding Graph, which models the service flows (see Figure 5). Furthermore, VNF FGs can be nested to define complex functions. VNFs are implemented in software, being interconnected through the logical links that are part of a virtualized network overlay, which can be implemented using SDN.

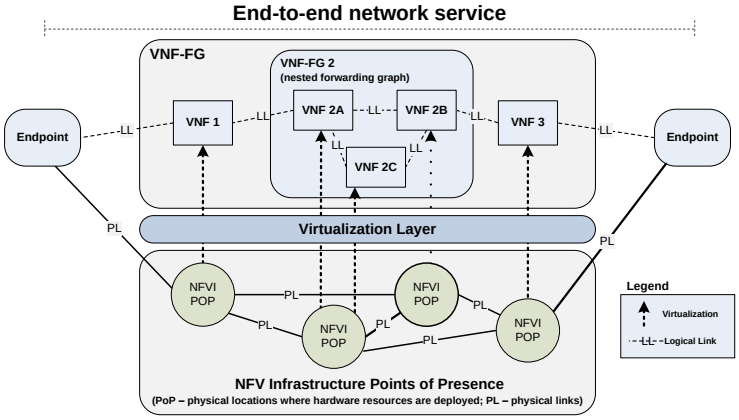


Figure 5: NFV end-to-end service with VNFs (adapted from (Ersue 2013))

Eventually, even Physical Network Functions (conventional network devices with close coupled software and hardware that perform network functions) can be involved in a Network Forwarding Graph service chain (the concept of service chain is not exclusive of NFV). A virtualization layer abstracts the physical resources (computing, storage, and networking) on top of which the VNFs are deployed and implemented, with the supporting NFV Infrastructure (NFVI) being spread across different physical locations, called Points of Presence (NFVI PoPs), as shown in Figure 5.

NFV as an enabler for a new generation of distributed IACS

Use cases such as Internet of Things (IoT), wire to water generation, micro generation, smart metering or smart water management constitute a new generation of distributed IACS that can only be supported with the help of a complex distributed software stack, potentially also requiring the involvement of third-parties, such as telecommunications and cloud operator infrastructures – for this reason, the introduction of Network Function Virtualization component appliances, distributed across geographically dispersed infrastructure PoPs, makes entire sense,.

As the IACS enters the customer premises, the NFV service abstraction model (services as composition of VNFs) provides an effective way to introduce support components along the service path – for instance, a data collection and analysis VNF can be added to the customer service chain (eventually within a virtual Business Gateway service abstraction) to provide data collection for smart metering scenarios. The same rationale applies for security purposes, as cyber-physical protection (for example, to implement bump-in-the-wire encryption) or security anomaly detection VNFs can be integrated within service chains, also using SDN to create flexible security monitoring and reaction capabilities. Moreover, Distributed IDS (DIDS) components may be consolidated in the form of VNFs optimally deployed in order to reduce service overhead and rationalize resources. For instance, the DIDS components might be deployed in the form of VNFs, either shared among several Business Gateway FGs or used exclusively by a service instance (Cruz 2015). Some manufacturers (RAD 2015) (ECI 2015) are starting to propose NFV products for ICS applications that implement this philosophy, NFV capabilities in access nodes for optical transport or packet switched networks, for hosting firewall, encryption or traffic monitoring VNFs

NFV is also an enabler for fog computing (or "edge computing") scenarios, allowing parts of the infrastructure to be deployed on the network edges, using virtualized platforms located between end user devices and the cloud data centers. This approach addresses the need to process large data streams in real time while working within the limits of available bandwidth, by placing some of transactions and resources at the edge of the cloud (in locations close to end users), thus improving the efficiency of the infrastructure by offloading processing tasks before passing it to the cloud. For these reasons, fog computing is becoming a cornerstone concept for distributed IACS architectures, providing a way to deal with the information volume generated by sensor streams in an efficient way.

The NFV paradigm is naturally compatible with fundamental premises for implementation of fog computing distributed topologies. As such, it is envisioned that distributed awareness and IACS cyber-security detection capabilities will take advantage of the NFV paradigm to support its underlying deployment model, departing from the conventional, self-contained model and moving towards an architecture capable of keeping up with the geographically dispersed nature

of IoT IACS. Also, the VNF deployment criteria may consider the availability of specific capabilities (such as raw processing capacity) in a specific NFVI POP – for instance, per-subscriber security event processing components may be hosted in a different NFVI POP from the one(s) hosting other VNFs for the DIDS service.

Real-time Hypervisors + SDN = towards a virtualized PLC

Born in the mainframe era, Virtual Machine Monitors (also called Hypervisors) have ultimately evolved towards being supported in open, Commercial Off-the-shelf (COTS) hardware platforms. Specifically, type-1 (bare metal) hypervisors have become popular in large-scale virtualization scenarios such as datacenters, bringing several benefits in terms of resource consolidation, business continuity, scalability, management and security.

But most type-1 hypervisors are optimized for ICT loads, being unsuitable for several ICS application use cases, mostly due to the overhead of the mediation and translation mechanisms abstracting the host hardware from the VM. This situation gradually began to change, as some operators started virtualizing hosts with services deployed on general-purpose OS, such as SCADA Master Stations (MS), Human-Machine Interfaces (HMI) or Historian Database servers (HDB), using conventional type-1 hypervisors. This was possible due to the development of hardware-assisted memory management and I/O mechanisms to implement robust resource affinity and reservation (such as VT-d and PCI SRV-IO (Garcia-Valls 2014) support), providing performance guarantees while avoiding the effect of resource overprovisioning.

Other ICS elements, such as process control devices can also potentially benefit from virtualization technologies. For instance, (Cahn 2013) proposed the virtualization of Intelligent Electronic Devices (IEDs) used to collect information from sensors and power equipment, with the purpose of optimizing the maintenance and cost overheads, while increasing reliability. The same rationale could be applied to Programmable Logic Controllers (PLC) devices, which constitute the focus of this section.

PLCs are pervasive devices in ICS, such as SCADA systems, being designed to control industrial processes. Contemporary PLCs are the outcome of an evolutionary process that started with the first generation of relay-based devices, progressively incorporating technologies such as microprocessors, microcontrollers and communications capabilities, ranging from serial point-to-point or bus topologies to Ethernet and TCP/IP. Despite modern PLCs often being embedded devices with commodity Instruction Set Architecture (ISA) System-on-Chip or CPUs (PowerPC, x86 or ARM), running Real-Time Operating Systems (RTOS), its virtualization was not deemed feasible until recently, due to the lack of specific hardware, software and infrastructure support.

Towards the virtual PLC

PLCs are designed for reduced and deterministic latency, operating under strict timing constrains that are dependent on factors such as the end-to-end and event response latencies across components on interconnected buses, or signal and message propagation delays. These requirements are incompatible with the use of several virtualization technologies, such as conventional type-1 hypervisors, due to overhead issues and the lack of support for real-time payloads.

However, recent developments such as the implementation of low-latency deterministic network connectivity for converged Ethernet and the availability of real-time hypervisors made it possible to virtualize components of the PLC architecture. The vPLC architecture hereby proposed (Figure 6) takes advantage of these capabilities, by decoupling the PLC execution environment from I/O modules – using an SDN-enabled Ethernet fabric to provide connectivity to the I/O subsystem. This architecture departs from the SoftPLC concept, as proposed by products such as (Codesys) or (ISaGRAF), by adopting an approach in line with (Intel 2013) and (IntervalZero 2011), with the added benefit of a convergent fabric scenario with SDN capabilities.

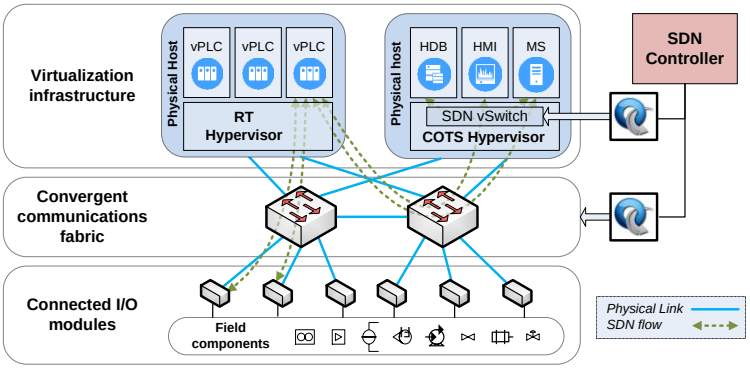


Figure 6: The vPLC architecture

In the vPLC, the PLC I/O bus is replaced by high-speed networking capabilities, with SDN allowing for the creation of flexible virtual channels on the I/O fabric, accommodating the connectivity flows between the vPLC instances and the I/O modules, such as sensor interfaces or motion controllers, providing traffic isolation. Moreover, such I/O modules can be built with reduced complexity, thanks to recent progress in terms of Field-Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuit (ASIC) technology. SDN reconfiguration is managed by means of an SDN controller, via a High-Availability (HA) server (not depicted in the figure), which interacts with its northbound interface. The HA server continuously monitors the SDN switch statistics and path reachability, triggering reconfiguration procedures in case of performance degradation or failure.

This decentralized model shares similarities with remote or distributed I/O PLC topologies, with networked I/O modules acting as extensions of the PLC rack. This goes in line with the Converged Plantwide Ethernet (CWpE) (Didier 2011) architecture, or even critical avionics systems, which replace legacy interconnects with Ethernet-based technologies, such as Avionics Full-Duplex Switched Ethernet (AFDX) (Fuchs 2012).

Advances in cut-through switching, together with Remote Direct Memory Access techniques (RDMA), particularly in converged Ethernet scenarios, have allowed for port-to-port latencies of the order of the hundredths of nanoseconds in 10G Ethernet switch fabrics and application latencies in the order of microseconds (Beck 2011). Additionally, resources such as Intel's Data Plane Development Kit (DPDK) (Zhang 2014) allow for the implementation of low latency, high-throughput packet processing mechanisms that bypass kernels, bringing the network stack into user space and enabling adapters to perform Direct Memory Access operations to application memory. This enables satisfying requirements for single-digit microsecond jitter and restricted determinism, allowing for bare-metal performance on commodity server hardware. On top of this, proposals such as the 802.1Qbv Time Sensitive Networking (IEEE) standard provide compliance with real-time requirements in the microsecond range on conventional Ethernet.

As for computing resources, there are two factors that must be considered. First, modern x86 or ARM processors have become capable of replacing microcontrollers in standalone PLC applications (Kean 2010), due to improvements in terms of raw performance, low latency I/O mechanisms or the availability of ISA extensions suitable for Digital Signal Processing tasks. Second, the availability of real-time static partitioning hypervisors, such as Jailhouse (Siemens), Xtratum (Crespo 2010), X-Hyp (X-HYP) or PikeOS (Baumann 2011) enables hosting RTOS guest VMs for real-time workloads. Some hypervisors, such as Xtratum and PikeOS, even replicate the ARINC 653 (Fuchs 2012) partitioning model for safety-critical avionics RTOS, with a Multiple Independent Levels of Security/Safety (MILS) (Alves-Foss 2006) architecture.

The benefits of this approach are manifold. The price tag for entry-level PLCs is comparable to a COTS server that can host several vPLC instances, being kept out of the factory floor or industrial environment. Distributed I/O on converged Ethernet also provides cost-effective performance and reliability benefits, as communications between different vPLC instances can take place across the convergent fabric or even locally, if co-located on the same host, with SDN allowing for flexible creation of communications channels, for differentiated requirements. Moreover, I/O modules – the components with highest failure rate in PLCs – can be easily and quickly replaced, in case of failure.

Particularly, the potential advantages of the vPLC in terms of reliability, safety and security are

considerable, as it can take advantage of datacenter-like redundant power, computing and communications resources. Other benefits are also envisioned, namely:

- Hypervisors allow for migration of virtualized ICS components, as well as instance cloning for pre-deployment tests;
- PLC watchdogs and system-level debugging and tracing mechanisms can be implemented at the hypervisor level, which is able to oversee and control the vPLC partition behavior;
- vPLCs benefit from partitioning isolation, with VMs being easy to restore in a fresh state in case of tampering or other malicious activity;
- SDN-managed isolated I/O paths ease the implementation of flexible, on demand, protection mechanisms at the I/O level (as shown in Section 3) also paving the way for the introduction of NFV components at the ICS level.

Overall, these benefits suggest that virtualizing a PLC could be feasible even for a single instance per device, using Industrial-grade Single Board Computers, instead of COTS servers.

Conclusion

This paper discussed the implications of the progressive introduction of virtualization technologies in ICS, with a special focus on security and reliability aspects. The virtualization of both network and computing virtualization was analyzed from an ICS-centric standpoint, covering recent developments as well as proposing new use cases and approaches to improve network and systems security.

Starting with an overview of network virtualization technologies such as SDN and NFV and their application within ICS and distributed IACS, the paper next addressed the issue of using hypervisor technologies for real-time workloads. In this latter perspective, a virtual PLC (vPLC) architecture was discussed, which transcends the simple virtualization of the PLC device, constituting an integrated approach where the device merges with the infrastructure, in a seamless way. The vPLC takes advantage of network and computing virtualization technologies to propose a converged approach for plan-wide consolidation of the ICS infrastructure, with performance, cost and security benefits. This proposal is presently under development by a team that includes the authors of this paper.