



City Research Online

City, University of London Institutional Repository

Citation: Petroulakis, N. E. (2019). A pattern-based framework for the design of secure and dependable SDN/NFV-enabled networks. (Unpublished Doctoral thesis, City, University of London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/24065/>

Link to published version:

Copyright: City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

Reuse: Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

A Pattern-Based Framework for the Design of Secure and Dependable SDN/NFV-Enabled Networks




Nikolaos E. Petroulakis

Department of Computer Science
School of Mathematics, Computer Science and Engineering
City, University of London

Supervisor Prof. George Spanoudakis

Thesis for the degree of Doctor of Philosophy

To my beloved children 

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text.

Nikolaos E. Petroulakis
November 2019

Acknowledgements

This PhD is the result of a hard but satisfying work with many challenging but valuable experiences that it would not have been possible without the support and guidance of many people to who I would like to express my sincere appreciation.

First of all, I would like to express my gratitude to my supervisor Prof. George Spanoudakis for giving me the opportunity to work with him on pursuing this PhD. His expertise and his consistent guidance together with his vision helped me to complete this research. Under his supervision, I had the opportunity to investigate and acquire further knowledge in new research topics and directions. Without his support and constant feedback, this PhD would not have been realised.

Besides my advisor, I would like to thank my second supervisor, Dr. Christos Kloukinas, for all his insightful comments and his valuable support during the whole period of my research.

I would also like to express my sincere appreciation to my external supervisor Dr. Ioannis Askoxylakis for his mentoring and guidance from the beginning of this PhD until the end of my research. Sharing with me his knowledge and technical know how were precious for the accomplishment of this work.

My sincere thanks to my closest friends for their unconditional support and patience. After the completion of this PhD, we will have more spare time to spend together as this was not so often possible during the writing period.

My deepest gratitude to my family. More specifically, I would like to thank my parents who have always been there for me. I would also like to express my thanks to my sister for standing by me not only as a sister but also as a friend.

Last but not least, I would like to thank my wife [REDACTED], my inspiration, for her practical and emotional support and for her patience and humour. She has always been a devoted co-traveller in this journey.

Abstract

As the world becomes an interconnected network where objects and humans interact, cyber and physical networks appear to play an important role in smart ecosystems due to their increasing use on critical infrastructure and smart cities. Software Defined Networking (SDN) and Network Function Virtualisation (NFV) are a promising combination for programmable connectivity, rapid service provisioning and service chaining as they offer the necessary end-to-end optimisations. However, with the actual exponential growth of connected devices, future networks, such as SDN and NFV, require open architectures, facilitated by standards and a strong ecosystem. In this thesis, a model-based approach is proposed to support the design and verification of secure and dependable SDN/NFV-enabled networks. The model is based on the development of a pattern-based approach to design executable patterns as solutions for reusable designs and interactions of objects, encoded in a rule-based reasoning system, able to guarantee security and dependability (S&D) properties in SDN/NFV-enabled networks. To execute S&D patterns, a pattern-based framework is implemented for the insertion of patterns at design and at runtime level. The developed pattern framework highlights also the benefit of leveraging the flexibility of SDN/NFV-enabled networks to deploy enhanced reactive security mechanisms for the protection of the industrial network via the use of service function chaining (SFC). To prove the importance of this approach and the functionality of the pattern framework, different pattern instances are implemented to guarantee S&D in network infrastructures. The developed design patterns are able to design network topologies, guarantee network properties and offer security service provisioning and chaining. Finally, in order to evaluate the developed patterns in the pattern framework, three different use cases are described, where a number of usage scenarios are deployed and evaluated experimentally.

Table of contents

List of figures	xv
List of tables	xix
Notations	xxi
1 Introduction	1
1.1 Overview	1
1.2 Motivation and Research Challenges	7
1.3 Research Aims and Objectives	12
1.4 Research Assumption and Hypothesis	13
1.5 Research Contributions	13
1.5.1 Pattern Schema	14
1.5.2 Pattern Instances	14
1.5.3 Pattern Framework	15
1.5.4 Reactive Security Leveraging SFC	15
1.5.5 Network Simulator	16
1.5.6 Evaluation of the Pattern Framework	16
1.6 Publications	16
1.7 Outline of Thesis	20
2 Background and Literature Review	23
2.1 Overview	23
2.2 Security and Dependability Attributes	23
2.3 Security and Dependability Threats	25
2.3.1 Generic Threats	26
2.3.2 Radio Access Threats	28
2.3.3 SDN/NFV Threats	29
2.4 Security and Dependability Countermeasures	30

2.4.1	Generic Countermeasures	31
2.4.2	Radio Access Countermeasures	35
2.4.3	SDN/NFV Countermeasures	36
2.5	Service Provisioning and Chaining	41
2.5.1	Security Service Functions	42
2.5.2	Service Function Chaining	44
2.5.3	SFC Related Research	45
2.6	Design Patterns	46
2.7	Open Issues Addressed by this Work	48
2.8	Summary	49
3	Definition of the Pattern Schema and Language	51
3.1	Overview	51
3.2	Pattern Definition	51
3.3	Pattern Topology	53
3.4	Pattern Requirements and Properties of Compositions	54
3.4.1	Composition Properties	54
3.4.2	Functional Requirements	56
3.4.3	Non-Functional Requirements	56
3.5	Pattern Specification	57
3.6	Pattern Composition and Reasoning	58
3.6.1	Forward Chaining	58
3.6.2	Backward Chaining	60
3.7	Pattern Components	60
3.7.1	Network Graphs	60
3.7.2	Network Components	64
3.7.3	SFC Terms and Components	64
3.7.4	Data Flows and Policies	66
3.8	Pattern Language	69
3.9	Summary	72
4	Secure and Dependable Design Patterns	75
4.1	Overview	75
4.2	Topology Patterns	75
4.2.1	Physical Topology Patterns	75
4.2.2	Logical Topology Patterns	86
4.3	Path Discovery Pattern	90

4.4	Reliability Patterns	93
4.4.1	Reliability in Compositions	93
4.4.2	Serial and Parallel Reliability Pattern	96
4.4.3	Serial-Parallel Reliability Pattern	99
4.5	Fault Tolerance, Detection and Restoration Patterns	101
4.5.1	Fault Tolerance Pattern	102
4.5.2	Fault Detection and Restoration Pattern	103
4.6	Security Patterns	105
4.6.1	Link Encryption Pattern	105
4.6.2	End-to-End Encryption Pattern	109
4.7	Service Function Chaining Patterns	113
4.7.1	VNF Instantiation Pattern	114
4.7.2	SFC Path Finding Pattern	119
4.8	Summary	122
5	Implementation of the Pattern Framework	123
5.1	Overview	123
5.2	Pattern Framework	124
5.2.1	Pattern Schema and Requirements	124
5.2.2	Pattern Engine in the Application Layer	126
5.2.3	Pattern Engine in the SDN Controller	127
5.3	SFC Reactive Security	130
5.3.1	Virtual Network Functions	132
5.3.2	SFC Manager	135
5.3.3	SFC GUI	136
5.3.4	SFC in the NFV MANO	137
5.3.5	Dynamic SFC instantiation	138
5.4	Network Simulator	139
5.5	Testing and Evaluation Environment	143
5.5.1	Emulated Infrastructure	143
5.5.2	Virtual Infrastructure	143
5.5.3	Software Setup	145
5.6	Summary	145
6	Evaluation of Design Patterns in the Pattern Framework	147
6.1	Overview	147
6.2	Design Network Topologies	147

6.2.1	End-to-End Connected Network Topologies	148
6.2.2	Maximum Coverage and Redundancy in Network Topologies . . .	150
6.2.3	Scalable Network Design	151
6.3	Design and Verification of S&D Networks	152
6.3.1	Reliable Network Designs	153
6.3.2	Fault Tolerance, Detection and Restoration in SDN	158
6.3.3	Secure Transmission in SDN-enabled Networks	162
6.4	Design Secure Industrial Networks Leveraging Service Function Chaining .	167
6.4.1	Virtual Functions Instantiation based on SFC Requests	169
6.4.2	SFC Path Finding and Traffic Classification	172
6.5	Summary	178
7	Conclusion	181
7.1	Overview	181
7.2	Summary of the Research	181
7.3	Contributions	182
7.4	Limitations	184
7.5	Future Work	186
	References	189
	Appendix A Developed Java Classes	203
	Appendix B Service Function Chaining Files	231

List of figures

1.1	Evolution of Industry	2
1.2	SDN Architecture	4
1.3	SDN Mapping in NFV architecture	7
2.1	Relation Between non-Functional and Functional Attributes	26
2.2	SDN Threats	29
2.3	IPsec Framework	34
3.1	Stepwise Decomposition	56
3.2	Forward Chaining Schema	59
3.3	Class Diagram of the Network Components in Graphs	62
3.4	Class Diagram of the Network Components in Service Function Chaining	66
3.5	Class Diagram of Packet Components	67
3.6	OpenFlow Rule Example	68
3.7	Class Diagram of Network Components	71
4.1	Physical Network Topologies	76
4.2	Network Pattern Compositions (i) Line (ii) Tree (iii) Mesh (v) Full-Mesh	77
4.3	Distance and Position of Nodes	79
4.4	Line Composition	80
4.5	Line Decomposition	81
4.6	Stepwise Decomposition	85
4.7	Basic Logical Topologies: (a) Sequence (b) Parallel-split-join (c) Multi-choice-join (d) Exclusive-choice-join	87
4.8	Class Diagram of the Workflow Patterns	88
4.9	Path Decomposition	91
4.10	Non-reducible Split of Fully Mesh Pattern	95
4.11	Activity Diagram of Reliability Pattern	97
4.12	Reliability Serial-Parallel Pattern Decomposition	100

4.13	Fault Tolerance Pattern	102
4.14	Fault Detection and Restoration Pattern	104
4.15	Activity Diagram of the E2E Security Procedure	111
4.16	Service Functions (i) on Same Service Node, (ii) on Same Switch (iii) on Same Domain (iv) on Different Domains	115
4.17	VNF Instantiation based on SFC Request	117
4.18	Conceptual, Simple and Extensive Service Function Chaining Example . .	120
	(a) Conceptual service function chain architecture	120
	(b) Simple Service Function Chain	120
	(c) Extensive Service Function Chain	120
5.1	Architecture of the Pattern Framework	124
5.2	Drools Patterns	126
5.3	Network Design within the Pattern Framework	127
5.4	Network Verification and Adaptation within the Pattern Framework	129
5.5	OpenDaylight SDN Pattern Framework GUI	130
5.6	Security Functions in NFV Architecture	132
5.7	Reactive Security SFC GUI	137
5.8	Expanded, NFV-O Managed and ETSI-aligned, Framework Architecture . .	138
5.9	SFC Requests in the Pattern Framework	139
5.10	Network Topology Json Example	141
5.11	Implemented Network Simulator	142
5.12	Network Simulator and Pattern Framework Interaction	142
5.13	Hypervisors	144
6.1	Evaluated Use Case and Scenarios	148
6.2	End to End Connected Networks	149
6.3	Line Pattern Output on Network Simulator	150
6.4	Mesh Networking in Rural and Urban Environments	150
6.5	Mesh Pattern Output on Network Simulator	152
6.6	Scalable Network Designs based on Tree Topologies	153
6.7	Network Simulator Outputs on Tree Pattern	154
6.8	Reliable Network Designs	155
6.9	Design Phases of a Sensor Network with Reliability (a) 96% (b) 98% (c) 99.9%	156
6.10	OpenDaylight SDN Infrastructure Topology	158
6.11	Network Simulator Outputs on Reliability Patterns	159
	(a) Serial and Parallel Output	159

(b) Serial-Parallel Output	159
6.12 Fault Tolerance in SDN	160
6.13 Experimental Results of Fault Detection and Restoration Pattern	161
6.14 End to End and Link Encryption Use Case	162
6.15 Link to Link Encryption in Network Topologies	164
(a) Network Topology	164
(b) Network Topology Security	164
6.16 End to End Encryption in Network Topologies	166
6.17 Evaluation of the End-to-End Security Pattern in the Network Simulator (green: secure, red: insecure, orange: undefined)	168
(a) Partially Secure Traffic from n1 to n2	168
(b) Partially Secure Traffic from n1 to n3	168
(c) Multi-hop Partially Secure Traffic from n2 to n3	168
(d) Insecure Traffic from n2 to n3	168
(e) Partially Secure Traffic from n2 to n4	168
(f) Fully Secure Traffic from n1 to n4	168
6.18 Reactive Security - Implemented Service Chains	169
6.19 SFC - Per Traffic Type Classification Example	170
(a) Intra Domain	170
(b) Inter Domain	170
6.20 VNF Instantiation in OpenStack	172
6.21 VNF Instantiation in Proxmox	173
6.22 Service Functions Imported in the ODL	173
6.23 Service Node Imported in the ODL	174
6.24 Service Function Chains Imported in the ODL	174
6.25 Service Function Forwarders Imported in the ODL	175
6.26 Access Control Lists Imported in the ODL	175
6.27 OpenFlow Rules Inside the Forwarders and the Classifiers	176
6.28 Demo Shell Script	176
6.29 Reactive Security - Demo	177
6.30 Real-time Traffic Flows on the Controller's GUI	178
(a) Legitimate traffic	178
(b) Legitimate traffic classified by DPI	178
(c) SCADA traffic	178
(d) SCADA traffic classified by DPI	178
(e) Malicious traffic	178

(f)	Malicious traffic classified by DPI	178
-----	---	-----

List of tables

2.1	SDN Fault Tolerance Mechanisms	40
2.2	Examples of Network Security Service Functions	43
2.3	Security Properties and Functions	44
3.1	Comparison Between Forward and Backward Chaining	61
3.2	Pattern Language Semantics	72
4.1	Composition Metrics	89
4.2	Reliability in Physical Network Topology Patterns	96
5.1	Abstract SFC Component Structure	136
6.1	Experimental Results of the Line Pattern Execution	149
6.2	Experimental Results of the Mesh Pattern Execution	151
6.3	Experimental Results of the Tree Pattern Execution	151
6.4	Experimental Results of the Serial and Parallel Pattern Execution	157
6.5	Experimental Results of the Serial-Parallel Pattern Execution	157
6.6	Experimental Results of Fault Tolerance Pattern Flow Configurations	160
6.7	Link Encryption on a Network Topology	163
6.8	End to End Security on a Network Topology	166
6.9	Experimental Results of the SFC Reactive Security	177

Notations

Property Notations

$C = \{c_1, c_2, \dots\}$	Set of components c
$A = \{a_1, a_2, \dots\}$	Set of activities a
$\mathcal{P}(c)$	Property of component c
$\mathcal{R}(\mathcal{P}(c))$	Requirement of property \mathcal{P}
$P = \{p_1, p_2, \dots\}$	Set of probability of components
$R = \{r_1, r_2, \dots\}$	Set of reliability of atomic components

Graph Notations

$G = (V, E)$	Graph with V vertices and E edges
$N = \{n_1, n_2, \dots\}$	Set of physical nodes n
$L = \{l_1, l_2, \dots\}$	Set of physical links $l_{n,n'}$
$G = (N, L)$	Physical network graph
$P = \{p_1, p_2, \dots\}$	Set of network paths $p = \{n_1, n_2, \dots\}$
$\bar{N} = \{\bar{n}_1, \bar{n}_2, \dots\}$	Set of virtual nodes \bar{n}
$\bar{L} = \{\bar{l}_1, \bar{l}_2, \dots\}$	Set of virtual links $\bar{l} = l(\bar{n}', \bar{n})$
$\bar{G} = (\bar{N}, \bar{L})$	Virtual network graph
$N^d = \{n_1^d, n_2^d, \dots\}$	Set of demanded nodes n^d
$L^d = \{\bar{l}_1^d, \bar{l}_2^d, \dots\}$	Set of demanded virtual links l^d
$G^d = (N^d, L^d)$	Network graph demand
$d(n, n')$	Distance between two nodes
$r(n)$	Range capability of node
$n(x, y)$	Position of a node
$deg(n)$	Degree of node n
$weight(l)$	Weight of a link l
$G = (N, L, W)$	Weighted network graph
$G = (N, L, F)$	Filter network graph

SFC Notations

$F = \{f_1, f_2, \dots\}$	Set of virtual network functions f
f_i^j	Function type i in service node j
$f_{i,u}$	Function f_i instantiated at most u times
$S = \{s_1, s_2, \dots\}$	Set of service function chains $s = \{f_1, f_2, \dots\}$
s^d	Service function chain demand

Resource Notations

CP	Computation power capacity
M	Memory capacity
T	Storage capacity
E	Energy capacity
D	Delay capacity
BW	Bandwidth capacity
$R_n = \{CP_n, M_n, T_n, E_n\}$	Node n resource capacity
$R_l = \{D_l, BW_l\}$	Link l resource capacity
R_n^d	Node n resource capacity demand d
R_l^d	Link l resource capacity demand d
$R_f = \{CP_f, M_f, T_f, E_f\}$	Function resource capacity
R_f^d	Function resource capacity demand

Data and Flow Notations

p	Data packet
p_{pld}	Data packet payload
p_{hdr}	Data packet header
k	Encryption key
$K = \{k_1, k_2, \dots\}$	Set of encryption keys
c	Cipher packet
E_k	Encryption function where E_k^S the symmetric and E_k^A the asymmetric
D_k	Decryption function where D_k^S the symmetric and D_k^A the asymmetric
f	where f_a is the ACL and f_o the Openflow Rule
$F = \{f_1, f_2, \dots\}$	Set of data flow rules

Chapter 1

Introduction

1.1 Overview

In an interconnected cyber-physical world, people, devices and infrastructures interact, establishing a smart environment in which the exchange of data and decisions is continuous. With the anticipated exponential growth of connected devices, future networks require open architectures, facilitated by standards and a strong ecosystem. Such devices need a simple interface to the connected network to enable the kind of communication service characterised with the required security and dependability guarantees. In response, the network should grant the network resources and program the intermediate networking devices based on device profile and privileges. A similar requirement comes also from business applications where application itself asks for particular network end-to-end guarantees based on users and tenants requests.

The fifth generation of networking (5G) aims to provide the next generation of mobile networks to fulfill the increasing demand in the business contexts of *2020 and beyond* including service management and orchestration [1, 2]. Software Defined Networking (SDN) and Network Function Virtualisation (NFV), important parts of 5G networking [3], provide promising combination leading to programmable connectivity, rapid service provisioning and service chaining and can thus help to decrease the capital and operational expenditure costs (CAPEX/OPEX) in the control network infrastructure. Especially, with the fast growth of SDN together with NFV and their integration with existing network architectures, the design of networks enters in a new era. SDN and NFV in the *Industry 4.0* arrive as new concepts to promote the computerisation of the manufacturing part of the network. Industry 4.0 [4] is the current trend for data exchange and automation in manufacturing technologies that can help in communicating essential technologies such as the Internet of Things (IoT), communication machine-to-machine (M2M) and Cyber-Physical Systems (CPS) [5], cloud

computing and cognitive computing. The evolution of the industry can be found in Figure 1.1. Furthermore, by appropriately leveraging the flexibility of SDN/NFV-enabled networks in the context of the adopted security mechanisms, industrial infrastructures can not only match but also improve their security posture compared to the existing legacy networking environments [6]. However, industrial networks typically come with strict performance, security, and reliability requirements [7]. Nevertheless, SDN and NFV expand the attack surface of the communication infrastructure, necessitating the introduction of additional security mechanisms.

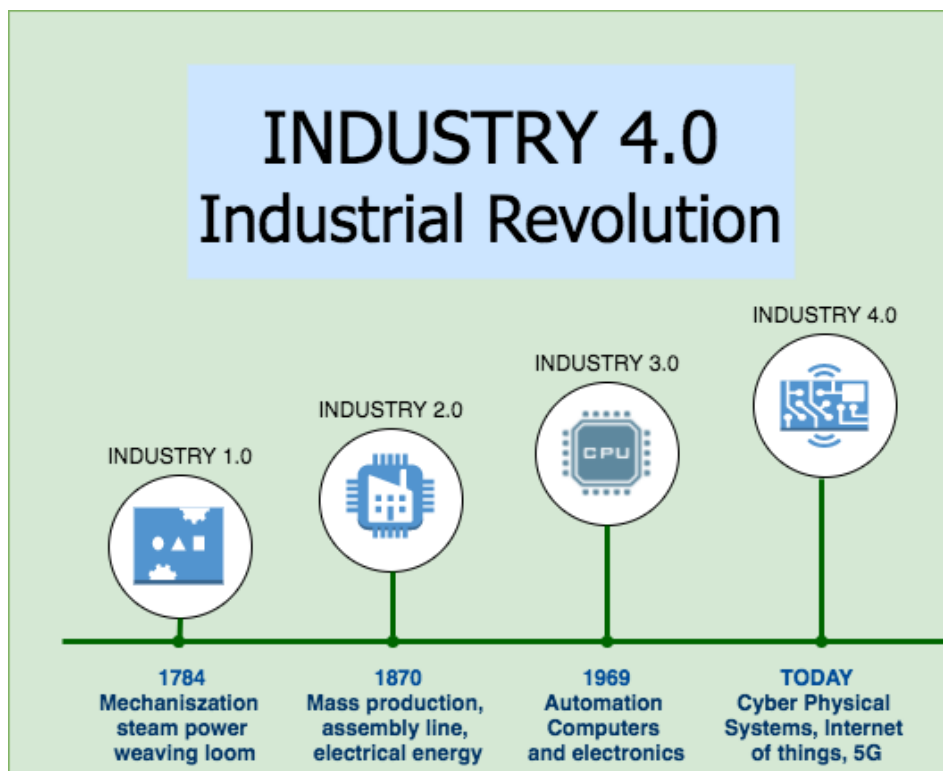


Fig. 1.1 Evolution of Industry

The key concept of SDN is the decoupling of control and packet forwarding functionality of the network. In legacy networks, both data and control functions are performed by the forwarding devices of the network. In SDN, the separation has two specific functionality planes: the control plane and the data plane. The separation of these two planes in SDNs has two significant advantages. The first advantage includes the reduction of the complexity in the configuration and alteration of the control functions of the network. This can be done since the forwarding devices of the network, which tend to have proprietary implementations, are no longer responsible for executing a set of functions. The second advantage enables the

implementation of more consistent control policies through fewer and uniformly accessible controllers.

In traditional networks, administrators manually control the application access and configure the network accordingly. This process demands a lot of engineering and operations effort. In an SDN environment, the access of the applications to network resources is simplified through the enabled network programmability. SDN has been envisioned as a promising technology to reduce complexity and increase the flexibility of network configuration and management. It allows network programmability, allowing network control to be decoupled from the forwarding plane and the forwarding plane to be directly programmable by the control plane. In SDN networks, the control plane, handling management operations, is logically centralised and physically decoupled from data plane, responsible for data forwarding operations, thus enabling high network configurability and programmability. The decoupling of the control plane from data plane provides SDN deployments the ability to easily add new powerful network set of functions or protocols. Although the control plane can be distributed [8], its logical centralised architecture offers one of the most important advantages of it such as the simplification of network configuration, operation and management. The required high level network policies and functionalities (e.g. routing, traffic engineering, access control) are translated to low level network commands. Based on this translation, the controller can insert appropriate flow rules to the programmable switching elements. The described network programmability is expected to alleviate the burden of the data onslaught from Internet of Things (IoT) deployments. This can be done primarily via centralised Network Resources Optimisation (NRO) and optimally exploiting the underutilised network resources [9]. As presented in [10], an SDN controller is employed to decouple the control plane, which runs on top of the Wireless Sensor Networks (WSN), from the data plane, which is still implemented in the sensor nodes. This appears to play an very important role especially in smart cities and on critical infrastructure and is considered to be one of the key elements of the 21st century. More specifically, the main target of SDN is to drive the reconfiguration of these capabilities through specifications embedded especially in critical infrastructures. In this view, SDN becomes another component in the IoT implementation stack that, like other components, can be dynamically configured.

According to the Open Networking Foundation (ONF) [11], the SDN architecture consists of different layers: the infrastructure layer, the control layer and the application layer including also the respective intermediate interfaces, as can be seen in the Figure 1.2.

- The **Infrastructure Layer**, also referred as data plane, is responsible for the data forwarding functionality of the network. The functionality of this plane is guaranteed through a set of physical network devices (network elements). This layer includes

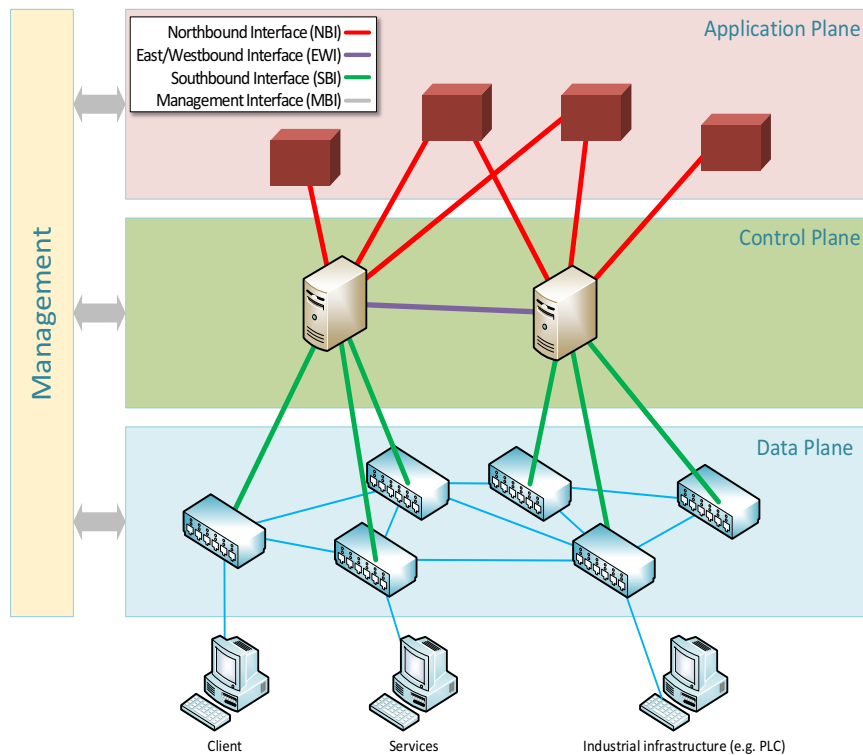


Fig. 1.2 SDN Architecture

SDN network deployments containing physical instances of the network such as the switching devices, the communication medium etc. (wired or wireless).

- The **Control Layer**, also referred as control plane, is responsible for the control functionality of the network. The functionality of this plane is realised through a set of devices and controllers that are able to facilitate the creation and destruction of network flows and paths. This plane includes both the hardware (e.g. controllers, interfaces) and software used to realise SDN control (e.g. interfaces for the controller communication), along with system configuration and control data. Different controllers are responsible to control the SDN network over disjoint subsets of forwarding devices. The different parts of the SDN can be handled by the controller through these forwarding devices. The control layer is focused on the control functionality of the network containing base network service functions such as topology/flow/connection manager and flow statistics. An SDN controller is able to (i) proactively plan redundant paths for critical services and (ii) react in real-time in case of emergency. There is a number of open source and commercial SDN controllers such as OpenDaylight (ODL) [12], Open Network Operating System (ONOS) [13], POS, NOS, Ryu, Floodlight, Onix, Brocade, Juniper [14].

- The **Application Plane** is responsible for generic network management auditing, and reporting functionalities (e.g., SDN management, monitoring and security). All the applications that are used to implement network functionalities such as network virtualisation, network monitoring, intrusion detection and flow balancing are realised on this layer. Furthermore, the application layer includes SDN applications, SDN/network management and security/dependability management.

There are different application program interfaces (APIs) which are used for the communication between the different layers of SDN architecture.

- The **Southbound Interface (SBI)** is used to connect the control layer with the infrastructure layer. The SBI is capable of supporting multiple protocols, e.g., OpenFlow [15], BGP [16], PCEP [17] or NetCONF [18]. This interface enables the communication between these forwarding devices and the controllers of the network. The OpenFlow standard, managed and promoted by the ONF, is one of the most popular SBI SDN standards. One of the main functionalities of the SBI is to support (i) Initial discovery of device capabilities and physical interconnections, (ii) runtime configuration and (iii) synchronous and asynchronous monitoring.
- The **Northbound Interface (NBI)** is available for the application layer to interact with the control layer. The NBI is implemented by the controllers of the SDN and is used to facilitate the communication between controllers and the network management applications. Representative examples of such APIs are FML, Procera, Frenetic, Maple and RESTful. In this direction, ODL and ONOS are the most prominent open source SDN Controllers which expose open NBI APIs, to be used by SDN applications. In addition, platform-oriented services and other extensions can also be inserted into the controller platform for enhanced SDN functionality.
- The **East/West Bound Interface (EWBI)** is implemented by the different controllers of the SDN and is used to facilitate communications between them. Representative examples of such APIs are ALTO and Hyperflow [19, 20].

On the other hand, NFV's main vision is to address networking open issues by utilising standard IT virtualisation technology. Furthermore, it aims to transform traditional network operations, as software can easily be moved to, or instantiated in, various locations (e.g. data centres, network nodes, end-user premises) without the need to use new equipment. In this manner, a large number of network services can be easily deployed on standard servers in a virtualised manner leveraging a number of important business and operational benefits.

In this direction, Open Platform for NFV (OPNFV¹) is an open source project focusing on accelerating the evolution of NFV. Moreover, OpenMANO² proposes an open source platform for NFV Management and Orchestration (MANO).

According to the European Telecommunications Standards Institute (ETSI)³ the NFV architecture consists of different components and layers as described below:

- **Operations Support Systems and Business Support Systems (OSS/BSS)** is responsible for network management and service provision.
- **Virtual Network Functions (VNF)** can be run as software instances on top of an abstracted infrastructure for a more dynamic and cost-effective network sharing. VNFs deliver software-only entities (e.g. vRouter, Firewall, vHoneypot etc.) to be installed and run, storage and switching platforms residing in the local control centre.
- **NFV Infrastructure (NFVI)** can provide the separation of hardware and software components that build up the environment in which related VNFs are deployed, managed and executed agnostically over hardware platform.
- **NFV Orchestrator (NFVO)** are responsible for on-boarding of new network services (NS) and VNF packages, NS lifecycle management, global resource management, validation and authorisation of NFVI resource requests.
- **Virtualised Network Function Manager (VNFM)** oversees lifecycle management of VNF instances, coordination and adaptation role for configuration and event reporting between NFVI and E/NMS.
- **Virtualised Infrastructure Manager (VIM)** controls and manages the NFVI compute, storage and network resources and corresponding virtualisation. It contains physical hardware resources (computing, storage and network) and the virtualisation layer, all managed by the (candidate component e.g. OpenStack).

NFV technology applies to all layers of SDN infrastructure including *control plane* functions as well as *data plane* packet processing and has the potential to revolutionise both fixed and mobile network infrastructures. The fit of SDN layers in the NFV architecture can provide a clear view on how both concepts may be highly complementary as described in [21] and depicted in Figure 1.3. Although the location and the functionality of each layer of SDN architecture may be placed in more than one location in NFV architecture, this can be

¹<https://www.opnfv.org>

²<https://github.com/nfvlabs/openmano>

³<http://www.etsi.org>

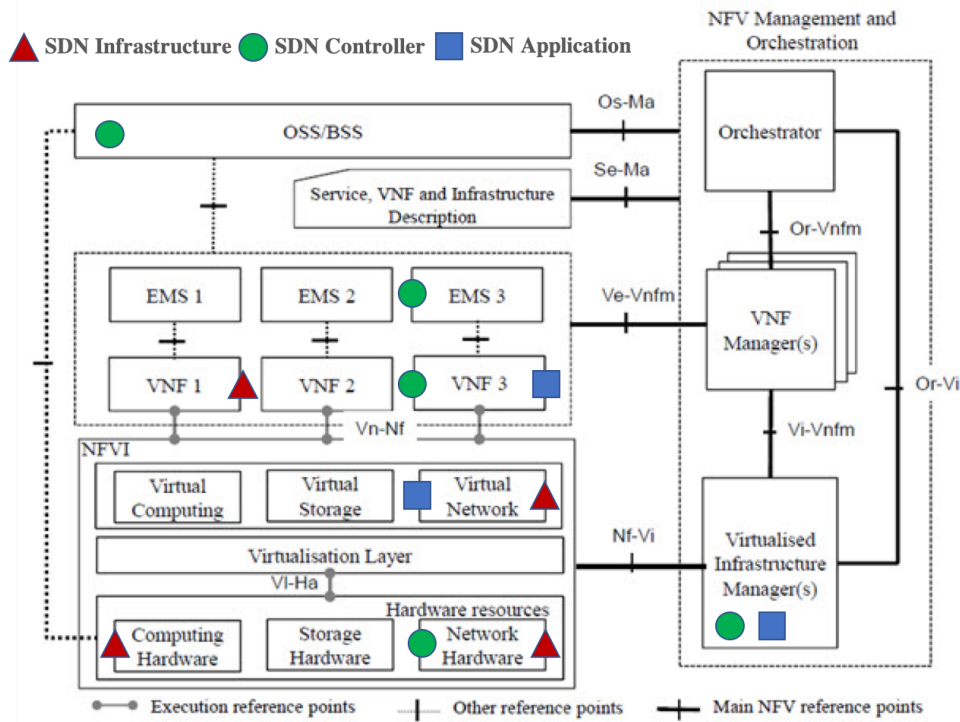


Fig. 1.3 SDN Mapping in NFV architecture

valid due to the NFV different view of the virtualisation concept. Therefore, the integration of network orchestration with NFV and SDN can allow network operators to deploy their services automatically in the cloud environment. Moreover, NFV is mostly involved with abstraction of service function aiming to bring flexibility and cost reduction. On the other hand, SDN focuses with the networking abstraction promises to bring unified programmable control and open interfaces. Before network orchestration, the same has been achieved via lengthy and sometimes error prone manual processes. Therefore, from a functional point of view (non-automated) orchestration has always been the part of service provider's solutions.

1.2 Motivation and Research Challenges

Critical infrastructures such as wind parks, water and gas distribution networks, power grids, etc. constitute complex infrastructures [6], which reliable, safe and secure operation is of paramount importance for daily activities at both national and international level. Military applications, factories and industries, bridges, medical and health and environmental applications are some examples where security threats should be taken into consideration. Global networks such as SDN and NFV together with the IoT create an enormous potential

for new generations of applications. Synergies arising through the convergence of consumer, business and industrial Internet, can create open networks to connect people, data, and *things*. A series of innovations across the landscape have converged to make network products, platforms and devices technically and economically feasible. However, despite these advancements the realisation of the potential risks requires overcoming significant business and technical hurdles. All the above risks should be considered during the design of network architectures due to their increasing role in the implementation of IoT and SDN involving integrated cyber and physical components and devices.

Today, organisations are targeted by cyber adversaries with a range of goals from political activism and sabotage to intellectual-property theft and financial gain. In the era of advanced cyber threats, including mobile threats, greater situational awareness is essential to detect and mitigate sophisticated attacks effectively. Organisations need to obtain the latest data on threats, relate it to real-time insights into their dynamic IT and business environments, determine what is relevant, make risk decisions, and take defensive action. Intelligence gathering and analysis have become essential of information security capabilities. However, most organisations while extending their network infrastructure by exploiting the use of SDN/NFV and/or moving towards IoT, have not been designed with this objective in mind. Security and Dependability (S&D) on SDN/NFV-enabled networks are both critical issues for public security and safety. Cyber-attacks on remote monitored and managed IoT add new security and safety threats, and consequently energy and economic losses. Integrating IoT and SDN can increase the efficiency of the network by responding to changes or events detected at IoT application layer, with network reconfiguration. One of the challenges of future networks is to develop SDN/NFV capabilities tailored to the IoT. Lately, SDN extensions have been proposed to directly incorporate IoT devices in SDN networks. Applications running on top of SDN architecture need to be resource and network-aware, in order to take full advantage of underlying network programmability and to become more agile. Moreover, SDN requires a careful investigation of new S&D capabilities and risks, which have not been relevant in legacy systems. The design and verification methods for developing secure and dependable system networks is necessary and should be considered at the design level to guarantee security and mitigate safety threats, on remote monitored and managed networks. However, the design of such networks effectively encounters difficulties that stem from the highly distributed and heterogeneous nature of SDN and the extent of intelligence, dependability and security that they need to demonstrate during their operation.

In addition to the above, paradigms of SDN/NFV security techniques and in consideration of the criticality of industrial networks, the following **principles** outline design considerations towards a secure and dependable network implementation [22]:

- **Dynamic device association** will ensure network function continuity and minimise downtime and data loss. Network elements should be able to dynamically associate to a backup controller in case of compromise or inaccessibility of the primary controller.
- **Replication** is an essential function for achieving dependability of a system or an entire infrastructure. Replicated multiple instances of the controller as well as application replication will ensure failure tolerance and minimise downtime whether the threat is an attack or a physical disaster.
- **Self-healing mechanisms** whether proactive or reactive in combination with proper maintenance can provide diversity in the recovery process, thus enhanced protection towards attacks by exploiting targeted vulnerabilities.
- **Diversity** of controller types (e.g. different operating systems, hardware) can improve dependability as it is unlikely that a variety of software and hardware combinations will have the same vulnerabilities.

SDN/NFV-enabled network infrastructures face a number of security, privacy and dependability issues which need to be resolved in order to step forward into the interconnected world and the Future Internet. These issues can be categorised as: i) Data Plane issues (e.g. data forging in switching devices), ii) Control Plane issues (e.g. controllers/interfaces exploitation) and iii) Application Plane issues (e.g. network monitoring, traffic engineering) [23]. Furthermore, the discrepancy between technologies and the attempt to interconnect them have brought new security challenges and gaps which need to be filled. By appropriately leveraging the flexibility of SDN/NFV-enabled networks in the context of the adopted security mechanisms, industrial infrastructures can not only match but also improve their security posture compared to the existing, traditional networking environments [6]. Nevertheless, the flexibility of SDN networks guarantees that they can also help provide better security for industrial networks. Due to the controller's global view of the network and the ability to reprogram the data plane at real-time, SDN allows not only to revisit old security concepts (e.g. firewalls) but introduce new techniques (e.g. steering suspicious traffic to Supervisory Control and Data Acquisition (SCADA) Honeypots, adopting moving target defence and other reactive techniques). The deployment of these enhanced security concepts is in line with the enhanced protection requirements of critical infrastructures, given that the old paradigm of perimeter defences and trusted internal networks is obsolete, as the attacks demonstrated in [24]. Thus, enhanced security services should be a requirement, as evidenced by the

North American Electric Reliability Corporation (NERC)⁴ Critical Infrastructure Protection standards (CIP) (ie. CIP-007-6 [25]). This update dictates the required continuous network monitoring and deployment of network defences to detect/block malicious activity within the utilities' perimeter. In typical network deployments, the end-to-end traffic of various applications typically must go through several network services (e.g. firewalls, load-balancers, WAN accelerators). It can also be referred to as Service Functions (or L4-L7 Services, or Network Functions, depending on the source/organisation) that are placed along its path. This traditional networking concept and the associated service deployments have a number of **constraints** and **inefficiencies** [26], such as:

- **Topology constraints:** network services are highly dependent on a specific network topology, which is hard to update.
- **Complex configuration and scaling-out:** a consequence of topological dependencies, especially when trying to ensure consistent ordering of service functions and/or when symmetric traffic flows are needed this complexity also hinders scaling out the infrastructure.
- **Constrained high availability:** as alternative and/or redundant service functions must typically be placed on the same network location as the primary one.
- **Inconsistent or inelastic service chains:** network administrators have no consistent way to impose and verify the ordering of individual service functions, rather than using strict topologies. On the other hand, these topology constraints necessitate that traffic goes through a rigid set of services functions. This often imposes unnecessary capacity and latency costs, while changes to this service chain can introduce a significant administrative burden.
- **Coarse policy enforcement:** classification capabilities and the associated policy enforcement mechanisms are of coarse nature, e.g. using topology information.
- **Coarse traffic selection criteria:** as all traffic in a particular network segment typically has to traverse all the service functions along its path.

As network services become decoupled from the underlying hardware, enterprises have the opportunity to create software virtual networks, thus simplifying deployment and management; a direct impact on CAPEX and OPEX. NFV comes with the promise of a fully virtualised network security solution that does not suffer from feature deprecation when

⁴<https://www.nerc.com>

compared to hardware-based solutions. The above are exacerbated nowadays with the ubiquitous use of virtual platforms, which necessitates the dynamicity and flexibility of service environments. This is even more pronounced in service provider and/or cloud environments, with infrastructures spanning different domains and serving numerous tenants, each with their own requirements. The tenants may share a subset of the providers' service functions, and may require dynamic changes to traffic and service function routing, to follow updates to their policies (e.g. security) or Service Level Agreements. Security can be achieved using the same centralised management platforms used today on physical solutions and promises to offer enhanced visibility into applications, users and content. A further improvement over hardware-based networks is that security can now be performed, in a simplified and transparent manner, on the network device (VNF) itself, thus providing the ability to protect against a wider range of threats. An effective NFV security program has to secure users, clients, applications, data, servers and the network itself. As with other elements of an IT Infrastructure, it is useful to approach NFV security from a functional perspective rather than through the devices involved. This means starting with a list of functions that must be applied holistically rather than attempting to secure each entity separately.

Based on the above, the most critical identified **challenges** that this thesis aims to investigate and address are detailed below:

- **Connectivity and scalability** is required due to the fast-growing number of interconnected users, smart objects and applications. At the network layer, the vastly increased demands require highly efficient programmable connectivity. Scalability in the network infrastructure level requires seamless discovery and bootstrapping of highly efficient orchestration, event processing and analytics and platform integration.
- **End-to-end security and privacy** remains a particularly challenging problem, due to the difficulty of: (a) analysing vulnerabilities in the complex end-to-end compositions of heterogeneous components, (b) selecting appropriate controls (e.g., different schemes for ID and key management, different encryption mechanisms) and (c) preserving end-to-end security and privacy under dynamic changes in applications and security incidents.
- **Reliability and fault tolerance** require the design of reliable network architectures to guarantee also end-to-end availability in order to avoid potential attacks or failures and enable fault tolerance reactive mechanisms at runtime. Furthermore, the provision of reliable and fault tolerance networking is required integrating the current advancements and vulnerabilities of SDN and NFV technologies.

- **Service provisioning and chaining** is one of the main challenges (objectives) of Industry 4.0 aim to reduce the time from several days to several minutes. This also implies the potential to reduce CAPEX and OPEX, especially for short lived service. Moreover, security functions can be chained in order to reduce the impact of the security functions on the network's performance and to alleviate the burden of deploying and managing the security services themselves.

1.3 Research Aims and Objectives

All the above challenges give rise to significant complexities, and relate to the design, implementation and deployment stack of network infrastructures. To address them, the overall aim of this thesis is:

The purpose of this thesis is to develop a pattern framework to enable and guarantee connectivity, S&D behaviour and service provisioning in SDN/NFV-enabled network infrastructures. The pattern framework supports cross-layer intelligent dynamic adaptation. The above will be validated, using diverse usage scenarios to design network topologies, guarantee S&D network properties and provide security network provisioning leveraging service function chaining in SDN/NFV-enabled network infrastructures.

The purpose of the work is to overpass the above open challenges by the use of design patterns applied in a pattern framework. More specifically, to achieve the overall aim of this thesis, the following key objectives were pursued:

- **Objective 1:** Review the literature on the most effective security and dependability challenges, threats and mitigation mechanisms to identify gaps and controls relevant to the overall research aim of the thesis.
- **Objective 2:** Definition of a pattern specification and pattern language for expressing the pattern instances.
- **Objective 3:** Development of a number of pattern instances in order to design network topologies, guarantee security and dependability properties and service provisioning and chaining.
- **Objective 4:** Deployment of a pattern framework able to enforce the developed patterns providing proactive and reactive S&D properties in SDN/NFV-enabled architectures.
- **Objective 5:** Evaluation of the proposed pattern framework, through different use cases and scenarios.

1.4 Research Assumption and Hypothesis

Regarding the proposed approach followed by this research, the following assumptions are made as starting points and directions of this thesis:

- The proposed design of large-scale network topologies in this research does not take in account some additional parameters such as cost, energy consumptions in constrained environments that are considered in deployment of actual topologies.
- The communication protocols, the semantic interoperability and the exposed interfaces between the connected network component are assumed to be compatible during the design and evaluation phase.
- The satisfaction of S&D properties in the design of SDN/NFV-enabled networks through the developed patterns requires the existence of a deployed network containing programmable switches controlled by an SDN controller.
- The focus of this research is to propose a way on how networks can be designed and managed through a model-driven approach based on the developed pattern schema. Therefore, the provision of additional more advanced patterns can offer further designs and deployments ie. in industrial environments but the initial assumption of the importance of the pattern schema will remain the same.
- Finally, the involved components and parties for the security assurance guarantees are assumed to be trusted, fully complied with the required security and trustworthy policies.

1.5 Research Contributions

The main contributions of this thesis contain the specification of a **pattern schema** for the definition of **pattern instances**. In addition, the development of the **pattern framework** together with the deployment of pattern-based **reactive security** leveraging service function chaining are also presented in this research. Moreover, a **network simulator** is implemented to monitor and management of the deployed SDN/NFV-enabled networks. Finally, the last contribution includes the **evaluation** of the proposed framework and developed mechanisms as a complete solution to enforce the different patterns on different use cases and scenarios. All the previously identified developed contributions are also described in the following subsections.

1.5.1 Pattern Schema

The first contribution of this thesis is the specification of the pattern schema. The pattern schema includes the pattern specification and the pattern language.

- The **pattern specification** defines a template able to describe the patterns. These patterns can be used as a model driven reusable solutions to general problems, to design, operate and verify network infrastructures regarding the functional properties satisfied, such as connectivity and scalability, or the non-functional ones, such as security and dependability. The specification contains the basic parts of the pattern such as the name, the problem, the solution and the contribution with respect to the state of the art. The main contribution of the approach is that design patterns can encode designs of SDN/NFV-enabled network topologies, which are proven to satisfy S&D properties and enable the semantic interoperability.
- The definition of a generic **pattern language** is proposed to encode patterns through rule-based reasoning. By the use of this pattern language, patterns are encoded as design solutions that can guarantee network properties for higher layers in the implementation stack of SDN/NFV-enabled networks. The semantics of the pattern language provide interaction with the SDN controller and the NFV MANO for handling and instantiating physical and virtual network component such as nodes, links and service functions. Specific semantics have been also defined to enable the insertion, modification and deletion of OpenFlow (OF) rules through the controller to the programmable switches of SDN/NFV-enabled infrastructures.

1.5.2 Pattern Instances

The second contribution of this work includes the definition of different design patterns able to support network designs, verification, management, monitoring and service provision in SDN/NFV-enabled network architectures.

- **Topology Patterns** can design network topologies to support connectivity, scalability and coverage based on different network topology patterns.
- **Path Finding Pattern** can offer end-to-end connectivity verification and guarantees in network topologies.
- **Dependability Patterns** can assure end-to-end dependable networks enabling reliability, fault detection, restoration and tolerance.

- **Security Patterns** can provide link encryption and end-to-end secure transmissions.
- **Service Function Chaining Patterns** can provide security provision for reactive security based on the defined Service Function Chaining (SFC) patterns.

1.5.3 Pattern Framework

The third contribution of this work is the development of a pattern framework that can be used by designers or administrators of SDN/NFV-enabled networks to (a) create designs of their systems to satisfy network properties, (b) verify if existing designs systems satisfy required properties and (c) provide reactive security monitor and management of SDN/NFV-enabled networks at runtime. The pattern framework consists of two parts: the pattern engine for the design and verification of network infrastructures and the pattern user interface for enabling the insertion of implemented patterns in the pattern engine as described below.

- **Pattern Engine** is implemented to handle design patterns and to interact with network components of SDN/NFV-enabled infrastructure. The pattern engine is based on the Drools Business Rules Management System [27] that enables the insertion, modification, execution and deletion of design patterns expressed as pattern rules. Moreover, the respective interfaces are used to enable also the interaction with core functionalities of the controller and the SDN/NFV-enabled network architectures. Pattern engine is distributed in two layers, i) in the application layer for designing network physical and virtual topologies and ii) in the SDN controller to manage and monitor network configurations.
- **Pattern Graphical Use Interface (GUI)** is developed to provide suitable interfaces to insert defined patterns at design and runtime to monitor and manage network together with pattern handling.

1.5.4 Reactive Security Leveraging SFC

The fourth contribution of this work is the development of a reactive security solution to enhance SFC, enabling also the interaction between the developed pattern framework and the respective SFC patterns. In addition, a number of different mechanisms such as IDS/DPI and honeypot are deployed in order to assure end-to-end security. The presented solution allows the continuous monitoring of the SDN/NFV-enabled networks, with provisions to reduce CAPEX and OPEX based on the traffic classification and forwarding based on the different identified type of traffics. Finally, the reactive security solution provides the capability for interaction with NFV MANO infrastructures enabling the design of virtualised environments.

1.5.5 Network Simulator

The fifth contribution of this work, includes the development of a network simulator to monitor and manage deployed SDN/NFV-enabled network infrastructures. The need for the development of such simulator came after the lack of a simulator able to cover all the involved network elements of our infrastructure. After the development of this simulator, it is possible to preview network topologies at runtime as designed by the related patterns and evaluated through the pattern framework and the reactive security. Finally, the simulator is able to depict not only physical infrastructures but also virtual ones including also the presentation of service function chains.

1.5.6 Evaluation of the Pattern Framework

Finally, the last contribution of this work is the deployment and experimental evaluation of the proposed scheme covering different aspects such as performance, usability, level of assurance in SDN/NFV-enabled designs developed and emulated on different use cases and usage scenarios. The first use case focuses on the design of network topologies based on different topology patterns, the second use case is involved with the design of S&D network designs and the third use case includes the evaluation of the reactive security built upon the pattern framework leveraging SFC in an industrial environment.

1.6 Publications

The contributions of this thesis have been also submitted and published to different journal, technical reports and conference papers. Bellow all published papers are presented.

Journals

- **Nikolaos E. Petroulakis**, Konstantinos Fysarakis, Ioannis Askoxylakis and George Spanoudakis, *Reactive security for SDN/NFV-enabled industrial networks leveraging service function chaining* in Transactions on Network and Service Management, Wiley July 2018.

In this work, we present a Reactive Security Framework for next generation 5G (and SDN/NFV in specific) enabled industrial networks leveraged by SFC. More specifically, considering the energy production critical infrastructures, the framework features enhanced security functions, such as SCADA honeypots, are modelled based upon an operational wind park and ready to be deployed in one. The presented framework allows the continuous monitoring of the wind park industrial network, with provisions

to reduce the impact of the security functions on the network's performance and to alleviate the burden of deploying and managing the security services themselves.

- **Nikolaos E. Petroulakis**, George Spanoudakis, Ioannis G. Askoxylakis, *Patterns for the design of secure and dependable software defined networks* – Elsevier Computer Networks, November 2016.

In this paper, we present a model driven approach to the design and verification of secure and dependable SDN networks that is based on S&D network design patterns (referred to as S&D patterns in the rest of this paper). These patterns can be used to design and/or verify SDN network infrastructures and identify suitable paths and nodes that can guarantee S&D properties. S&D patterns can be used to design SDN infrastructures, and determine also the type, location and connectivity of end nodes with forwarding devices. At the control layer, S&D patterns can ensure secure connectivity between the controllers and the programmable switches.

- **Nikolaos E. Petroulakis**, Elias Z. Tragos, Alexandros Fragkiadakis, George Spanoudakis, *A Lightweight Framework for Secure Life-logging in Smart Environments*, Information Security Technical Report, Elsevier, Volume 17, Issue 3, Pages 58–70, February 2013.

The purpose of this paper is to present in details the current topics of life-logging in smart environments, while describing interconnection issues, security threats and suggesting a lightweight framework for ensuring security, privacy and trustworthy life-logging. In order to investigate the efficiency of the lightweight framework and the impact of the security attacks on energy consumption, an experimental test-bed was developed including two interconnected users and one smart attacker, who attempts to intercept transmitted messages or interfere with the communication link. Several mitigation factors, such as power control, channel assignment and AES-128 encryption were applied for secure life-logging. Finally, research into the degradation of the consumed energy regarding the described intrusions is presented.

Technical Reports

- Adrian Belmonte Martin (ENISA), Louis Marinos (ENISA), Evangelos Rekleitis (ENISA), George Spanoudakis (City University London) and **Nikolaos Petroulakis** (City University London) *Threat Landscape and Good Practice Guide for Software Defined Networks/5G*, European Union Agency For Network And Information Security, December 2015.

In this report, we review threats and potential compromises related to the security of

SDN/5G networks. More specifically, this report contains a review of the emerging threat landscape of 5G networks with particular focus on Software Defined Networking. It also considers security of NFV and radio network access. To provide a comprehensive account of the emerging threat SDN/5G landscape, this report has identified related network assets and the security threats, challenges and risks arising for these assets. Driven by the identified threats and risks, this report has also reviewed and identified existing security mechanisms and good practices for SDN/5G/NFV, and based on these it has analysed gaps and provided technical, policy and organisational recommendations for proactively enhancing the security of SDN/5G.

Conferences

- **Nikolaos E. Petroulakis**, George Spanoudakis, Ioannis G. Askoxylakis, *Fault Tolerance using an SDN Pattern Framework* in Globecom 2017.

In this paper, we propose a pattern framework built in an SDN controller able to import design patterns in a rule-based language in order to provide fault tolerance in SDN networks. We evaluate the framework for network fault tolerance as it appears to be a critical topic for research and we propose suitable patterns able to guarantee network connectivity, fault detection, fault restoration and fault tolerance. To evaluate the SDN Pattern Framework, we import our patterns to handle network component as retrieved by the inventory of the controller. Patterns are able to install flows and decrease the failover time in case of faults and network failures providing fault tolerance in SDN network architecture.

- Konstantinos Fysarakis, **Nikolaos E. Petroulakis**, Andreas Roos, Khawar Abbasi, Petra Vizarreta, George Petropoulos, Ermin Sakic, George Spanoudakis, and Ioannis Askoxylakis, *A Reactive Security Framework for Operational Wind Parks Using Service Function Chaining*, ISCC 2017.

This work highlights the benefit of leveraging the flexibility of SDN/NFV-enabled networks to deploy enhanced, reactive security mechanisms for the protection of the industrial network, via the use of Service Function Chaining. Moreover, a proof of concept implementation of the reactive security framework for an industrial-grade wind park network is presented. The framework is equipped with SDN and Supervisory Control and Data Acquisition (SCADA) honeypots, modelled on (and deployable to) an actual, operating wind park, allowing continuous monitoring of the industrial network and detailed analysis of potential attacks, thus isolating attackers and enabling the assessment of their level of sophistication.

- **Nikolaos E. Petroulakis**, George Spanoudakis, Ioannis G. Askoxylakis, Andreas Miaoudakis and Apostolos Traganitis, *A Pattern-Based Approach for Designing Reliable Cyber-Physical Systems*, Globecom 2015.

The purpose of this work is the development of a pattern-based approach for the design of CPS. The main contribution of the approach is that encodes designs of CPS, which are proven to satisfy S&D properties, as CPS design patterns. The first set of S&D patterns includes the Reliability Component Composition (RCC) Patterns for designing reliable CPS. RCC patterns are encoded in Drools, which is a rule-based reasoning system. To evaluate our approach, we use RCC patterns as a methodology for designing a reliable wireless sensor network attached to a physical architecture to send monitored data to a central controller through relay nodes and paths.

- **Nikolaos E. Petroulakis**, Ioannis G. Askoxylakis, Apostolos Traganitis, George Spanoudakis, *A Privacy-Level Model for User Centric Cyber Physical Systems*, In the 1st International Conference on Human Aspects of Information Security, Privacy and Trust (affiliated with the 15th International Conference on Human-Computer Interaction), Las Vegas, USA, July 2013.

This work presents an overview and analysis of the most effective attacks, privacy challenges and mitigation techniques for preserving the privacy of users and their interconnected devices. In order to preserve privacy, a privacy-level model is proposed in which users have the capability of assigning different privacy levels based on the variety and severity of privacy challenges and devices' capabilities. Finally, we evaluate the performance of specific CPSs at different privacy-levels in terms of time and consumed energy in an experimental test-bed that we have developed.

Other Related Publications of Candidate

- Petra Vizarreta, Amaury van Bemten, Ermin Sakic, Khawar Abbasi, Khawar, **Nikolaos E. Petroulakis**, and Wolfgang Kellerer and Carmen Mas Machuca, Incentives for a Softwarization of Wind Park Communication Networks", IEEE Communications Magazine, 2019. Othonas Soutatos, George Spanoudakis, Konstantinos Fysarakis, Ioannis G. Askoxylakis, George Alexandris, Andreas I. Miaoudakis, **Nikolaos E. Petroulakis**: Towards a Security, Privacy, Dependability, Interoperability Framework for the Internet of Things. CAMAD 2018:
- E. Sakic , V. Kulkarni, V. Theodorou, A. Matsiuk, S. Kuenzer, **N. E. Petroulakis** and K. Fysarakis, VirtuWind—An SDN-and NFV-Based Architecture for Softwarized Industrial Networks. In International Conference on Measurement, Modelling and Evaluation of Computing Systems, Munich, February 2018.

- Ioanins Askoxylakis, **Nikolaos Petroulakis**, Vivek Kulkarni, Florian Zieger, VirtuWind–Security in a Virtual and Programmable Industrial Network Prototype Deployed in an operational Wind ParkI Askoxylakis - ERCIM News, 2016.
- **Nikolaos E. Petroulakis**, Toktam Mahmoodi, Vivek Kulkarni, Petra Vizarreta, Andreas Roos, Khawar Abbasi, Xavier Vilajosana, Spiros Spirou, Anton Masiuk, Ermin Sakic, Yannis Askoxylakis, VirtuWind: Virtual and Programmable Industrial Network Prototype Deployed in Operational Wind Park, EUCNC 2016.
- Alexandros G Fragkiadakis, Vasilios A Siris, **Nikolaos E. Petroulakis**, and Apostolos Traganitis. Detection of Jamming Attacks, Local versus Collaborative Detection. in IEEE Wireless Communications and Mobile Computing, 15 (2), 276-294, 2015.
- Ioannis G Askoxylakis, Antonis Makrogiannakis, Andreas Miaoudakis, Stefanos Papadakis, **Nikolaos E. Petroulakis**, Manolis Surligas, Apostolos Traganitis, Nikolaos Vervelakis, 2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Athens, Greece, 1-3 December 2014,
- Andreas I. Miaoudakis, **Nikolaos E. Petroulakis**, Diomidis Kastanis, Ioannis G. Askoxylakis, "Communications in Emergency and Crisis Situations", in the 2nd International Conference on Distributed, Ambient and Pervasive Interactions, (affiliated with the 16th International Conference on Human-Computer Interaction), Heraklion, Greece, July, 2014.
- Panos Chatziadam, Ioannis G. Askoxylakis, **Nikolaos E. Petroulakis**, Alexandros G Fragkiadakis, "Early Warning Intrusion Detection System", in the 7th International Conference on Trust and Trustworthy Computing (TRUST), Heraklion, Greece, June 2014.

1.7 Outline of Thesis

The rest of this thesis is organised in 7 chapters as follows:

- Chapter 1 includes an introduction to the topic, motivation, objectives, main contributions and publications.
- Chapter 2 provides background and literature review including also an overview analysis of the most effective security challenges, fault detection and mitigation techniques for preserving security and dependability of network infrastructures.

-
- Chapter 3 defines the conceptual model of the pattern schema and the pattern language.
 - Chapter 4 presents different design pattern instances to design and verify network topologies that guarantee security, dependability and service provisioning.
 - Chapter 5 describes the implementation of an architecture, including the development of the pattern framework, the reactive security and the network simulator.
 - Chapter 6 provides the evaluation of the proposed framework to design and verify SDN/NFV-enabled networks on different use cases.
 - Chapter 7 presents concluding remarks of this research, limitations and future objectives.

Chapter 2

Background and Literature Review

2.1 Overview

SDN/NFV-enabled networks face a number of security, privacy and dependability issues which need to be resolved in order to step forward into the interconnected world and the Future Internet. Furthermore, the discrepancy between technologies and the attempt to interconnect them have brought new security challenges and gaps which need to be filled. In this chapter, an overview analysis of security and dependability attributes is presented. In addition, the most effective attacks, privacy challenges and mitigation techniques for preserving security, privacy and dependability in SDN/NFV-enabled networks are analysed. Security and privacy threats are occurred from passive attacks, such as eavesdropping and traffic analysis or from active attacks, such as impersonation and jamming. Suitable countermeasures which can be applied in generic and SDN/NFV-enabled networks independently of devices' capabilities and operating systems, are described to protect data transmission, identity, location and routing paths.

2.2 Security and Dependability Attributes

The general concept of CIA triad (Confidentiality, Integrity and Availability) supported by [28] can be applied successfully for the secure data transmissions. In addition, dependability is the ability of a system to deliver its intended level of service to its users [29]. The main attributes which constitute dependability are reliability, availability and fault tolerance. More specifically, the core security and dependability attributes can be defined as follows:

- **Confidentiality** focuses on keeping information private and ensuring that only the right people will have access to it [30]. Confidentiality is ensured when information

does not disclose to not authorised agents. This means that data should be not revealed not only from the component which reacts as the source or destination, but also through all the intermediate components. In addition, confidentiality is also related to privacy. A passive listener can easily identify traffic without being identified. The widespread use of network devices on uncontrolled environment endangers the possibility of disclosing private data that should not be revealed. For instance, a wireless installed camera for recording possible intruders, connected to an insecure home network could be a susceptible threat to disclose the private actions of a family. Privacy is a major concept in insecure smart environments in which a variety of data from users and devices are exchanged and collected. The mass production and transfer of sensitive data exposes the danger of privacy violation.

- **Integrity** guarantees the degrees of complete, consistent and accurate data [30]. Integrity confirms also that data has not been modified. Impersonation attacks involve the interaction of an adversary with the human user. A malicious node can easily intercept transmitted information or impersonate a receiver. The adversary acts either as a man in the middle or as a masquerade, pretending to be a legal node in the network to apply spoofing attacks. These kinds of attacks not only appear to be critical for a user's privacy but also the consequences of such attacks can be extremely dangerous.
- **Availability** guarantees that information is available when it is needed [30]. The lack of channel availability has a severe influence on the security of network. Network availability is the ability of a system to be operational and accessible when required for use.

In addition to the CIA concept, other properties such as *reliability*, *fault tolerance* and *authentication* can be added in the list of S&D properties.

- **Reliability** is the ability of a system to perform a required function under stated conditions, for a specified period of time [30]. It is an attribute of system dependability and it is also correlated with availability [29]. For hardware components, the property is usually provided by the manufacturer. This is calculated by the complexity and the age of the component. Moreover, reliability in networks is the probability of successful packet reception [31]. Different network topologies such as star, hierarchical/tree or mesh networks affect the reliability of the network and of the system respectively. Other factors which affect the reliability of a link are the transmission range of the signal strength, noise, fading effects, interference, modulation method, and frequency. Especially in wireless networks, the reliability of links can be classified into two

main categories the deterministic models and the probabilistic ones. The exponential increase in reliability can lead to a net increase in the energy efficiency.

- **Fault Tolerance** is the ability of a system or component to continue normal operation despite the presence of hardware or software faults [30]. Network fault tolerance appears to be a critical topic for research [32]. Fault detection, fault restoration and fault tolerance can be included also in the network characteristics related also to network availability.
- **Authentication** is the process of determining whether someone or something is, in fact, who or what it is declared to be [33]. The main scope of authentication is the provision of personal identification including verification of user knowledge, ownership and user characteristics.

Apart from the non-functional S&D properties, there are some functional properties also related to the QoS and the S&D properties that affect the design of network topologies such as connectivity, scalability and coverage as described below:

- **Connectivity** is the property that given a network, a path between end nodes can be determined. Moreover, a network is connected if every pair of nodes are connected through a path.
- **Scalability** is the ability of a network to change in size or scale. This includes the capability of a seamless discovery and bootstrapping of additional components as well as highly efficient orchestration, event processing and analytics and platform integration.
- **Coverage** on network architectures is the property describing the geographic area where the nodes can communicate and the capability of service provision and assurance.

Finally, Figure 2.1 depicts the correlation between non-functional and functional S&D attributes.

2.3 Security and Dependability Threats

S&D threats can be classified in generic ones, radio access or SDN/NFV related ones as described in the next subsections.

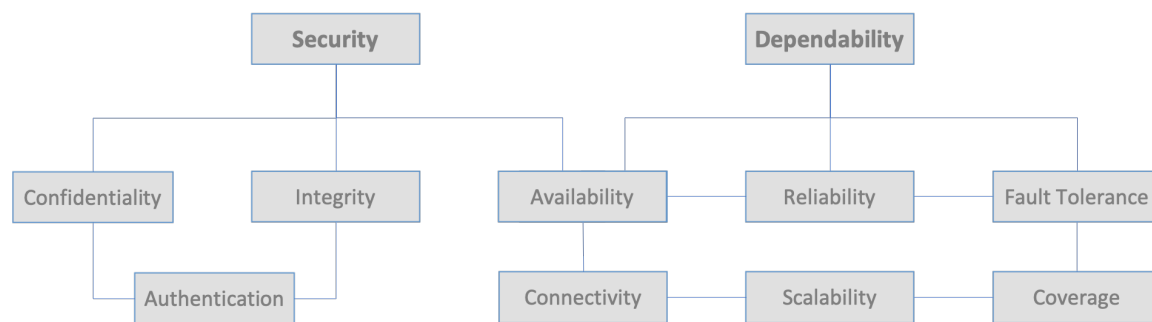


Fig. 2.1 Relation Between non-Functional and Functional Attributes

2.3.1 Generic Threats

Security and privacy attacks may include physical and cyber tampering or compromising devices and can be classified to passive and active ones. Passive are the attacks where an adversary monitors traffic without interacting with the victim or modifying transmitted data. The most common passive attacks are eavesdropping and traffic analysis. An active attack occurs when an adversary attempts to modify exchanged messages, destroy the communication or replay transmitted data. The most severe active attacks are impersonation and denial of service.

2.3.1.1 Confidentiality Threats

The most **confidentiality threats** include the eavesdropping and the traffic analysis as described below:

- **Eavesdropping** occurs when an adversary stealing personal data, monitors and listens to the exchanged data with the intention to extract private data. That includes traffic sniffing, which involves tapping data flows within a network enabled by weak or no encryption in the relevant interface. Privacy seems to be quite challenging because of the inability of devices to anticipate and sense possible eavesdroppers. The disclosure of sensitive information such as identities and message payload, are severe privacy violations from eavesdropping. For example, the disclosure of sensed medical data such as patient's personal data, blood pressure, vital signs or sugar level, transmitted to a remote hospital or to a doctor's office, may reveal the patient's identity and condition.
- **Traffic analysis** attacks can be applied by adversaries who do not have the ability to decrypt data payload. However, they can obtain private information such as data sources, the location of devices and data routes. This can be done by the use of sniffers and packet analysers on the wireless data transmission for tracking the traffic

flow information hop-by-hop [34]. The disclosure of sensitive information about the location, track and identity of a user may cause significant problems for him and his interconnected network and users. The problem of the panda and the hunter describes the situation in which scientists attempt to locate the position of a panda but they have to hide its location from panda hunters as well [35]. Revealing the topology, nature and routing paths of a transmission could be used by adversaries to track, destroy interrupt and invade the privacy of a network. Moreover, the danger of a compromised relay node is a result of location disclosure.

- **Impersonation Masquerades/Identity spoofing** identity spoofing retrieves and masquerades a legitimate entity of software component or human agents. In the case of SDN, an attacker can impersonate a legitimate controller to instantiate network flows, divert traffic etc.
- **Tampering** defines the altering of packet content through the network by an attacker.

2.3.1.2 Availability Threats

The most common **availability threats** occur by the Denial of Service attacks or through other physical network threats as described below:

- **Denial of Service(DoS)** results in lack of network availability. DoS can include any kind of attack which attempts to make the network resources unavailable but also causing reduction or disruption of a service. An active adversary may apply DoS attacks by destroying or modifying the communication channel. In particular for SDN, DoS threats may occur to all layers of the SDN reference architecture. Flood bandwidth or resources of network elements can be caused by DoS attackers at the data plane. This threat in many occasions originates by multiple compromised systems, such as botnets, which are flooding the targeted network with traffic. Congesting controllers through a large number of forged flow arrivals, causing network performance degradation and interruption may occur in the control plane. Traditional defences are ineffective in the cases of SDN control plane DoS attacks because DoS defences' approaches focusing on protecting data plane. Finally, at the application layer, network management applications can be severely affected severely by DoS.
- **Physical Network Threats** are usually located in uncontrolled environments where physical attacks might occur [34]. Physical threats include attacks related to destroying, disabling, stealing or altering physical and cyber infrastructures. Damages and losses are threats related to intentional or unintentional destruction of network infrastructures.

Failures and malfunctions are threats which includes failures or insufficient functioning of network infrastructures.

2.3.2 Radio Access Threats

Since network elements, such as access points and sensors, are usually equipped with one wireless communication radio, security threats are mostly compared to wireless networks threats. The main difference between wireless and wired networks is that suitable security mechanisms are absent because of the lack of respective architectures and resources. In SDN, there are some additional threats due to arising 5G technology [36, 37]; in particular the use of wireless communication in 5G. Such threats, as described in [38] can be distinguished into:

- **User emulation:** The wireless medium can be exploited by adversaries that mimic incumbent signals. Nodes launching such attacks can be (i) greedy mobile nodes that by transmitting fake incumbent signals force all other users to vacate a specific band (spectrum hole) in order to acquire its exclusive use and (ii) malicious mobile nodes (adversaries) that mimic incumbent signals in order to cause DoS attacks. Malicious nodes can cooperate and transmit fake incumbent signals in more than one band, thus causing extensive DoS attacks making a radio hop from band to band, severely disrupting its operation.
- **Spectrum sensing data falsification:** The received signal power may become lower compared to path loss models due to transmission features such as signal fading, multi-path propagation, etc. [39]. This may lead to harmful interference due to undetected primary signals.
- **Medium Access Control (MAC) layer attack:** This category of attacks includes:
 - MAC spoofing, where attackers send spurious messages aiming to disrupt the operation of network(e.g. channel negotiation),
 - Congestion attacks, where attackers flood common control channel in order to cause an extended DoS attack.
- **Jamming attacks:** are very serious security threats in wireless transmission in secure or insecure communication channel due to collisions and channel occupations [40, 41]. The preservation of security is disrupted when an attacker applies collisions or jamming attacks creating electromagnetic interference. An adversary, causing interference in a channel in which users interchange sensitive or critical messages, may cause reportable

privacy violations, such as data destruction or infinite retransmission of messages, exhausting the batteries of resource constrained devices. Furthermore, the delayed transmission of critical information, such as private medical data of a patient to the doctor's database, means the patient's safety might be endangered.

2.3.3 SDN/NFV Threats

To secure SDN and NFV network deployments the related threats have to be considered.

2.3.3.1 Software Defined Networking Threats

Having a centralised controller architecture as well as total control network programmability, SDN introduces new threats in addition to those of classic network deployments. These threats can be extracted by examining the generic SDN architecture shown in Figure 2.2 and are described as follows:

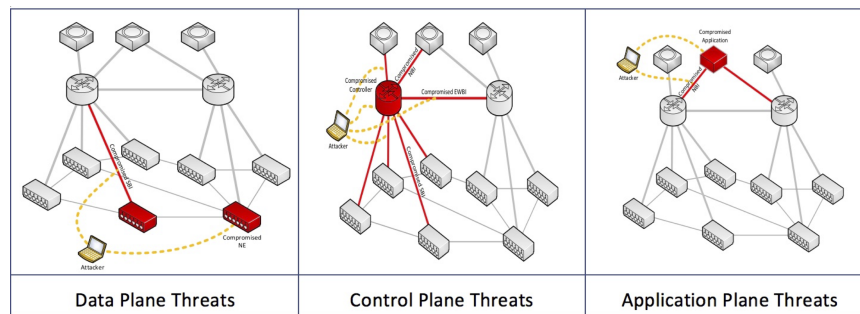


Fig. 2.2 SDN Threats

- **Data forging:** The injection of faked flows to the SDN switches in order to cause DoS attack.
- **API exploitation:** The exploitation of the APIs that the control plane uses to communicate with the SDN devices including: i) the SBI for the communication between controllers and switching devices ii) the NBI for the communication between controllers and application and iii) the EWBI for the communication between different controllers.
- **Controller exploitation:** The SDN controller is running a so called Network Operation System (NOS) so any vulnerability of the NOS implementation poses a threat to the network.

- **Traffic diversion:** This threat includes compromising a network element in order to divert traffic flows and to enable eavesdropping. Traffic diversion is a data plane related threat. In the case of network virtualisation, a specific traffic diversion ensues when isolation between network slices is compromised or when the enforcing access to a slice in the edge equipment is either bypassed or misconfigured.
- **Management exploitation:** The workstations that are used to manage the network are a well-known attack target. Compromising the management systems can be used to maliciously alter the network behaviour. The ability of the SDN managing to easily reconfigure the whole network makes this thread more critical.
- **Application exploitation:** SDN applications have access to various network functionalities. A vulnerable application can be exploited to access and maliciously alter network functionalities.

2.3.3.2 Network Virtualisation Threats

Network virtualisation threats [42–45] are threats related to the underlying IT infrastructure used for virtualising network operations. Although NFV is a promising solution for service providers, it encounters certain security challenges that could have detrimental effects on its performance and could cause a security threat which may hinder its implementation in the telecommunication industry.

The move to a virtual environment must be done carefully and with an understanding of how the new infrastructure will change IT planning and management. In addition, proper planning guarantees the existence of mechanisms that can help tackle any virtualisation challenge. Some common threats to services running on virtual environments include:

- Generic virtualisation threats (e.g. memory leakage)
- Network function threats (e.g. distributed DoS attacks)
- Hybrid threats, as a result of the union of proprietary and NFV technologies.

In reference to the taxonomy of threats outlined NFV [46] threats can be seen as threats under the *Nefarious Activity/Abuse* and *Eavesdropping/Interception/ Hijacking* categories.

2.4 Security and Dependability Countermeasures

In an insufficient security environment, new lightweight approaches should be considered in order to overcome the lack of trust and privacy for thereby avoiding security dangers. There-

fore, suitable countermeasures which can be applied in a variety of devices and operating systems, are described to protect data transmission, identity, location and routing paths.

2.4.1 Generic Countermeasures

2.4.1.1 Encryption for Confidentiality

Encryption is an efficient countermeasure to protect confidentiality and to limit access control of data in case of a passive attack such as eavesdropping or active attacks such as impersonation by the use of cipher and cryptographic check sums. When cryptographic algorithms are not used, an attacker can eavesdrop or compromise the transmitted data easily. Nevertheless, if there is no encryption applied, a malicious node can easily intercept transmitted information or impersonate a receiver. In addition, an adversary knowing the password may decrypt ciphered data. To avoid the danger of revealing the encryption key, a predefined set of anonymous keys changing frequently could protect the encrypted transmission between nodes. If the payload message is encrypted then a malicious node that has the encryption key can also launch an impersonation attack. For instance, an impersonation attack on devices interconnected with a patient, may cause false alarms or the modification of medical data can put patient's life in danger.

The goal of cryptography is to guarantee all the major properties such as confidentiality, integrity, authentication and non-repudiation. There two type of encryption asymmetric and symmetric.

- **Symmetric encryption** applies the same key for decryption and encryption. Non-repudiation cannot be guaranteed using symmetric encryption because the use of sharing keys cannot determine which party applied message encryption and decryption.
- **Asymmetric encryption** depends on the use of two keys the public and private. The owner of the private key is able to encrypt data which can only be decrypted by the use of the public key. On the other hand, data that is encrypted with the public key of an owner can be decrypted only by the use of the private key.

Compared to symmetric encryption, asymmetric encryption is also computing and time consuming process. The encryption of data by the use of the private key cannot guarantee the confidentiality property since all the users can decrypt messages by the use of public key. However, integrity can be guaranteed since data cannot be modified in addition to non-repudiation and authenticity. Conversely, encryption by the use of the public key can guarantee confidentiality since only the owner of the private key can decrypt the message but

it cannot guarantee integrity and authentication since anyone can decrypt messages using a public key.

Existing solutions include either link encryption or end-to-end encryption to protect data confidentiality.

- **Link encryption** is able to encrypt not only the payload of a packet but also the packet header, addresses and destination. Only instructions and parameters to synchronise communication methods for data transmission are not encrypted. The packet must be encrypted and decrypted in every hop by encrypting and decrypting packet using different security keys. Although link encryption is a time and resource consuming procedure, it is able to protect efficiently data transmission from eavesdroppers and traffic analysis sniffers. However, link encryption usually is incorporated in protocols. In this case, routers can decrypt the header to read the address and encrypt and send it to destination. Link encryption occurs in the data link and physical layers of the lower OSI layers.
- **End-to-end encryption (E2E)** is able to encrypt the payload of the packet and not the header. The data is encrypted in the source and decrypted in the destination. The user is able to select whether specific data should be encrypted or not. E2E encryption does not require relay nodes to have the encryption keys. However, the data is vulnerable to attackers due to the unencrypted header data. Formal verification with E2E encryption for secure networks is proposed in [47]. The advantages of E2E security methods are described in [48].

Network encryption can be applied by the use of Internet Protocol Security (IPsec), Secure Socket Layer (SSL), Transport Layer Security (TLS) and Secure Shell (SSH). The main differences of the two protocols are described in [49]. More specifically the most common encryption methods can be found below:

- **Wi-Fi Protected Area (WPA/WPA2):** security protocol for wireless networks designed to improve the WEP. WPA is based on TKIP when WPA2 is based on the extensible authentication protocol (EAP) and AES.
- **Encryption protocols:** SSL (Netscape)/TLS (IETF) (layer 5), SSH (5-7), PGP, Kerberos, IPsec
- **Encryption algorithms for integrity:** SHA1, SHA256, SHA512, SHA3
- **Encryption algorithms for confidentiality:** AES, DES, Triple DES

- **In application layer:** HTTPS for HTTP, LDAPS for LDAP, WSS for SOAP
- **Other methods** include 802.1x for authentication, KS,KIV, ATM, 802.1AE or MAC-Sec - provides data confidentiality, data integrity and data origin authentication, point-to-Point Tunnelling Protocol (PPTP)

In order to guarantee integrity, **Message Integrity Codes (MICs) or Message Authentication Codes (MACs)** can be used to mitigate tampering of data and guarantee integrity on data transmission. These codes ensure the integrity of the MAC header and the attached payload data. If the messages exchanged between the nodes have a MIC in the headers and payload, it is impossible for an attacker to launch a successful impersonation attack but it is possible to become a passive listener.

IPSec [49] is a protocol in Layer 3 of OSI model that can send and receive cryptographic type of messages such as Transport Control Protocol (TCP), User Datagram Protocol (UDP). IPSec can be used in end-hosts, switches ,gateways and routers enabling end-to-end security. IPSec flow protection has been proposed by Internet Engineering Task Force (IETF) [50] for SDN traffic protection [51]. Based on this approach, a flow protection policy should be defined in order to establish E2E security between hosts and gateways. The main parts of IPSec include the following:

- **IPSec Mode:** IPSec operates in two modes the transport and tunnel one. In **transport mode**, there is no modification in the IP header but it is used to secure the communication between end to end traffic transmission by encapsulating and securing only the payload. The **tunnel mode** includes the hash of the entire packet including also the IP header and is used to tunnel traffic between routers and gateways.
- **IPSec Headers:** IPSec headers can provide security in two ways either by the use of the Authentication Header (AH) and **Encapsulating Security Payload (ESP)**. Both AH and ESP can protect *data origin authentication* by authenticating the source traffic enabled by the suitable of MACs (i.e. DSA/RSA) [52]. Moreover, *data integrity* is enabled by the support of integrity checks algorithms (ie. MD5/SHA). To ensure that data has not been altered and *anti-replay* can be guaranteed by detecting rejected replayed packets in the receiver [53]. In addition, ESP can also support *data confidentiality* by the use of symmetric key encryption algorithm (ie. DES/3DES, AES) to encrypt the packets before the submission. The IPSec framework can be found in Figure 2.3 [54].
- **IPSec Sessions:** The procedure of the session establishment is made through the Key Exchange (IKE). Internet Security Association and Key Management Protocol (ISAKMP) provides a method for automatically setting up **Security Associations (SA)**



Fig. 2.3 IPsec Framework

and managing their cryptographic keys. SA supports Security Association Database (SAD), Security Policy Database (SPD) and Security Parameter Index (SPI) to define the needed algorithm for packet decryption. IPSec has also the capability to establish secure session by the use of SA for both AH and ESP but not at the same time. When ESP header is applied, an IPSec header is added to the packet containing the SPI field which includes the destination address, the protocol and the SA.

- **IPSec Policy Procedure:** To protect traffic, traffic classification can be done by the use of Access Control List (ACL) to permit the traffic that should be encrypted or deny the traffic sent unencrypted [55]. The procedure involves also peers authentication, SA negotiation and finally the creation and termination of encrypted tunnels.

2.4.1.2 Fault Tolerance for Availability

The most common solutions to guarantee fault tolerance and avoid a single point of failure, include the replication of paths forwarding traffic in parallel, the use of redundant paths and the ability to switch in case of failure (failover) and traffic diversity. Fault tolerance

mechanisms exist in both legacy and SDN. More specifically, for SDN there are three types of network failures: in data plane failures (link or switches), control plane (connection between switch and controller) and controllers. Traditional defence approaches focus on protecting the data plane and are therefore ineffective in the cases of SDN control plane attacks [23]. A list of a variety of fault tolerance mechanisms are presented in the SDN/NFV countermeasures section.

2.4.2 Radio Access Countermeasures

For the radio access protection, a number of different countermeasures are proposed as described below.

- **Anonymity** of source node's identity and location assures that path will reach their destination through trusted intermediate nodes. To protect the identity of source and receiver, identities of messages should be hidden either encrypted or not undefined [56]. Pseudonyms can be an effective way to hide the real identity of a node. Although pseudonyms seem to be an effective solution, fixed pseudonyms cannot prevent adversaries from deducing the topology of the network through traffic analysis [57]. Furthermore, changing identities frequently may thwart attackers from identity disclosure.
- **Location protection** can be established by hiding the received signal strength and the time interval of a wireless network element [58]. To hide the location of the transmitter, variations in signal strength and in time delay are employed. When actual encrypted data are not exchanged, dummy messages can be sent to mask the channel, hiding the actual data transmission. This mechanism can keep the bandwidth constant and hide the traffic to confuse passive listeners from effective eavesdropping and traffic analysis [34, 59]. Works, such as [59–61], focus on location privacy and route protection, providing partial privacy protection. Furthermore, if a packet has reached the range of radio waves, then it is difficult to locate the source [57]. This can be done by the use of multi-hop routing which can also prevent adversaries from identifying the source and the routing paths of transmissions.
- **Frequency hopping** (or channel hopping) can be used to avoid jamming attack [62, 63]. Investigations on channel assignment, based on energy detection and received signal strength in wireless networks, are presented in [64]. It can also prevent continuous passive listening attacks [65], protect source location and routing paths and assure data transmission [66].

- **Signal strength increment** can mitigate weak jamming attacks [66]. Suitable mitigation techniques include also the detection using algorithms based on dropped packets or the decrease of the signal to noise ratio [40]. Data transmission is assured by the use of acknowledgments. Even though acknowledgement mechanisms do not guarantee data integrity, their use can ensure valid packet reception.
- **Intrusion detection of jamming attacks** based on experiments in 802.11 using software defined radios, are described in [67, 68]. Theoretical analysis and simulation results for mitigating mechanisms to detect jam attacks in wireless sensors networks are presented in [69, 70].
- **Multi-path and multi-hop routing** is applied to protect the topology of routing paths. Changing routing paths may thwart adversaries from jamming attacks [60]. Finally, the creation of fake paths could potentially prevent an adversary from tracking the routing path and destroying the transmission [59, 71].
- **Random delay slots** can be used for collision avoidance.

2.4.3 SDN/NFV Countermeasures

Various techniques have been proposed to ensure security and network function resilience. The state of the art in protecting SDN can be summarised as follows:

- **Trust between devices and controllers** is of paramount importance as only a trustworthy infrastructure can provide a dependable network. Trustworthiness between SDN devices is a must and can be implemented as mandatory authentication or as a factor that is appointed to devices. A device must maintain its trustworthiness level in order to be part of the SDN ecosystem.
- **Trust between applications and controller** software can be tricky as software is meant to change and evolve as well as become untrustworthy when vulnerabilities and exploits surface. A trustworthiness model that monitors and evaluates all SDN software becomes a key player in achieving application reliability and resilience.
- **Security domains** can help minimise exposure and risk by isolating software components such as SDN applications and core controller function. Proven techniques such as sandboxing have already paved the path towards this type of isolation. A predefined minimal set of interactions allows proper levels of communication and interoperability.

- **Secure components** are necessary for the SDN infrastructure. Tamper-proof device techniques such as Trusted Computing Base (TCB) can help assure data confidentiality if a system is compromised or stolen.
- **Fast and reliable software update and patching** will ensure that the SDN software is secure (vulnerability patching) and performing at its best, utilising the hardware to the fullest. Software updates and patching are perhaps the most important, totally effective and least convoluted elements of proactive security.
- **Reliable connectivity** in 5G and SDN can be established in recover network failures as proposed by [72] either creating automatic tunnel re-establishment or on-demand tunnel re-establishment.

In order to provide efficient secure solutions in existing SDN environment, different frameworks and products have been developed.

- **SDN Flow integrity and conflict resolution:**
 - **FortNOX** is an extension of the NOX controller that provides flow rule contradiction validation via a non-bypassable mediation real-time service [73]. This is performed during the rule insertion requests of OpenFlow applications.
 - **FlowChecker** proposes a tool for verifying the consistency of different OpenFlow enabled switches, validating the correctness of the flow table entries, debugging reachability and security issues [74].
 - **NICE** is a tool for automating the testing of OpenFlow applications that combines model checking and symbolic execution to quickly explore the state space of unmodified controller programs written for the popular NOX platform [75].
 - **FLOVER** decomposes network security policies in sets of assertions referred to as non-bypass properties that specify whether a certain packet/flow matching a set of conditions should be forwarded to its destination [76].
- **Strong Authentication:** Ethane [77] utilises a Central Domain Controller (CDC) that enables controlled admittance and routing of flows by implementing global-policy validated and strongly-authenticated bindings between users, devices and services. FortNOX is a software extension that utilises role-based authorisation and security constraint enforcement [73].
- **Encryption on the SBI/NBI/EWBI:** The communication channel between each SDN layer must be well protected. As a security measure, techniques such as secure coding,

deployment of integrity checks and application for digital signing, should be used. Moreover, all communication channels can be hardened using TLS security [78]. Although OpenFlow protocol is not the only protocol utilised in SDN, its TLS support can protect the communication channel efficiently. However, the lack of TLS version or reference in OpenFlow specification [79] could cause the adaption of non-interoperable implementations and their TLS version vulnerabilities that have been fixed in later version such as man-in-the-middle attacks in TLS 1.0 and 1.1 or the lack of 1.2 to support newer cryptographic algorithms [80, 81].

- **Access Control:**

- **Panopticon** ensures E2E network policy by implementing a mechanism that ensures the routing of inbound network traffic via at least one SDN capable switch [82]. This mechanism carries the name of Solitary Confinement Tree and utilises VLANs to perform the necessary traffic flow. This is particularly important when dealing with Hybrid SDN infrastructures.
- **FlowNAC** is a Network Access Control solution that implements access to services as a set of flows that can independently requested but only simultaneously authorised [83]. Policies are formed dynamically as the result of data plane services, providing holistic authentication and authorisation capabilities locally at the data plane level. This authentication and authorisation is performed on per service basis.

- **Application isolation and Sandboxing:**

- **Rosemary** introduces a micro-NOS sandboxing strategy to safeguard the control layer from errant operations resulting from network applications [84]. Rosemary supports context separation, resource utilisation monitoring, and a micro-NOS permissions structure which limits the library functionality that a network application is allowed to access.
- **PERMOF** is a fine-grained permission system that introduces a shim layer between application and network operating system [85]. This shim layer acts as an isolation mechanism that prevents application direct access to kernel code or memory. Caller identity is used to perform permission control on a pre-configured policy.

- **Security Enhancements:**

- **FRESCO** is a security application development framework that is intended to enable the quick design of detection and mitigation modules based on OpenFlow [86]. It provides a scripting API that enables the development of security monitoring and threat detection logic as modular libraries.
- Security Enhanced Floodlight (**SE-Floodlight**) [87] is an implementation of an OpenFlow security mediation service for enforcing network security. It is similar to FRESCO except there is more functionality due to the extensions set by the new OpenFlow specification.
- **AVANT-GUARD** is a data plane extension consisting of actuating triggers and a connection migration module [88]. It is designed to enhance the scalability of SDN security applications and thus their ability of tackling a dynamic range of network threats. A connection migration module provides shielding to the control plane from data plane originating saturation attacks.

Suitable fault tolerance mechanisms (Figure 2.1) have been developed in SDN that intend to guarantee fault tolerance as described in the following:

- **DefenceFlow** is a commercial application that detects and resolves DoS attacks [89]. Its operation is based on pattern matching techniques performed on traffic statistics collected from SDN forwarding devices. In case of DoS detection, DefenceFlow, redirects traffic to the nearest mitigation device. Mitigation devices can be placed in any location within the SDN network.
- **HP Sentinel Security** implements an SDN application that monitors the flow creation process. Flows are compared to a threat reputation database in order to verify IP Address and DNS name [90]. If the lookup is positive, traffic is dropped on the forwarding devices. The NOX controller is one of the well-known SDN controllers in regards to its expandability and enhancement capabilities. One very important enhancement enables NOX to exercise traffic anomaly detection capabilities [91].
- **FatTire** [92] is a language for writing fault-tolerant SDN programs in terms of paths through the network and explicit fault-tolerance requirements. The main features of FatTire include fast-failover OpenFlow mechanisms, and correct behaviour during periods of failure recovery. It focuses on data plane including the fault tolerance between the source, the destination and the intermediate functions such as IDS and firewall. The paths are defined by the use of the breath-first-traverse algorithm.

- **Coronet** [93], on the other hand, proposes a fault tolerant system for SDN controller which is able to receive topology information and protect data plane link/switch failures and is based on Dijkstra's shortest path algorithm.
- **Ravana** [94] is a fault tolerant SDN controller platform that processes the control messages transactionally and supports fault tolerance for both controller and switches. The advantage of Ravana is that (i) adapts replicated state machines for control state replication and adds mechanisms for ensuring the consistency of switch state and (ii) extends existing channel interface between controllers and switches. Ravana is evaluated in the Ryu controller which supports ZooKeeper.
- **SMaRtLight** [95] is a practical fault-tolerant SDN controller which is based on a shared database of stored network states. It supports primary and backup controllers (master slave) are used to replicate SDN controllers. Paxos algorithm is used to implement Replicated State Machine.
- **LegoSDN** [96] is able to tolerate SDN application failures,
- **Onix** [97] supports scalability and availability in SDN control platforms.
- **AFRO** [98] proposes an automatic failure recovery for POX SDN controller based on simpler, failure-agnostic controller modules.

Table 2.1 SDN Fault Tolerance Mechanisms

Name	Description	Controller
FatTire	language for fault-tolerance (data plane)	Netcore compiler
SmaRtLight	fault-tolerant (master/slave controllers)	Floodlight and NOX
Coronet	fault tolerance (link/switch)	NOX
LegoSDN	tolerate SDN application failures	FloodLight
Ravana	fault tolerant for controller/switch	Ryu
AFRO	automatic failure recovery	POX
Pyretic	language to flow policy	POX

Although existing Byzantine Fault Tolerance (BFT) solutions are suited for the file system environment, there are some problems to apply BFT to the SDN network. However, there are some existing BFT solutions that are suited for the file system environment such as SDN.

- **BFT-SMaRT** Authors in [99] propose a framework which is based to tolerate malicious faults in both control and data plane. They implement two BFT controllers state machine BFT. The first controller is based on SimpleBFT:OpenFlowJ with state machine BFT-SMaRt (a state of the art tool for creating BFT [100]) and BeaconBFT: Beacon with BFT-SMaRt. Their paradigm is based on existing controller suites instead of developing entirely new paradigms for constructing SDN controllers, as FatTire and Coronet. However, the use of a service proxy, in which related information is gathered, considered to be replaced by enabling this functionality in the switches but hardware integration presents a significant roadblock moving forward. As future work, they intend to increase performance of controller Speculative BFT and integrate BFT in the switch and not in a proxy since hardware integration presents a significant roadblock moving forward.
- **BFT in Cloud** Authors in [101] propose a Byzantine-resilient secure SDN with multiple controllers in cloud. In their contribution, a secure SDN architecture in which every switch is controlled by multiple controllers are proposed. Their proposed framework is based on NOX controller simulated in Mininet.
- Other fault tolerant mechanisms as presented by [94]: (i) Distributed SDN control with consistent reliable storage (ONIX and ONOS apply replicate durable states primary slave), (ii) Distributed SDN control with state machine replication (HyperFlow which is based in NOX) using publish/subscribe paradigm among the controllers, (iii) Distributed SDN with weaker ordering requirements as described in BGP routing (iv) Traditional fault-tolerance techniques: Viewstamped Replication (VSR), Paxos and raft, (v) TCP fault-tolerant mechanisms: have huge overhead, (vi) Virtual Machine (VM) fault tolerance Remus and Kemari and (vii) Observational indistinguishably (Lime)

2.5 Service Provisioning and Chaining

SDN-based deployments introduce great potential for innovation in the network usage, high speed and agile service provisioning, as well as enhanced network flexibility and holistic management. The combination of the global network-wide view and the network programmability supports the process of harvesting intelligence from existing security devices such as Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), followed by analysis and centralised reprogramming of the network. This approach can render the SDN more robust to malicious attacks than traditional networks [102].

2.5.1 Security Service Functions

Security services are a prime example of traditional network service functions that can benefit from the adoption of SFC, especially in the context of SDN networks. Indeed, security functions such as ACL, Segment, Edge and Application Firewalls, IDS/IPS and Deep Packet Inspection (DPI) are some of the principal service functions considered by IETF when presenting SFC use cases pertaining to Data Centers [103] and Mobile Networks [104]. Said IETF studies consider several use cases and highlight the numerous drawbacks of using traditional service provision methods when applying, among others, the security functions. The security services are typically deployed as monolithic platforms (often hardware-based), installed at fixed locations inside and/or at the edge of trust domains, and are deployed rigid and static, often lacking automatic reconfiguration and customisation capabilities. This approach, combined with the typical networks' architectural restrictions mentioned above, increase operational complexity, prohibit dynamic updates and impose significant (and often unnecessary) performance overheads, as each network packet must be processed by a series of predefined service functions, even when these are redundant [105].

The most common security service functions are detailed below:

- **IDS/IPS** is a service able to monitor traffic or system activities for suspicious activities or attack violations and is also able to prevent malicious attacks if needed (in the case of IPS).
- **Firewall** is a service or appliance running within a virtualised environment providing packet filtering. Legacy firewalls (e.g. actual hardware appliances) are also supported and can easily be integrated into the architecture.
- **Honeynet** is formed by a set of functions (Honeypots), emulating a production network deployment, able to attract and detect attacks, acting as a decoy or dummy target.
- **DPI** is a function for advanced packet filtering (data and header) running at the application layer of OSI reference model. In DPI packet payloads are matched against a set of predefined patterns.
- **Network Virtualisation**, via the use of Virtual Local Area Network (VLAN) [106]), a VLAN-like encapsulation technique to encapsulate MAC-based OSI layer 2 Ethernet frames within layer 4 UDP packets, brings the scalability and isolation benefits needed in virtualised computing environments.
- **ACLs** are used at the entry of network domain to route traffic to the appropriate isolated virtual networks and the corresponding security service functions.

Table 2.2 Examples of Network Security Service Functions

Appliance	Examples
Intrusion Detection System	Snort ¹ , Bro ²
Antivirus/Anti-SPAM	ClamAV ³
L7 Firewall	Linux L7-filter ⁴ , ModSecurity ⁵
L7 Load Balancer	F5 ⁶ and A10 ⁷
Leakage/Data Loss - Prevention System	Checkpoint DLP ⁸
Network Analyser/Classifier	Qosmos ⁹
Traffic Shaper/WAN optimisation	Blue Coat PacketShapper ¹⁰

- **Packet inspectors** to detect malformed packets or malicious activity (IPFiX, distributed DoS).
- **Secure communication protocols** with packet encapsulation services (e.g. IPSec).
- Instead of the service functions used, other than the ones employed above, other Service Functions could be included in a real deployment, such as HTTP header enrichment functions, TCP optimisers, Resource Signalling, etc.

A typical example of an important, and also ubiquitous, security-related function is DPI, whereby packet payloads are matched against a set of predefined patterns. DPI imposes a significant performance overhead, because of the pattern matching mechanisms that is a core operation and thus largely unavoidable (motivating a wealth of research efforts focusing on improving their performance [107, 108]). Nevertheless, DPI, in either form, is part of many network (hardware or software) appliances and middleboxes; some examples can be seen in Table 2.2 [109]. As Bremner-Barr et al. [109] have demonstrated, extracting the DPI functionality and providing it as a common service function to various applications (combining and matching DPI patterns from different sources) can result in significant performance gains; their benchmarks, involving a single Snort-based IDS service function, run in Mininet over OpenFlow to emulate an SDN deployment, compared to two separate traditional instances of Snort, showed that the former (i.e. the single DPI service function) performed 67%-86% faster than the latter.

Apart from the specific security protocols, security functions can be used to guarantee specific network S&D properties. Table 2.3 presents the S&D properties as satisfied by the security network functions.

Table 2.3 Security Properties and Functions

Properties	Security			Dependability	
Function	Access Control	Confidentiality	integrity	Availability	Reliability
Firewall	o			o	o
IDS/IDS		o		o	o
DPI			o	o	
IPSec	o	o	o		
Load-balancer				o	o
Honey-Pot	o	o		o	

2.5.2 Service Function Chaining

SFC provides the ability to define an ordered list of network services to create a service chain [110] on network topologies. Different factors and constraints should be considered in order to define suitable SFCs related to Service Function Instance (SFIs) and Service Function Links (SFLs) especially for dynamic instantiation such as:

- Topological constraints due to SFI location
- Resource capacity Service Node that holds SFI
- Energy consumption of a service node that holds SFI
- Utilisation rate of a SFI
- Throughput constraint or demand constraint of a SFI

¹<http://www.snort.org>

²<http://bro-ids.org>

³<http://www.clamav.net/>

⁴<https://sourceforge.net/projects/l7-filter/>

⁵<https://www.modsecurity.org/>

⁶<https://f5.com/products/modules/local-traffic-manager>

⁷http://www.a10networks.com/products/axseries-aflex_advanced_scripting.php

⁸<http://www.checkpoint.com/products/dlp-software-blade/>

⁹<http://www.qosmos.com/products/technology-overview/>

¹⁰<https://www.bluecoat.com/>

- Bandwidth allocation constraint and Link capacity of a SFL
- Latency and delay of a SFL
- Lack of availability due to failures or attacks on an instance SFI or a SFL

SFC aims to address these issues via a service-specific overlay that creates a service-oriented topology, on top of the existing network topology, thus providing service function interoperability [110]. An SDN-based SFC Architecture, such as the one defined by the ONF [111], can extend this concept by exploiting the flexibility and advanced capabilities of SDN, to provide novel and comprehensive solutions for the aforementioned weaknesses of the legacy networks.

The concept of SFC has shown promising results in enabling the faster time-to-market for the new services in the domain of telecom operators. These services are *stitched* together in the network to create a service chain. IETF, and the SFC Working Group [112] in particular, built on top of the work of the ONF for the standardisation of SFC adopting and extending features from relevant research efforts, where needed. Moreover, special care is given to the security of the SFC mechanisms. This can be done by guaranteeing the integrity of SFC-related data added to the packets for identifying the service functions chains and by ensuring that no sensitive SFC data (and the associated metadata), crosses different SFC domains, or legacy networks (non-SDN), unprotected.

2.5.3 SFC Related Research

Several SFC-related research efforts can be identified in the literature. Nevertheless, a previous survey on the use of SFC [113] reveals a lack of work focusing on security-related applications, and this is a gap that the framework presented in Chapter can cover. In terms of the key technological building blocks, Network Service Headers (NSH) [114] is an approach that involves the introduction of SFC-specific 4-byte headers which include all the information needed (including associated metadata) to reach a policy decision with regard to what service chain the traffic should follow. As part of the relevant IETF efforts, the NSH approach has been extended to define a new service plane protocol (a dedicated service plane) for the creation of dynamic service chains [115].

StEERING [116] is an OpenFlow-based alternative that allows for per-subscriber and per-traffic type/application traffic routing to the various service functions. This can be done via simple policies propagated from a centralised control point, but does not consider the security-based classification that forms the basis of the work presented here. Researchers

have also introduced **SIMPLE** [117], a policy enforcement layer that focuses on middleware-specific traffic steering and considers the inclusion of legacy service instances into the chain. It is based on monitoring and correlating packet headers before and after they traverse a specific service function, though this leads to a rather complex process (collecting packets for correlation, matching packets with high accuracy etc.).

The chaining of VNFs is another aspect examined in the literature, which considers the trend of virtualising networks and network functions in modern networks. More specifically, ETSI proposes a security management and monitoring specification in NFV that enable active and passive monitoring of the VNF and the SFC as provisioned in the NFV environment [118]. From this perspective, Megraghdam et al. [119] present a formal model for specifying VNF chains and propose a context-free language for denoting VNF compositions. Blendin et al. [120], exploit Linux namespaces to create isolated service instances per service chain, allowing one-to-one mapping of users to service instances; nevertheless, such an approach is not necessary in industrial environments, where, typically, the number of users is limited, and the management of multiple service instances can have a significant administrative burden.

Leveraging the benefits of SDN-based SFC deployments involves reversing this trend for monolithic, *all-in-one* common security services. This is an approach, brought forward in part because of the advancements in hardware performance, which meant that a single, relatively affordable, hardware platform had enough resources to accomplish multiple tasks simultaneously. In the context of SFC, the focus is on breaking-up these complex services into dedicated service functions, each providing a single task. This shift is not dissimilar to the emergence of the Microservices [121] as described in [122], software architectural style (i.e. the Microservices Software Architecture, MSA), which moves developers away from the once-dominant paradigm of building entire applications as a monolith (again, leveraging the benefits of more capable hardware - and mature, sophisticated programming tools), towards applications made up from many smaller services (elastic, resilient, composable, minimal and complete [123]), each of them performing a single function (adopting the *Do one thing and do it well* philosophy).

2.6 Design Patterns

Driven from software development methodology, Model-Driven Engineering (MDE) [124] can be used to analyse certain aspects of models, synthesise various types of artefacts and design secure and dependable systems. The concept of MDE not only can be used for software development but also for designing applications related to networks. Authors in [125] present an MDE framework for architecting wireless networks. Based on the MDE

paradigm, authors also in [126] propose the concept of Model-Driven Networking for the development of SDN applications base on Domain-Specific Language (DSML). MDE applies design patterns [127, 128] as solutions for reusable designs and interactions of objects by the use of formal proven properties [129]. Based on this concept, the design of a system has been simplified and facilitated by the modelling of design patterns.

The development of design patterns may benefit from existing implementations of software patterns as described in the literature in a variety of works [130–134]. Furthermore, the concept of component-based architecture composition is mainly applied on software components and service oriented architecture but it can be also used successfully for designing networks [135, 136]. Since network topologies are mainly related to the type of component composition and flows, they can be described adequately by workflows of process executions patterns [137]. Workflow patterns have been used for different applications such as for QoS aggregation for web service composition as proposed in [138]. Security workflow patterns for service compositions, based on enabling reasoning engines such as Drools, are also described in [139, 140].

With the softwarisation of networks in SDN, design patterns can be applied in all the different layers of SDN architectures. In general, design patterns have been also proposed for the graph algorithms in works such as [141] and [142]. More specifically in the data plane, the construction of network topologies includes the definition of network and traffic patterns in SDN as presented in works such as in [143]. In the application plane, design patterns can be used in northbound interface using RESTful API as proposed in [144]. Furthermore, SFC [110] aims to provide E2E security in SDN following security function compositions. Finally, the concept of intent-based engineering in SDN appears to enforce security policies [145].

Design pattern can be expressed in a number of different representative modes. The Web ontology language (OWL) is a family of knowledge representation languages for authoring ontologies able to express patterns as a set of constraints and requirements [146]. In addition, in **PROLOG** [147], design patterns go under various names: skeletons and techniques, cliches, program schemata, and logic description schemata. An alternative to design patterns is higher order programming [148]. The **Rete** algorithm is widely used to implement matching functionality within pattern-matching engines that exploit a match-resolve-act cycle to support forward chaining and inferencing [149]. Finally, **Drools** Business Rules Management System [27] is a rule engine to enable reasoning and support design patterns based on the predefined supported language.

Fault tolerance patterns have been proposed in works such as [150] and [151]. Although authors in [152] and [153] present fault tolerant patterns for software, the proposed archi-

tectural, detection, mitigation and recovery patterns can be also applied on SDN networks. Flow policy patterns as expressed by Frenetic languages such as Pyretic, can generate flow rules able to be installed in programmable switches of SDN networks [154].

Design patterns have been also proposed for SFC in virtualised network infrastructures [155]. In [156], authors present efficient patterns for SFC within NFVI. The key points of this work includes the definition of Language Requirements. In addition, patterns are presented to increase network performance by minimising network performance. Patterns can be also used to minimise switching and CPU resource required to implement each chain to (i) minimise latency by shorting the the number of physical hops and the number of queues each packet traversing a service chain, (ii) to minimise CPU processing by avoiding unnecessary software switching.

Another approach for efficient provisioning of security SFC using security patterns can be found in [155]. The problem statement of this work includes the (i) capture the security-related best practices for the deployment of network security functions; (ii) optimise security provisioning costs by simultaneously (a) placing the VNFs in the optimal locations and (b) finding the optimal routing paths through the proper VNF sequence for each traffic request, while respecting the capacity constraints and the security deployment constraints derived from security-related best practices; (c) scale to large-sized cloud computing data centres.

2.7 Open Issues Addressed by this Work

According to the literature review, a number of open issues exist in all layers of generic and SDN/NFV-enabled networks. This is related to the maturity level of developed solutions as introduced in the SDN and the NFV environments. The existing mechanisms propose partial solutions for the design and verification of S&D networks. More specifically, the following beyond the state of the art solutions are proposed and developed by this work:

- Existing developed mechanism do not provide efficient mechanisms for the design of SDN/NFV network topologies. This work proposes a pattern framework for developing compositional structures of SDN/NFV-enabled infrastructures.
- S&D patterns set necessary and sufficient conditions not only for composing different components in ways that guarantee S&D properties but also for ensuring network designs will use pattern framework in ways that guarantee such properties.
- Introducing an autonomic deployment of VNFs based on the defined SFC patterns and assigning physical resources to overlay virtual entities taking into account the context of the data and services that request access to those entities.

- Existing data plane fault tolerance solutions do not provide the required open and flexible design as our proposed approach.
- Workflows patterns have been used in our approach not for service composition but in an innovative way for creating network topologies based on executable patterns.
- The concept of SFC is extended in this work by creating dynamic security chains, following a backward chaining and enforcing security policies.
- Finally, in the proposed schema, it is possible to create a complete, open and extensible solution to address all the described open issues. This can also involve and introduce new concepts such as intent-based networking and artificial intelligence based on reasoning and learning suitable under development concepts of 5G networking.

2.8 Summary

In this chapter, a detailed overview of the most critical security and privacy challenges was presented. Network security appeared as a major concept especially in SDN/NFV-enabled networks. A definition of the most crucial security and dependability attributes was also given. Moreover, the most severe S&D generic, radio access and SDN/NFV threats were identified. Based on this research, a number of different countermeasures to mitigate the described threats were also presented. Furthermore, the background research on the security service functions and service function chaining were also described. In addition, different pattern-based approaches and model driven designs for secure and dependable were also presented. Finally, a description of the identified open issues and how are addressed and managed by this work were also provided.

Chapter 3

Definition of the Pattern Schema and Language

3.1 Overview

This chapter presents the pattern schema. More specifically, the pattern schema includes the definition of the pattern, containing the different aspects that covers the topology and the requirements of the proposed patterns. Based on the pattern definition, the specification of each pattern is provided. In addition, for the pattern composition and decomposition, forward and backward reasoning is also described to enable inference. Furthermore, the semantics of the developed pattern language and the developed class diagram that includes the interaction between the components is also presented. Finally, the presentation of the deployed pattern engine is also introduced.

3.2 Pattern Definition

The main focus of network design relies on specification analysis, design, verification, and validation of systems that include hardware/software, data, procedures, and facilities. In order to design network architectures with respect to S&D, a model-based approach can be used adequately. This model approach is based on patterns to design and validate network architectures as component compositions. Design patterns can give solutions to problems by the use of formal proven properties or by testing. These patterns should be able to define compositions of complex network topologies to guarantee the required S&D property at design level. Based on these patterns a designer will be able to construct SDN/NFV-enabled architectures without the need to prove previously verified network properties. Once proven,

relations between pattern component properties can be expressed as production rules to enable reasoning.

The compositions defined by patterns can be both vertical and horizontal, i.e., they can involve components at the same (horizontal) or different (vertical) layer in the reference architecture. To do so, design patterns should encode abstract and generic component interaction and orchestration protocols, enhanced (if necessary) by transformations to ensure the semantic compatibility of data or system functionality of the components that are (or need to be) composed. Furthermore, the component interaction and orchestration protocols encoded by the patterns must have an evidenced ability (i.e., an ability proven through formal verification or demonstrated through testing and/or operational monitoring) to achieve a semantically viable interoperability between their components.

Network patterns can be used as an instrument for designing, verifying and modifying the topology of networks. More specifically, the pattern can define generic ways of composing (i.e., establishing the connectivity between) and configuring the different and heterogeneous components that may exist at all layers of the implementation stack of network. The patterns will enable orchestrations that preserve the basic security properties of confidentiality, integrity and availability, as well as privacy, dependability and interoperability properties able to guarantee different states. More specifically, patterns can be applied as described below:

- **At design** time, the procedure includes the definition of a design problem and the required S&D property that must be guaranteed by the pattern. This can include the creation of network infrastructures or the discovery of available paths to assigns flow rules which are able to guarantee the required property.
- **In verification**, an existing network design (topology) and the required S&D properties are provided. The pattern is applied to analyse the former and ensure that the latter property is satisfied. The analysis is based on checking if the topology of the pattern matches totally or partly the network design and on whether the individual components that constitute the network with the particular topology have certain properties that can guarantee end-to-end network level S&D properties.
- Finally, **at runtime** when a property is violated, patterns are applied at runtime to alter the topology, the components, the paths or the forwarding rules of an operational network in order to ensure the satisfaction of S&D properties.

Patterns can be used to design infrastructures able to guarantee the S&D properties. These patterns can be used for the discovery of component compositions with verified properties.

The pattern specifies the order of component compositions constituting a primitive component orchestration. The main target of a pattern can be to find suitable component compositions in order to guarantee the required property. More precisely, the defined patterns are able to validate system properties and in case that the property is not guaranteed, the pattern will have to substitute components with other atomic ones or compositions in order to guarantee system properties. Based on above, the pattern specification schema is defined as follows:

Definition 1. *A pattern schema is an abstract structure for specifying design patterns which includes: (a) an abstract network topology, defining the control structure and data flows of the components, (b) functional requirements that should be satisfied by the components of the network that are composed according to the structure of (a), (c) the required S&D property that the pattern guarantees and (d) an execution pattern rule.*

The constituents (a)–(d) of the S&D pattern schema are discussed in more detail below. In the following sections, the description of the pattern schema is given including also the specification and the language of patterns.

3.3 Pattern Topology

The representation of networks as a constitution of physical and cyber component compositions and flows is essential for the model-based design. Component-based engineering can be applied for the composition of physical and cyber subsystems of networks. The main idea is that the composition of subsystems can also compose new systems enhancing their inputs, outputs, properties and attributes. Furthermore, the composition of components can create composition from atomic or from other compositions. For instance, the composition of two atomic components c_1 and c_2 can be defined as

$$c_1 \circ c_2 = c \quad (3.1)$$

More specifically, in the context of component composition, the following rule can be applied:

$$Comp(x, z) \circ Comp(z, y) \rightarrow Comp(x, y) \quad (3.2)$$

where x, z, y are the inputs and outputs of respective components c_1, c_2 .

On the other hand, the decomposition of a component composition can lead to atomic components:

$$c = c_1 \circ c_2 \quad (3.3)$$

For the component decomposition, when the following rule can be applied:

$$Comp(x, y) \rightarrow Comp(x, z) \circ Comp(z, y) \quad (3.4)$$

The composition of two components will perform as action a , the composition of two actions $a_1 \circ a_2$. On the other hand, the substitution of a system c with a composition of two objects c_1 and c_2 will invoke and save the same actions. In addition to that, a component c'_1 can substitute c_1 , if it provides and uses what the latter provides. The generic substitution approach is that components can be replaced by compositions able to perform the same actions. Based on that, the below action compositions can be defined:

$$a_1 \circ a_2 = a \quad (3.5)$$

Let c is a component which utilises an action a , to substitute this by a composition of c_1 and c_2 , the $a \subseteq a_1 \circ a_2$ should be guaranteed:

Definition 2. *Let c_1 and c_2 be components performing activities a_1 and a_2 , the component composition c will be equal to $c = c_1 \circ c_2$, with activity $a = a_1 \cap a_2$ for the serial composition and $a = a_1 \cup a_2$ for the parallel composition.*

3.4 Pattern Requirements and Properties of Compositions

The definition of compositions includes also a set of functional properties and non-functional properties and constraints expressed also as requirements that should be satisfied by the individual network components composed by the pattern and/or the component composition as a whole. To achieve this objective, **design patterns** are able to design physical and logical architecture so the following definition can be given:

$$\mathcal{R}(\mathcal{P}(c)) \quad (3.6)$$

where \mathcal{R} is the requirement of the composition c , c can express either a topology structure, components composition with source src and destination dst in case of a path and \mathcal{P} is the required property. As properties, the approach can be split to functional and non-functional requirements as described in the next subsections.

3.4.1 Composition Properties

The properties satisfied by components may be different from each other and different from the properties that the composition satisfies. In *component composition*, when c_1, \dots, c_n are

components that satisfy a property \mathcal{P} , then the composition $c = c_1 \circ c_2 \dots \circ c_n$ formed of these components should satisfy \mathcal{P} .

$$\mathcal{P}(c_1) \circ \dots \circ \mathcal{P}(c_n) \rightarrow \mathcal{P}(c) \quad (3.7)$$

More specifically, as an example let's consider a sequential composition of two components: $c_1 \circ c_2 \rightarrow c$. It exists when the following implication can be proved:

$$\text{When } (c_1 \text{ satisfies } \mathcal{P}) \text{ and } (c_2 \text{ satisfies } \mathcal{P}) \rightarrow (c \text{ satisfies } \mathcal{P})$$

On the other hand, component decomposition appears to be critical for system designs with respect to required properties. In *component decomposition*, when c is required to satisfy a property \mathcal{P} , then suitable components c_1, \dots, c_n should be found to satisfy $\mathcal{R}(\mathcal{P}(c_1), \dots, \mathcal{R}(\mathcal{P}(c_n)))$:

$$\mathcal{R}(\mathcal{P}(c)) \rightarrow \mathcal{R}(\mathcal{P}(c_1)) \text{ and } \dots \text{ and } \mathcal{R}(\mathcal{P}(c_n)) \quad (3.8)$$

As an example, let's consider a sequential decomposition of two components: $c \rightarrow c_1 \circ c_2$. When a required property should be guaranteed by the c , the subcomponents c_1 and c_2 should satisfy the condition:

$$\mathcal{R}(\mathcal{P}(c)) \rightarrow \mathcal{R}(\mathcal{P}(c_1)) \text{ and } \mathcal{R}(\mathcal{P}(c_2))$$

or

$$\mathcal{R}(\mathcal{P}(c)) \rightarrow \mathcal{R}(\mathcal{P}(c_1 \circ c_2))$$

If there are no atomic components to guarantee the required property, a recursive procedure is used in which successive (sub-) compositions are being generated until the atomic components bound to them satisfy the required properties. The decomposition can be analysed as follows:

$$\mathcal{R}(\mathcal{P}(c)) \rightarrow \mathcal{R}(\mathcal{P}(c_1)) \text{ and } \mathcal{R}(\mathcal{P}(c_2)) \rightarrow$$

$$(\mathcal{R}(\mathcal{P}(c_{1_1}) \circ \mathcal{R}(\mathcal{P}(c_{1_2}))) \text{ and } (\mathcal{R}(\mathcal{P}(c_{2_1}) \circ \mathcal{R}(\mathcal{P}(c_{2_2})))$$

$$\rightarrow \dots \rightarrow \text{until components } c_{1_1}, c_{1_2}, c_{2_1}, c_{2_2}$$

that satisfy the required property \mathcal{P} are found

Considering the components composition, Figure 3.1 depicts a stepwise decomposition of components based on sequence and parallel-split workflow patterns.

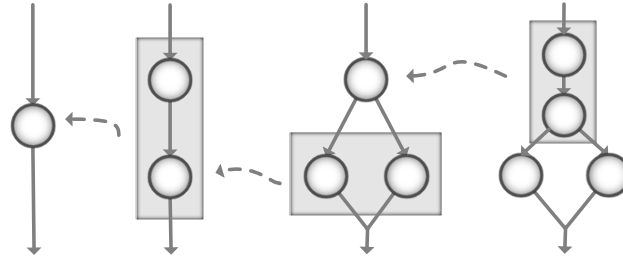


Fig. 3.1 Stepwise Decomposition

3.4.2 Functional Requirements

Functional Requirements characterise the functionalities and features of a system. Moreover, functional requirements state what a system should define what a system must do and how it reacts on specific inputs or situations. The list of functional requirements include a number of different technical details, data manipulation and processing and some constraints such as connectivity, coverage, scalability, resource management, activities and other functionalities.

3.4.3 Non-Functional Requirements

Non-functional Requirements describe the qualities of a systems. Network analysis includes whether non-functional requirements such as S&D properties including the confidentiality, integrity, availability, reliability, safety, maintainability together with QoS properties are preserved. The conditions depend on the respective non-functional requirement that networks guarantee. The satisfaction of a non-functional property can be defined by a boolean value (i.e. encryption enabled/disabled), an arithmetic measure (i.e. delay, encryption level) or a probability measure (i.e. reliability/uptime availability).

The S&D properties are also related to the components and the topology which are included in this network. However, it should be noted that the composition of two components which preserve a S&D property does not necessarily guarantee that the composition will preserve the same property. In addition, if a composition guarantees the conditions of a S&D property, the atomic components may not preserve the property. If those conditions are satisfied, the property will be guaranteed at the communication link. However in networks, it is important that properties are also guaranteed on the communication medium. Attacks on

wireless medium can also cause an attack on a system component. Since, a medium such as a wireless link cannot be modified, the security property should be satisfied both the output of the source node and at the input of the destination node.

3.5 Pattern Specification

The pattern specification is responsible to provide the required structure and meaning of the proposed pattern as can be defined bellow:

- i) **Name:** include a clear statement of the role and the definition of the pattern.
- ii) **Problem:** analyse the problem that the pattern should solve.
- iii) **Existing solutions:** describe the current state of the art solutions for the problem.
- iv) **Our solution:** describe the pattern, emphasising the key technical points of solution.
- v) **Evaluation:** experimentally or formally of the pattern.
- vi) **Contributions:** define the contributions and advances of the proposed pattern over state of the art existing solutions.

The template of each pattern is defined as follows:

- Case Study 1: <name>
- Problem
- Existing solutions
- Our solution (key technical points of solution)
 - Pattern X which ...
- Contributions (wrt SOTA)
 - C1
 - C2

3.6 Pattern Composition and Reasoning

Design patterns can be applied using recursively forward or backward chaining respectively enabling reasoning for an inference engine. Inference provides the required reasoning steps to move from premises to conclusions [157]. The inference engines match rules and facts in the Knowledge Base (KB). Backward and forward chaining are methods based on inference rules. Inference rules can be used to express the properties such as correctness, soundness, completeness, incompleteness and timeliness. Forward chaining is useful in verification and backward chaining can be used at design.

Algorithm 1 Forward Chaining

```

1: procedure FORWARD CHAINING
2:   newFact  $\leftarrow$  False
3:   for all Facts in Fact Base do
4:     if all premises match fact-base then
5:       for all Rules in Rule Base do
6:         if fact not in fact-base then
7:           add Fact to Fact Base
8:           newFact = True
9:         end if
10:      end for
11:    end if
12:  end for
13: end procedure

```

3.6.1 Forward Chaining

Forward chaining is a data-driven inferencing where the conditions proceed to the conclusion of the goal. When the premises of a rules are satisfied the rule infer the conclusion. Forward chaining systems usually employ a Breadth-First Search (BFS) strategy. BFS is a method for finding connected components of nodes in graph through the level by level node visit. The schema for Forward Chaining from Gonzalez and Dankel [158] is presented in Figure 3.2.

Syntax : < IF > conditions < THEN > conclusion < . >

The forward reasoning Algorithm 1, supports four different steps:

- **Match** premises of each rule with facts

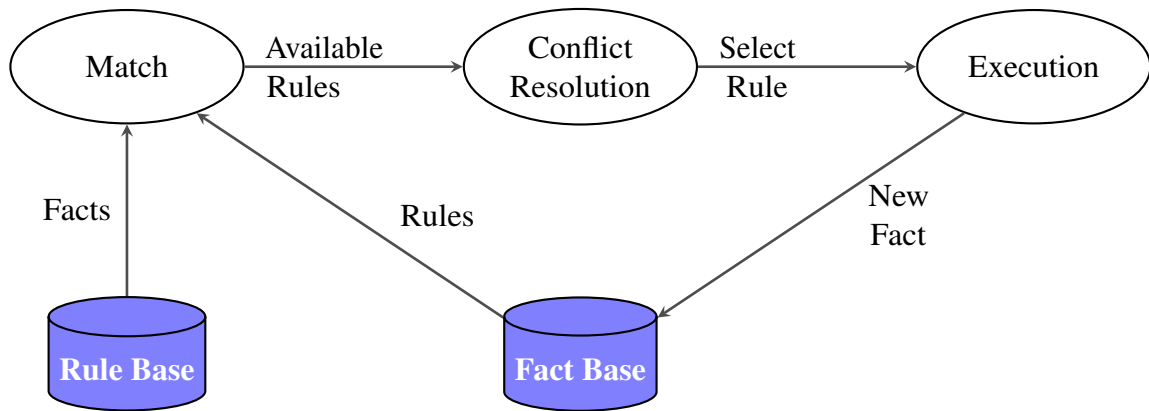


Fig. 3.2 Forward Chaining Schema

- **Conflict Resolution** Choose the rule based on its priority and ignore rules with known conclusions
- **Execution** the rule and the result as a fact in the fact base
- **End Condition** define the end of the procedure since there is no rule without results

In forward chaining, query is entailed if it maps by homomorphism to the enriched facts [159].

Algorithm 2 Backward Chaining

```

1: procedure BACKWARD CHAINING
2:   Input: Knowledge base: including facts and rule
3:   Outputs:
4:   goals Knowledge base
5:   goal isSatisfied  $\leftarrow$  False
6:   for all Facts in Fact Base do
7:     if all premises match fact-base then
8:       for all Rules in Rule Base do
9:         if fact not in fact-base then
10:          add Fact to Fact Base
11:          newFact = True
12:        end if
13:      end for
14:    end if
15:  end for
16: end procedure
  
```

3.6.2 Backward Chaining

Backward chaining is a goal-driven inferencing that provides a backward search from goal to the conditions used to satisfy this goal. Backward chaining tests if an hypothesis is true. The main idea of the backward chaining is based on goal reduction to prove the fact that appears in the conclusion and proves the premises of rule recursively. Backward chaining systems usually employ a Depth-First Search (DFS) strategy. DFS is a technique for solving graph problems such as finding connected components of directed or undirected graphs [160]. DFS can be used as an algorithm to search or traverse data structures of graphs or trees. The algorithm visits the nodes of the graph until to reach a leaf or a node without having non visited nodes. It starts at the root following a children sub tree. Backward chaining proceeds in the reverse manner compared to forward chaining because it applies the rules to rewrite the query in several ways and the initial query is entailed to the initial facts [159]. The forward reasoning is expressed in Algorithm 2.

$$\text{Syntax : conclusion} < IF > \text{conditions} < . >$$

A comparison of forward and backward chaining is provided in [161] and the most critical points are presented in table 3.1.

3.7 Pattern Components

Before the construction of the pattern language, a number of different components are required to be defined. The background knowledge and component definitions are provided in the next subsections.

3.7.1 Network Graphs

To express architectural topologies of networks including spanning tree component compositions as an integration of components and flows, the basics of graph theory approach can be used adequately [162, 163].

Definition 3. A graph $G=(V,E)$ is a directed graph that contains a set of vertices V and a set of edges E as ordered pairs of vertices so $V = \{u_1, u_2, \dots, u_i\}$ and $E = \{(u_1, u_2), (u_2, u_3), \dots, (u_{i-1}, u_i)\}$ then $E \subseteq V \times V$.

In Algorithm 3, the expression of forward chaining in graphs is described.

The generic graph theory approach can be defined to satisfy network topologies including physical (or virtual) network components such as nodes and links. Therefore, the term of

Table 3.1 Comparison Between Forward and Backward Chaining

Forward Reasoning	Backward Reasoning
Starts with the initial facts	Starts with some hypothesis or goal
Asks many questions	Asks few questions
Tests all the rules	Tests some rules
Slow, because it tests all the rules	Fast, because it tests less rules
Provides a huge amount of information from a small amount of data	Provides a small amount of information from just a small amount of data
Attempts to infer everything possible from the available information	Searches only that part of the knowledge base that is relevant to the current problem
Primarily data-driven	Goal-driven
Uses input; searches rules for answer	Begins with a hypothesis; seeks information until the hypothesis is accepted or rejected
Top-down reasoning	Bottom-up reasoning
Works forward to find conclusions from facts	Works backward to find facts that support the hypothesis
Tends to be breadth-first	Tends to be depth-first
Suitable for problems that start from data collection, e.g. planning, monitoring, control	Suitable for problems that start from a hypothesis, e.g. diagnosis
Non-focused because it infers all conclusions, may answer unrelated questions	Focused because all questions aim to prove the goal and search as only the part of KB that is related to the problem
Explanation not facilitated	Explanation facilitated
All data is available	Data must be acquired interactively (i.e. on demand)
A small number of initial states but a high number of conclusions	A small number of initial goals and a large number of rules match the facts
Forming a goal is difficult	Easy to form a goal

$G = (N, L)$ can be used instead of $G = (V, E)$, where N is a set of *nodes* that will be used instead of the term *vertices* and L is a set of *links* that will be used instead of the term *edges*. The following terms can be used to define the properties of network graphs.

- **Nodes** can be defined as $N = \{n_1, n_2, \dots\}$. Other definition of source can be *src* or *s* and *dst* or *t* as the destination.

Algorithm 3 Forward Chaining in Network Graphs

```

1: procedure FORWARD CHAINING IN GRAPHS
  Input: Graph  $G = (V, E)$ , directed or undirected with  $v \in V$  and  $e \in E : V \times V$ 
  Output: Path
2:   for all  $v \in V$  do
3:     while  $u \in V$  has an unvisited neighbour  $v \in V$  do
4:       for  $u \in j = 1$  to  $n - 1$  do
5:         if  $a \in M(i', j')$  then
6:            $M(i', j') = \{a\}$ 
7:         else
8:            $M(i', j') = M(i', j') - A$ 
9:         end if
10:      end for
11:    end while
12:  end for
13: end procedure

```

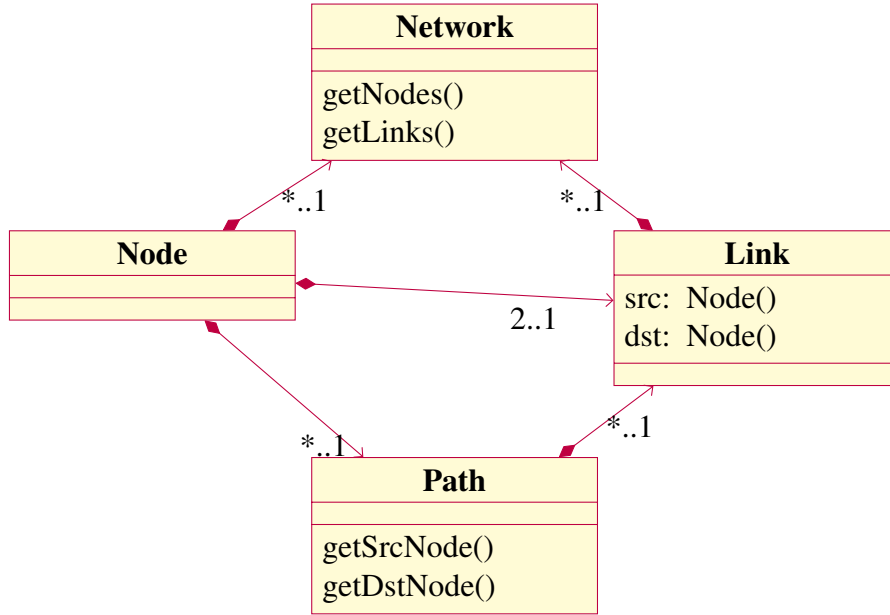


Fig. 3.3 Class Diagram of the Network Components in Graphs

- **Links** can be defined as $L = \{l_1, l_2, \dots\}$ where is an ordered pairs of nodes so $L \subseteq N \times N$:

$$L = \{l_1, l_2, \dots, l_i\} = \{\langle n_1, n_2 \rangle, \langle n_2, n_3 \rangle, \dots, \langle n_{i-1}, l_n \rangle\} \quad (3.9)$$

- **Path** between two nodes $Path(s,t)$ is a sequence of different nodes N and links L which connect source s and destination t . A path can be represented as:

$$Path(s,t) = (s, \langle s, n_1 \rangle, n_1, \langle n_1, n_2 \rangle, n_2, \dots, n_k, \dots \langle n_k, dst, \rangle, t) \quad (3.10)$$

- **Length** of a path is the sequence of $i + 1$ nodes and i consecutive links.
- **Adjacent** are two nodes $\{n_i, n_j\}$ when exists a link $l_{ij} \in L$ between them.
- **Directed** is the graph/network that each link (n_i, n_j) belonging to L has a direction that does not imply that (n_j, n_i) belongs to L , where *Links* are called *arcs*.
- **Weighted** is the graph $G = (N, L)$ when there is a value w_{ij} associated with each link l_{ij} representing a number or attribute (ie. cost, lengths, capabilities, link speed, capacity, security property etc.). The graph can be denoted as (G, W) or $G = (N, L, W)$. A weighted path based on the function can be represented as follows:

$$Path_w(s,t) = (s, \langle s, n_1 \rangle, n_1, \langle n_1, n_2 \rangle, n_2, \dots, n_k, \dots \langle n_k, t, \rangle, t) \quad (3.11)$$

- **Loop** exists when both endpoints of a link are on the same node.
- **Simple** is the graph that has not any loop and a node/link appear only once.
- **Parallel** defines the parallel links.
- **Degree** of a n node $deg(n)$ is the number of interconnected nodes (in-degree: number of nodes entering, out-degree: number of nodes leaving).
- **Weight** of a l link $weight(l)$ is the number related to the factor of the link.
- **Depth** of a node is the number of links from the root.
- **Connected** is the graph when every pair of nodes can be connected by a path.
- **Trial** is the path without repeated links.
- **Cycle or closed path** is the path that begins and ends at the same node and all links are different.
- **Tree** is a connected graph without cycles when there is only one path between two nodes.

- **Height** of a node is the number of nodes of the longest path to any node.
- **Diameter** of a network is the maximum length of any shortest path between an ingress and an outgress node.
- **Distance** of a path is the minimum number of links along a path from n_i to n_j .

3.7.2 Network Components

Network topologies include a number of different components and functions on different layers. The topologies can determine actual network infrastructures including different network elements such as end-hosts, network service functions, intermediate nodes and forwarding devices. Network components may be classified in two types the physical and the virtual ones. Especially in SDN, the term of SDN controller can be also included in the list of architectural components. More specifically, the following type of components are mainly used in this work.

- **End hosts** can be servers and clients, controllers and applications running on network infrastructures.
- **Forwarding Nodes** can include intermediate and forwarding nodes such as routers, switches (legacy of programmable), hubs, IoT gateways, wireless access points,
- **Network functions** may include Firewall, IDS, DPI, and Honeypot.
- **SDN Controllers** are responsible to manage and monitor network components of an SDN architecture.

In addition to that, regarding the network links, there are two categories the physical and the virtual. In the case of physical link, this can also include wireless or wired links such as Ethernet, coaxial or fibre-optic cables. Each of the described links can also have specific capabilities and constraints.

3.7.3 SFC Terms and Components

The basic terms and definitions of SFC architectural components and configurations are based on the IETF, SFC Architecture [110] and SFC environment Security requirements [164] as follows:

- **Service Function Chain (SFC)** defines an ordered set of abstract service functions and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of classification. The SFC is also called as Virtual Network Function Forwarding Graph (VNF-FG).
- **Service Function (SF)** is responsible for specific treatment of received packets. A set of service functions includes functions such as firewall, DPI, IDS, etc. Each function is located beside a forwarder which is able to forward the traffic to the specific function. These services may be composed of one or multiple instances. These may be the physical appliances or virtual machines running in NFVI. Service functions are key security mechanisms to be leveraged in a secure industrial infrastructure, and deployed as virtualised network service functions.
- **Service Function Instance (SFI)** denotes an instantiation of a service function.
- **Service Node (SN)** is a virtual or physical element that holds at least one service function. An SN can contain an entire SFC or a part of it.
- **Classifier** is responsible to classify incoming traffic based on the predefined flow rules of the ACLs and mark packets with the corresponding SF Chain Identifier. It can be on a data path or run as an application on top of a network controller. Classifiers are responsible for assigning traffic to the appropriate service chain, based on various criteria, such as its maliciousness or the tenant that it belongs to. It can be assumed that tenant identities have already been validated by authentication/authorisation components. Classifier is responsible to embed a header into the flow packet utilising the NSH field to facilitate the forwarding of flow packets along the service function chain path. This header also allows the transport of metadata to support various service chain related functionalities.
- **Service Function Forwarder (SFF)** is responsible for forwarding traffic to one or more connected service functions according to information carried in the SFC encapsulation, as well as handling traffic coming back from the service function (legacy or virtual). A set of service function forwarders can forward the traffic to the assigned service chain. Service Forwarders and Proxies (where needed) are responsible for steering traffic accordingly, in order to realise said Service Chains. The Service Function Forwarders steer traffic to the various Service Function Nodes. If the Service Function Nodes are not OpenFlow-speaking or SFC-aware, or are in different domains, SFC Proxies are necessary.

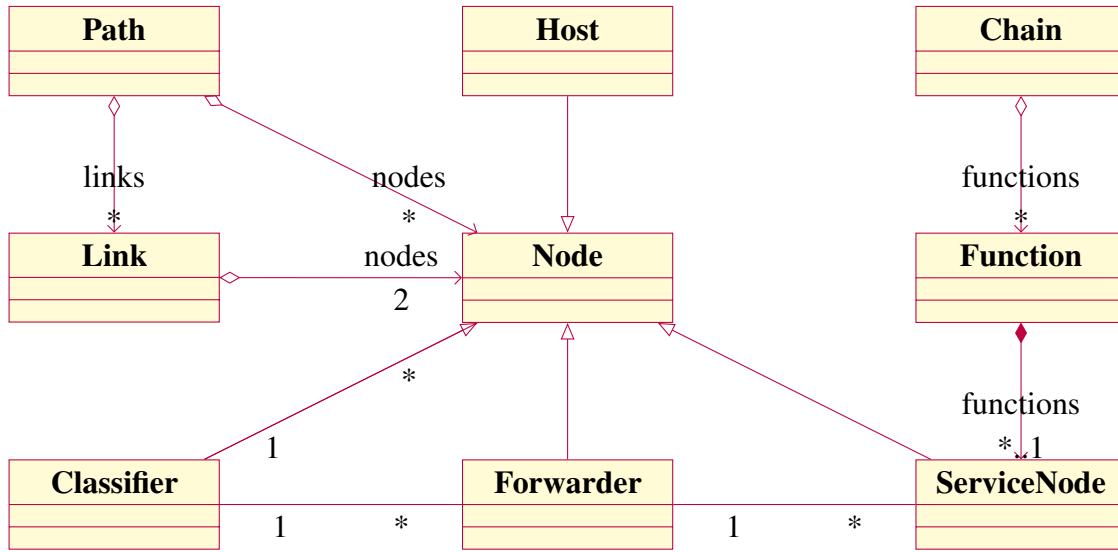


Fig. 3.4 Class Diagram of the Network Components in Service Function Chaining

- **Service Function Path (SFP)** is a constrained specification of where packets assigned to a certain route must go. Any overlay or underlay technology can be used to create service paths (VLAN, ECMP, GRE, VXLAN, etc.). SFP is related to the SFC and may provide information such as SF instances and in case this is not provided, the RSP can provide the actual path.
- **Rendered Service Path (RSP)** is the actual Service Chain that refers to the SFP and includes all the SFP and SFC information for creating the Service Chain. If SF details are not included in the SFP, path selection algorithms can be used to identify routes and to includes to programmable SFFs.
- **Tenant:** A tenant is an organisation that may use SFC on one's own private infrastructure or on an infrastructure shared with other tenants. The tenant may be using SFC to provide service to its customers or users.

3.7.4 Data Flows and Policies

Flows are considered the transport of physical quantities or the transmission of computed data on network infrastructures. More specifically, data flows include data packets and bit streams. On the other hand, physical quantities can be water and oil quantities. Especially for control flow analysis, a reverse postordering depth-first search can be used to produce natural linearisation of directed graphs.

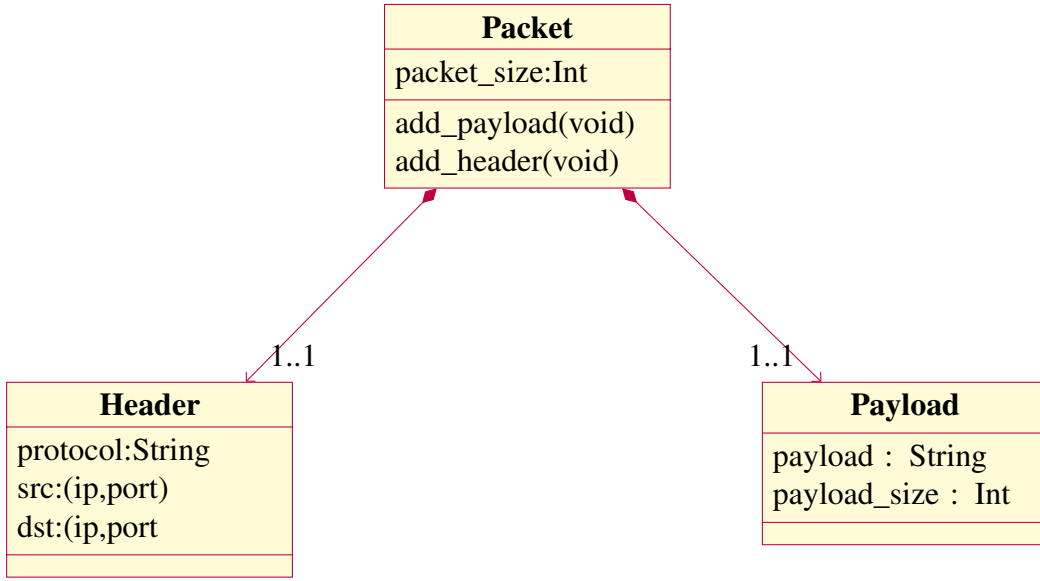


Fig. 3.5 Class Diagram of Packet Components

3.7.4.1 Network Packets

A network packet is a formatted unit of data carried by a packet switched network. Packet consists of the control information as included in the header and data in the payload as follows:

- **Packet** $p := \langle p_{hdr}, p_{pld} \rangle$.
- **Header** $p_{hdr} := \langle proto, srcIP, dstIP, srcPort, dstPort \rangle$ where $proto \in \{ip.icmp, tcp, udp\}$, $srcIp, dstIp \in IP$, $srcPort, dstPort \in (0..65535)$.
- **Payload** $p_{pld} := data\ payload$

3.7.4.2 Packet Encryption

The encryption of packet p is defined by the use of hash functions E so the $p \leftarrow E(p) = c$. The cipher c can be decrypted by the use of the decryption function D so the $c \leftarrow D(c) = p$. The key k which has specific length $length(k)$ can be used for encrypting and decrypting packets using the described respective encryption and decryption functions E_k/D_k respectively. In case of symmetric encryption the term can be expressed as E_k^S/D_k^S . In case of asymmetric encryption, the key can be expressed as k^{AB} from A to B and k^{BA} from B to A and the encryption/decryption functions as E_k^A/D_k^A .

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <flow xmlns="urn:opendaylight:flow:inventory">
3   <idle-timeout>0</idle-timeout>
4   <priority>1</priority>
5   <flow-name>flow1</flow-name>
6   <match>
7     <ipv4-source>10.0.0.2/24</ipv4-source>
8     <ipv4-destination>10.0.0.1/24</ipv4-destination>
9     <in-port>openflow1:1</in-port>
10  </match>
11  <id>1</id>
12  <table_id>0</table_id>
13  <instructions>
14    <instruction>
15      <order>1</order>
16      <apply-actions>
17        <action>
18          <output-action>
19            <output-node-connector>2</output-node-connector>
20          </output-action>
21          <order>1</order>
22        </action>
23      </apply-actions>
24    </instruction>
25  </instructions>
26 </flow>

```

Fig. 3.6 OpenFlow Rule Example

3.7.4.3 Flow Policies and Filtering

Flow rule policies can be used to classify and handle traffic and resource constraints. ACL is an example of a flow rule having a list of permissions which grant access and operations to objects such as network packets. ACLs are used to filter network traffic in the switches, routers, firewalls etc. An ACL is a list of filtering rules that are able to deny, redirect or allow traffic when packets match specific criteria such as i) source and destination IP address, MAC address and TCP/UDP port, ii) IP Protocol, iii) MAC ether type, iv) VLAN Priority, v) Differentiated Services Code Point (DSCP) and finally v) ICMP type and code.

OpenFlow is a protocol able to manage and modify flow entries in a switch forwarding table from a SDN controller . More specifically OpenFlow rules can managed by the controller to determine the network traffic path through programmable switches. Compared

to the ACL rules, OpenFlow rules enable also more sophisticated packet forwarding through specific paths and routes. The structure of the OpenFlow rule has some similarities with the ACL but it includes also some additional fields such as ingress port and output which are related to the path selection and classification. Finally, apart from the match and action capabilities of the OpenFlow protocol, there is also the capability for retrieving data flow statistics by the programmable switch. The structure of an OpenFlow rule can contain the following field as presented in Figure 3.6.

The main flow policy notations can be expressed as filtering rules applied to the properties of packets as presented below:

- **Flow Rule:** $f : \langle action : \{encrypt, bypass, drop\}, protocol, src, dst, property \rangle$
- **Set of Flow Rules:** $F = \langle f_1, f_2, \dots \rangle$
- **ACL Rule:** $f^a = \langle action : \{permit, deny, redirect\},$
 $match : \{src\{IP, MAC, Port\}, dst\{IP, MAC, Port\}, proto\{ip, icmp, tcp, udp\}, \dots \rangle$
- **OpenFlow Rule:** $f^o = \langle match : \{in port, src\{IP, MAC, Port\}, dst\{IP, MAC, Port\},$
 $vlanID, vlanPCP, ipSrc, ipDst, ipPr, ipToS\},$
 $action : \{Forward packet to port(s), encapsulate and forward to controller,$
 $drop packet, send to normal processing pipeline\} \rangle$
- **Filter Network** $G = \langle N, L, F \rangle$. where N are the Nodes, L the links and F the filters.

3.8 Pattern Language

Design patterns can be expressed as pattern rules. For that reason, a rule engine is required to support backward and forward chaining inference and verification. Drools rule engine [27] is selected to express design patterns as production rules. Enabling reasoning, driven by production rules, appears to be an efficient way to represent design patterns. It enables reasoning driven by production rules, usually used for business management, software development and service oriented architectures, but they can be also applied adequately to the design of network architectures. It supports backward and forward chaining inference and verification by applying and extending the Rete algorithm [149]. Each rule consists of two parts: the *when* conditions and the *then* actions. When the conditions in the Left Hand Side (LHS) are satisfied, then the rule is fired to execute the actions as described in its Right Hand Side (RHS). Drools inference engine is using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When such a rule

is found, the engine can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data. An example of a Drools rule can be seen in Rule 3.1. Furthermore, Drools can also support backward chaining as presented in Rule 3.2 as an inference method that can be described (in lay terms) as working backward from the goal(s).

```
1 rule "Hello World"
2   when
3     m : Message( status == Message.HELLO, myMessage : message )
4   then
5     System.out.println( myMessage );
6     m.setMessage( "Goodbye cruel world" );
7     m.setStatus( Message.GOODBYE );
8     update( m );
9   end
10
11 rule "GoodBye"
12   when
13     Message( status == Message.GOODBYE, myMessage : message )
14   then
15     System.out.println( myMessage );
16   end
```

Rule 3.1 Drools Sample

```
1 query isContainedIn( String x, String y )
2   Location( x, y; )
3   or
4   ( Location( z, y; ) and isContainedIn( x, z; ) )
5 end
6
7 rule "go1"
8   when
9     String( this == "go1" )
10    isContainedIn("office", "house"; )
11   then
12     System.out.println( "office is in the house" );
13   end
```

Rule 3.2 Backward Chaining Drools Sample

Computational models include a set of objects, rules and semantics. In order to specify and express design patterns with respect to S&D properties and constraints, the semantics of

the pattern language should be defined. As mentioned previously, different network topology ontologies such as *Nodes*, *Links* and *Flows* are included in the list of the components. The correlation between the different components are depicted in the class diagram as shown in Figure 3.7. Moreover, the *Req* represents the constraints as requirements of the topology and the required property. Therefore, different Java classes were developed to represent the different *Network Components* of the topology (nodes, links, paths and flows) and the *S&D Requirements and Properties* as needed by *Pattern Rules*. All the developed Java classes are included in the Annex of this thesis.

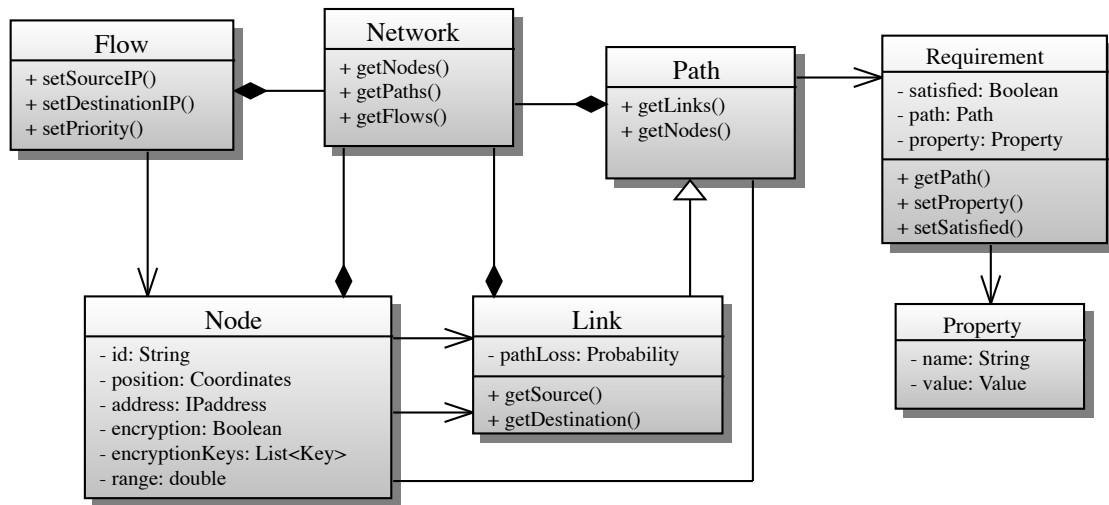


Fig. 3.7 Class Diagram of Network Components

Drools rule semantics are used to encode the topology of a pattern and the process of finding suitable component compositions in order to guarantee the required properties. The definition of the component in Drools language are based on the previous described network components. In the LHS, the network components which constitute the topology of the pattern are defined. In the RHS, new or updated compositions or atomic components can be inserted, updated, modified, executed or deleted/retracted of facts in the knowledge base which will update also the inventory list. When a network that matches the topology of a pattern does not satisfy the required property, the pattern may be used to substitute, add or remove components in order to satisfy the property. In Table 3.2 the most useful semantics are presented. Finally, the semantics of Drools language give the potentiality to represent more complex patterns by adding more variables and pattern properties.

Table 3.2 Pattern Language Semantics

Type	Syntax	Description
rule	rule "name"	name of the rule
Left Hand Side (LSH)		
when	Network Pattern Elements (Facts)	
	Node (address, ports, position, properties)	match network nodes
	Link (src, dst, weight)	match links between nodes
	Path (src, dest, links, nodes)	match paths between source and destination nodes, intermediate links
	Flow (src, dst, inPort, outPort, priority)	match flow rules between nodes
	Req (src, dst, pro, satisfied)	match requirements of pattern such as source, destination, satisfied property
	Conditional Elements	
	==	match conditions
	contains	contains object (logical)
	not	not match (logical)
	!=	not match (arithmetic)
	collect, accumulate, forall, exists	not arithmetic
Right Hand Side (RSH)		
then	Actions	
	modify (\$fact){pro=pro' }	modify knowledge base fact
	retract (\$fact)	retract knowledge base fact
	insert (new Fact ())	insert knowledge base fact
	update (\$fact)	update knowledge base fact
	Java commands	other Java language syntax

3.9 Summary

In this chapter, the pattern schema was defined. Design patterns express conditions that can guarantee specific S&D properties and can be used to design networks that have these properties and manage them during their deployment. The topology of the patterns and the component compositions preserving properties were also described. The functional and the non functional requirements as expressed in patterns are also presented. In addition, the

pattern specification was also provided. Moreover, the forwarding and backward reasoning inference mechanisms were also defined as the two main methods of reasoning when using an inference engine for expert systems, business and production rule systems. In addition, the different components as used by the proposed patterns were also included. Finally, a pattern language was proposed to design executable patterns, encoded in a rule-based reasoning system, able to guarantee S&D properties in SDN/NFV-enabled networks in order to mitigate security and dependability challenges.

Chapter 4

Secure and Dependable Design Patterns

4.1 Overview

This chapter presents the development of pattern instances supporting the design and verification of physical and virtual network topologies, based on the pattern specification schema as discussed in Chapter 3. Design patterns are proposed to design network topologies, guarantee S&D properties and finally provide service provisioning and chaining. In the next subsections, a number of different design patterns are proposed including the network topologies patterns, the path discovery pattern, the reliability and fault tolerance patterns, the security patterns and finally, the SFC patterns.

4.2 Topology Patterns

A network topology defines the arrangements of elements and can be either physical or logical. The design of network infrastructures can be based on topology patterns as described below.

4.2.1 Physical Topology Patterns

4.2.1.1 Background

A physical topology describes how the nodes are connected in the network, where are placed and what type of link exists between them. There are different physical network topologies (Figure 4.1) as described below:

- **Line** topology is used to define the connection between two endpoints.

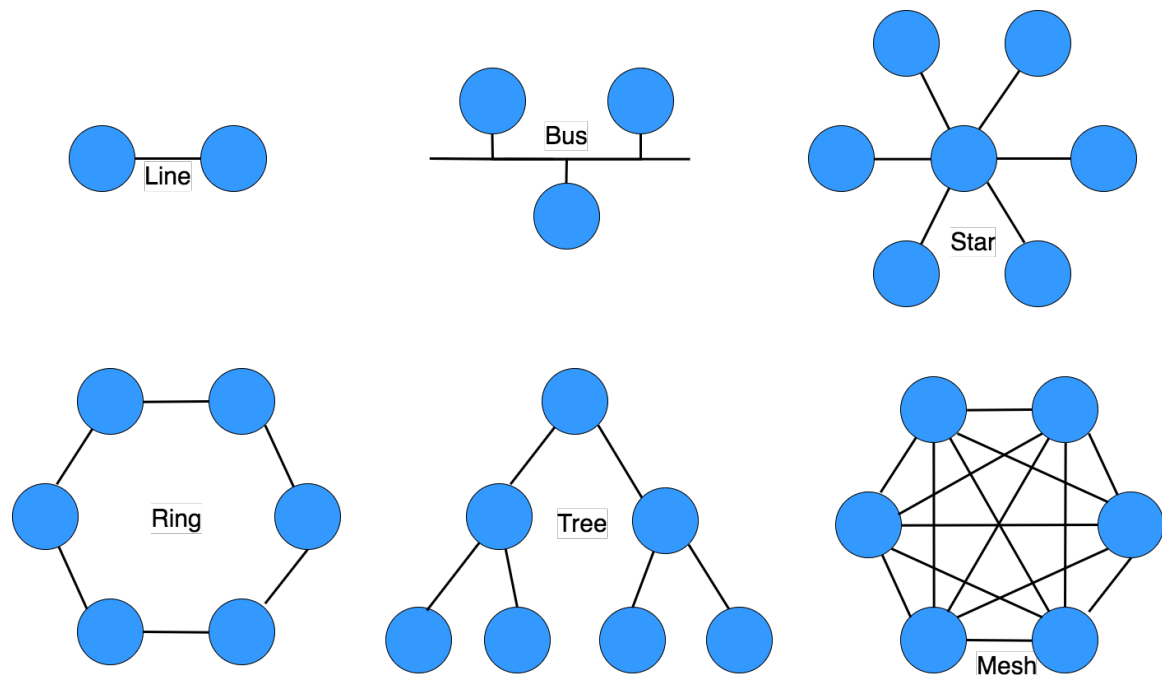


Fig. 4.1 Physical Network Topologies

- **Bus** topology uses a common backbone to connect all nodes. A single cable can be connected with a number of network nodes. A bus topology is cost effective and can be easily expanded.
- **Star** topology features a central connection node (network hub). Compared to the bus, a failure to the communication medium can only affect the attached node and not all the network. A star graph has only one node with degree greater than one.
- **Tree** topology contains a root node at the top level of the hierarchy connected to one or more nodes, one level lower. Tree is a connected graph without cycles. Cellular networks are tree topologies. A chain is a tree where there are no nodes of degree greater than two. A tree can be unreliable due to a single point of failure.
- **Ring** topology contains nodes with each node having two exactly neighbours. All packets travel to the same direction (clockwise or counterclockwise) providing the capability for redundancy in case of a failure.
- **Mesh** topology provides redundant paths. In mesh graph each node is connected with each adjacent/neighbouring node. Each node must have at least a degree of 2 or more. Fully meshed is the graph where every node is connected with all the other nodes. Grid

topology is also a mesh technology used in wireless sensor networks. The messages can be sent in routing logic or in flooding.

- **Hybrid** can combine variations of different network topologies. For instance, a hybrid tree can integrate multiple star topologies into a bus in hybrid approach. The leaf devices connect to a higher level and eventually, all nodes lead to a central node that controls all.

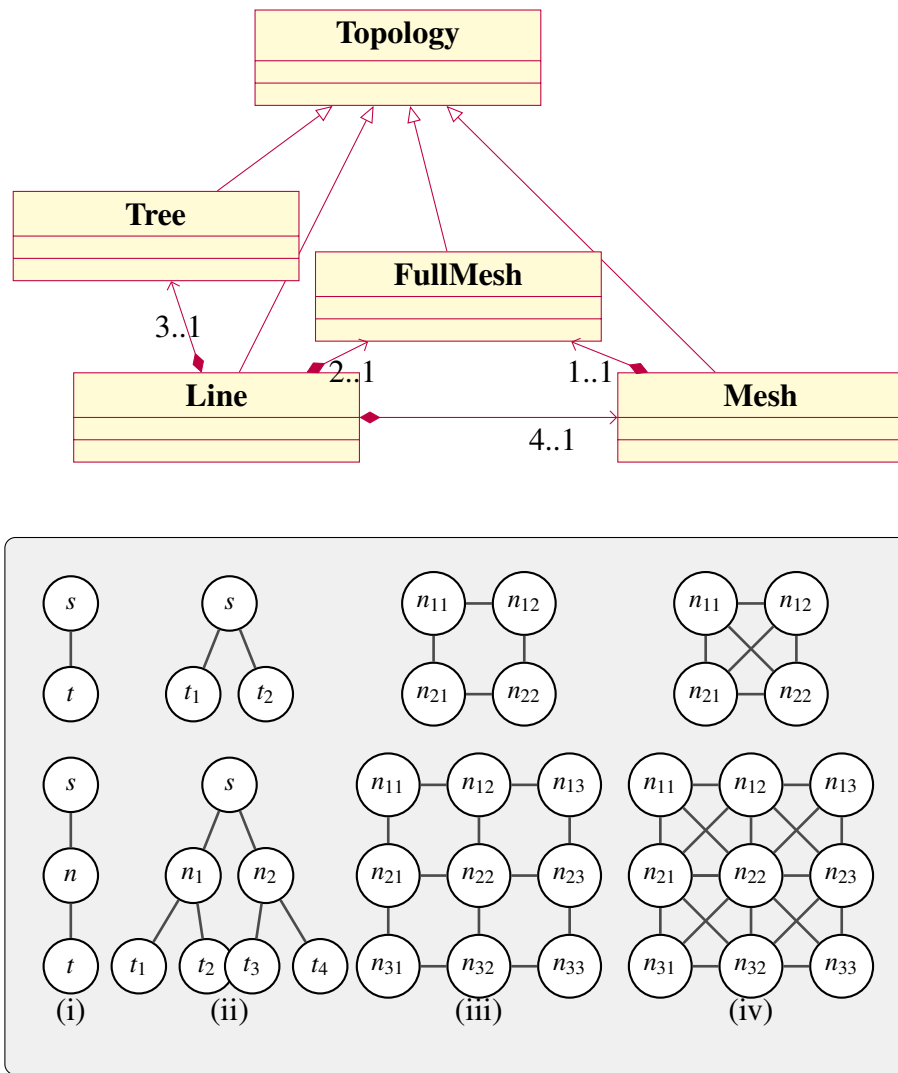


Fig. 4.2 Network Pattern Compositions (i) Line (ii) Tree (iii) Mesh (v) Full-Mesh

An important factor for all the network topologies includes the composition of the different network topology patterns. Based on this procedure, Figure 4.2 presents the composition phases based on basic network patterns as described in the next subsections.

4.2.1.2 Line Pattern

The **Line Pattern** appears to be the basic ingredient for all other basic topology patterns. It is based on the point-to-point network topology similarly to the sequence workflow pattern. The structure of the line pattern includes two placeholders: the *source* node s , the *destination* node t .

$$Line(s,t) = \{Node(s), Node(t)\} \quad (4.1)$$

When \exists a link $Link(s,t)$, then the line pattern defines an atomic composition of nodes. Moreover, when \exists a path $Path(s,t) \nsubseteq Link(s,t)$, the line pattern defines a composition of nodes. The degree of each node is $deg(n) \leq 2$ since all nodes apart from the leaf and the end nodes have degree 1 when all the other have 2. The Line Pattern can design network topologies guaranteeing the **connectivity** property. However, one crucial factor for the line topology is the **distance** between nodes and their **position**. Based on this requirement which affects initially the line pattern and consequently all the network topology designs, the following definitions are given:

- The **distance** between nodes is related to the range of the transmission in case of wireless connectivity or the maximum wired link distance. For instance, in Ethernet, the maximum cable distance between nodes is 100m. On the other hand for wireless networks, the transmission range of two nodes can be calculated by the Friis transmission equation [165]. The generic calculation for finding the distance d between two nodes $n_1 = (x_1, y_1), n_2 = (x_2, y_2)$ can be found as follows:

$$d_{n_1 n_2} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (4.2)$$

- The **position** of a node (x, y) between two other nodes $((x_1, y_1), (x_2, y_2))$ can be defined as follows:

- The straight line between these two points is defined as follows:

$$y = ax + b \Rightarrow y_1 = ax_1 + b, y_2 = ax_2 + b \quad (4.3)$$

- The relation between the nodes and the position of the requested node is:

$$a = \frac{y_1 - y_2}{x_1 - x_2}$$

$$b = y_1 - ax_1$$

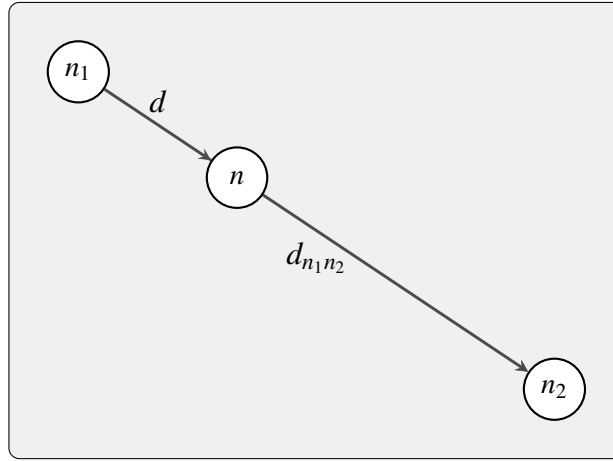


Fig. 4.3 Distance and Position of Nodes

$$y = ax + b = ax + y_1 - ax_1 = a(x - x_1) + y_1$$

- Based on the described equations, the position of a node $n = (x, y)$ with distance d from n_1 is:

$$d = \sqrt{(x - x_1)^2 + (y - y_1)^2} \Rightarrow$$

$$d^2 = (x - x_1)^2 + (y - y_1)^2 = (x - x_1)^2 + (a(x - x_1) + y_1 - y_1)^2 = (x - x_1)^2 + (a(x - x_1))^2 \Rightarrow$$

$$d^2 = (1 + a^2)(x - x_1)^2 \Rightarrow (x - x_1) = \pm \frac{d}{\sqrt{1 + a^2}} \Rightarrow$$

$$x = x_1 \pm \frac{d}{\sqrt{1 + a^2}} \Rightarrow$$

$$x = x_1 + \frac{d}{\sqrt{1 + a^2}}$$

$$y = a(x - x_1) + y_1 = a\left(x_1 + \left(\frac{d}{\sqrt{1 + a^2}} - x_1\right)\right) + y_1$$

- Finally, the position of the node n between the nodes n_1, n_2 is:

$$x = x_1 + \frac{d}{\sqrt{1 + a^2}}, y = a\left(\frac{d}{\sqrt{1 + a^2}}\right) + y_1 \quad (4.4)$$

The line topo pattern can design line network topologies either following forward chaining with line pattern composition or backward reasoning with pattern decomposition. The composition of line pattern can create a line network topology, as can be seen in Figure 4.3.

$$Line(s, n_1) \circ Line(n_1, n_2) \circ Line(n_2, n_3) \circ Line(n_3, t) =$$

$$\begin{aligned} &Line(Line(s, n_1), Line(n_1, n_2)) \circ Line(Line(n_2, n_3), Line(n_3, t)) = \\ &Line(Line(s, n_2), Line(n_2, t)) = Line(s, t) \end{aligned}$$

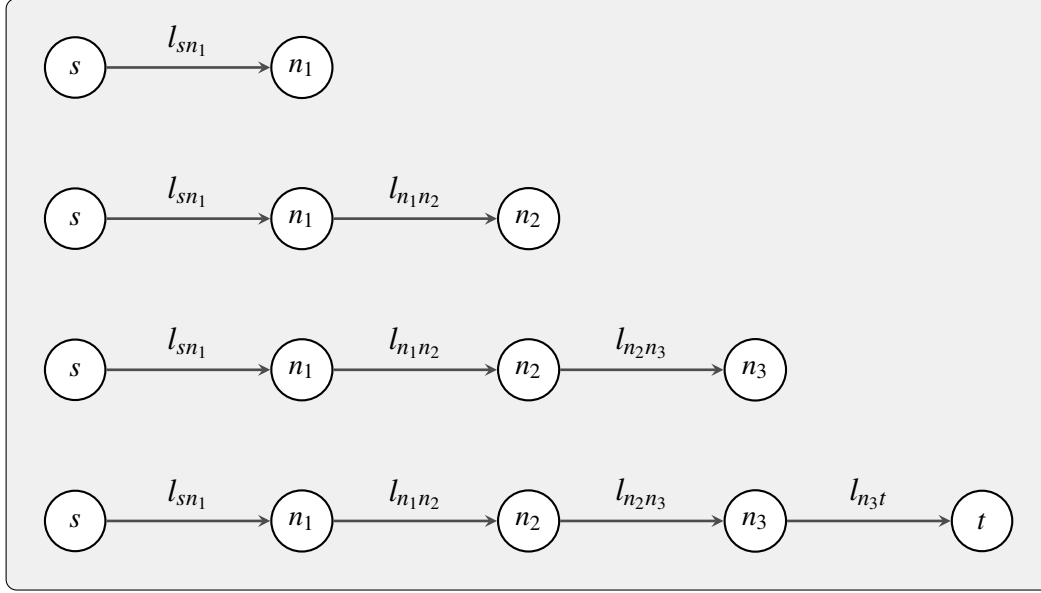


Fig. 4.4 Line Composition

Rule 4.1 expresses the *line topo composition pattern*. The main target of the pattern is to find the required nodes starting from a central node to a predefined range. The rule defines the structure of the pattern, including the source in *node1*, the destination placeholder as *node2*, and the maximum distance between them (*line 4*). When the required connectivity value (*span*) is greater than the distance (*dist*) between the nodes of the line composition, the rule will enter in the RHS (*lines 5-7*). A new node *dst* will be inserted in the working memory and will be placed in the *node2* position of the line topology, creating a link between these nodes *lines 11-12*). Furthermore, the new node will be inserted in the topology in the *node1* position (*line 13*). Finally, a new requirement will be added (*line 14*) triggering also the recursion of the pattern until the final goal is satisfied (*lines 15-16*).

The decomposition of the line pattern is analysed in the Figure 4.5:

$$Line(s, t) = Line(s, n_2) \circ Line(n_2, t) = Line(s, n_1) \circ Line(n_1, n_2) \circ Line(n_2, n_3) \circ Line(n_3, t)$$

The *line decomposition pattern* (Rule 4.2) is able to design line network topologies by applying backward chaining. Starting from the identification of the end nodes (*line 4*) and the requirement of the maximum allowed span between these nodes to guarantee the connectivity property (*lines 5-7*). When the requested distance between nodes is greater

```

1 rule "Line Topo Composition"
2 ruleflow-group "Topo Line Composition"
3 when
4     $topo: Line($src:node1,$dst:node2, $dist:distance)
5     $req: Req($topo:=topo,$pro:property,
6             $pro.name=="Connectivity",$span: $pro.value,
7             $dist<=$span, satisfied==false)
8     $graph: Graph()
9     $count: Integer()
10 then
11     $dst=new Node(++$count,"relay", new Point($src,$dst));
12     $topo.setNode2($dst); $graph.addNode($dst);
13     Link $link=new Link($src,$dst); $graph.addLink($link);
14     Line $line=new Line($dst, new Node(), $dist); insert($line);
15     Req req=new Req($line,new Connectivity($span-$dist),false);
16     insert(req);
17     if ($span<$dist){
18         $dst.setGroup("client");}
19     modify($req){satisfied = true};
20     update($graph); retract($count); insert($count);
21 end

```

Rule 4.1 Line Pattern Composition Pattern Rule

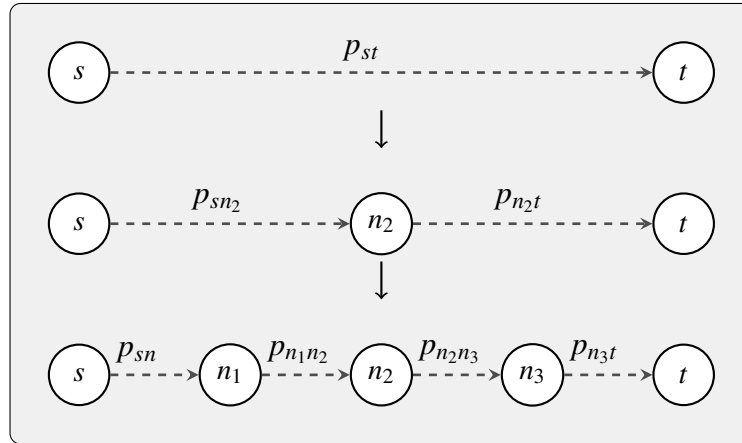


Fig. 4.5 Line Decomposition

than the maximum, the rule will enter to the RHS where a new node will be added (*line 11*). Moreover, two additional line components together with the requirements will be added in the knowledge base (*lines 12-15*). Finally, two additional *links* will be added in the knowledge based when the condition of $dist < span$ is satisfied in the RHS (*lines 16-18*).

```

1 rule "Line Topo Decomposition"
2 ruleflow-group "Topo Line Decomposition"
3 when
4     $topo: Line($src:=node1,$dst:=node2,$dist: distance)
5     $req: Req($topo:=topo,$pro:property,
6             $pro.name=="Connectivity",$span:$pro.value,
7             $dist>$span,satisfied==false)
8     $graph: Graph()
9     $count: Integer()
10 then
11     Node $node=new Node(++$count,"relay", new Point($src,$dist));
12     $graph.addNode($node);
13     Line $line1=new Line($src, $node); insert($line1);
14     Req $req1= new Req($line1,new Connectivity($dist/2),false);
15     insert($req1);
16     Line $line2=new Line($node, $dst);insert($line2);
17     Req $req2= new Req($line2,new Connectivity($dist/2),false);
18     insert($req2);
19     if ($span <= $dist){
20         Link $link1 = new Link($src,$node);$graph.addLink($link1);};
21         Link $link2=new Link($node,$dst); $graph.addLink($link2);};
22     modify($req){satisfied = true};
23     update($graph); retract($count); insert($count);
24 end

```

Rule 4.2 Line Pattern Decomposition Pattern Rule

4.2.1.3 Tree Pattern

The **Tree Pattern** is an extension of the star topology with at least three hierarchical levels. A tree can be unreliable due to a single point of failure. Tree pattern can design network topologies guaranteeing the **scalability** property. The tree topology consists of three different elements: the root, the nodes and the leafs. The degree of each intermediate node is $deg(n) \geq 2$ apart from the leaf and the end nodes are 1.

- The **root** is the first node of the tree that has no parents but only children.
- Each **node** has a single parent node which is closest to the root.
- Each **node** has zero or more children which are farthest from the root.
- The **leaf** is a node without a child.

When the central node is s , the destination nodes are t_1, t_2, \dots, t_k where k is the degree of the tree pattern, the number of the network links can be defined as follows:

$$Link(s, t_1), Link(s, t_2), \dots, Link(s, t_k) \quad (4.5)$$

As described in Figure 4.2(ii), when the composition of two nodes and the degree of root is 2, then the number and the type of components can be defined as follows:

$$Tree(s, t_1, t_2) = \{Line(s, t_1) \circ Line(s, t_2)\} = \{Node(s), Link(s, t_1), Link(s, t_2), Node(t_1), Node(t_2)\}$$

```

1 rule "Tree Topo Composition"
2 ruleflow-group "Topo Tree Composition"
3 when
4     $topo: Tree($root: node1, $leaf1: node2, $leaf2: node3,
5                 $dist: distance)
6     $req: Req($topo:=topo, $pro:=property,
7               $pro.name=="Scalability", $span:=$pro.value,
8               $dist<=$span, satisfied==false)
9     $graph: Graph()
10    $count: Integer()
11 then
12    $leaf1 = new Node(++$count, "relay", new Point());
13    $topo.setNode2($leaf1); $graph.addNode($leaf1);
14    Link $link12 = new Link($root, $leaf1);
15    $graph.addLink($link12);
16    Tree $topo1=new Tree($leaf1, $dist); insert($topo1);
17    Req $req1= new Req($topo1, new Scalability($span-$dist),
18                      false); insert($req1);
19    $leaf2 = new Node(++$count, "relay", new Point());
20    $topo.setNode3($leaf2); $graph.addNode($leaf2);
21    Link $link13 = new Link($root, $leaf2);
22    $graph.addLink($link13);
23    Tree $topo2 = new Tree($leaf2, $dist); insert($topo2);
24    Req $req2= new Req($topo2, new Scalability($span-$dist),
25                      false); insert($req2);
26    if ($dist>$span){
27        $leaf1.setGroup("client"); $leaf2.setGroup("client");
28    }
29    modify($req){satisfied = true};
30    update($graph); retract($count); insert($count);
31 end

```

Rule 4.3 Tree Topo Composition Pattern Rule

The decomposition phase of a tree topology can be defined as follows:

$$\begin{aligned}
 Tree(s, \{t_1, t_2, t_3, t_4\}) &= Tree(s, \{n_1, n_2\}) \circ Tree(n_1, \{t_1, t_2\}) \circ Tree(n_2, \{t_3, t_4\}) \\
 Tree(s, \{Tree(n_1, \{t_1, t_2\}), Tree(n_2, \{t_3, t_4\})\}) &= \\
 &= Tree(s, \{n_1, n_2\}) \circ Tree(n_1, \{t_1, t_2\}) \circ Tree(n_2, \{t_3, t_4\})
 \end{aligned}$$

The *tree topo composition pattern* can be expressed in the Rule 4.3. The tree topo rule can design tree network topologies following forward chaining where the main target of the pattern is to find the required nodes starting from a central node to reach the required scalability property by satisfying a number of end nodes. The rule defines the structure of the pattern, including the source in *root*, and two *leaf* nodes and the maximum distance between them (*lines 4-5*). When the required scalability value (*span*) is greater than the distance (*dist*) between the nodes of the line composition, the rule will enter in the RHS (*lines 6-8*). Two new nodes *leaf1, leaf2* will be inserted in the working memory and will be placed in the *node2, node3* position of the tree topology, with the additional links *lines 12-14, 16-18*). Finally, two new requirements will be added (*lines 15, 19*), triggering also the recursion of the pattern until the satisfaction of the final goal (*lines 20-21*).

4.2.1.4 Mesh Pattern

The **Mesh Pattern** is also based on the basic line network topology. In mesh graph, each node is connected with each adjacent/neighbouring node. Each node has a degree equal to 2 or more. Fully meshed is the graph where every node is connected with all the other nodes. Grid topology is a mesh technology used in wireless sensor networks. The messages can be sent in routing logic or in flooding where all messages are transferred to. Mesh pattern can design network topologies guaranteeing both **coverage** and **redundancy**.

The components of the mesh pattern are the central node s and the destination nodes t_1, t_2, \dots, t_k where k is the degree of the mesh pattern including a number of links:

$$Link(s, t_1), Link(s, t_2), \dots, Link(s, t_k)$$

In the described four-node pattern, the following structure can be defined:

$$\begin{aligned}
 Mesh(s, n_1, n_2, n_3) &= \{Node(s), Node(n_1), Node(n_2), Node(n_3)\} = \\
 &= \{Link(s, n_1), Link(s, n_2), Link(s, n_3), Link(n_1, n_2), Link(n_1, n_3), Link(n_2, n_3)\}
 \end{aligned} \tag{4.6}$$

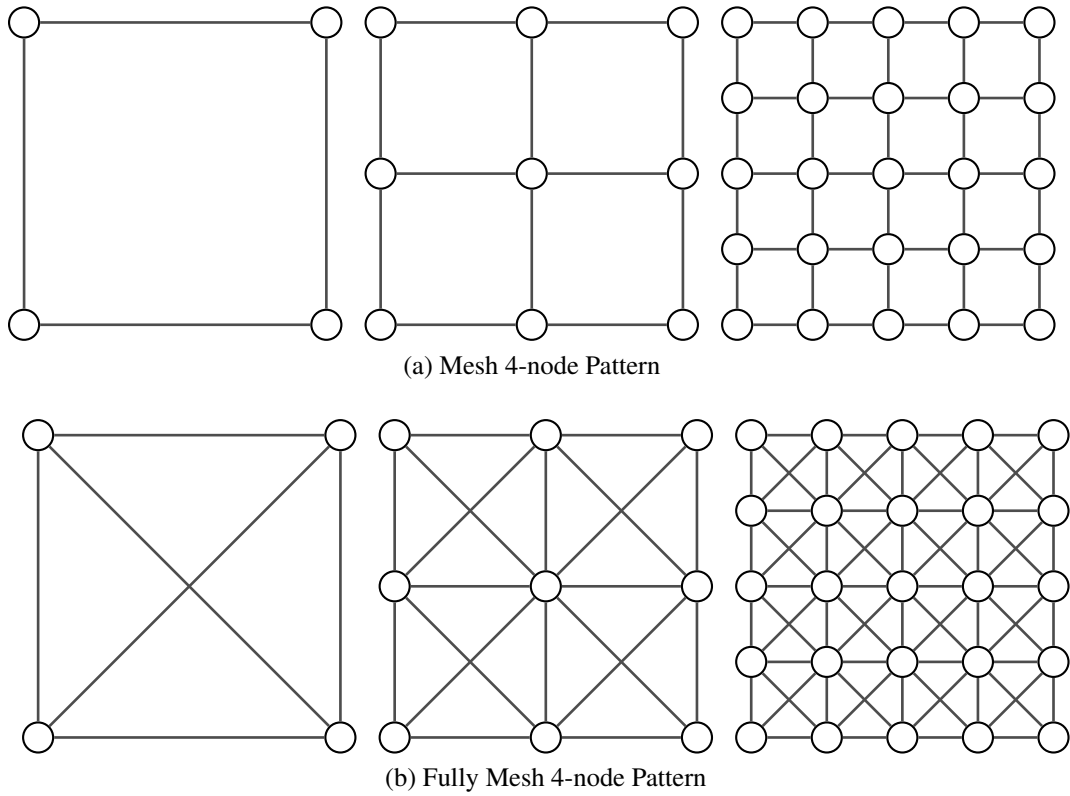


Fig. 4.6 Stepwise Decomposition

The composition of a mesh pattern is based on multiple point-to-point topology. The stepwise decomposition of the mesh and the fully-mesh pattern can be seen in Figure 4.6. More specifically, the decomposition in the mesh pattern can be described as follows:

$$\begin{aligned}
 Mesh(n_{11}, n_{13}, n_{23}, n_{33}) = \\
 Mesh(Mesh(n_{11}, n_{12}, n_{21}, n_{22}), Mesh(n_{12}, n_{13}, n_{22}, n_{23}), \\
 Mesh(n_{21}, n_{22}, n_{31}, n_{32}), Mesh(n_{22}, n_{23}, n_{32}, n_{33})
 \end{aligned}$$

The *Mesh Topo Decomposition Pattern* is presented in Rule 4.4. The pattern is able to design mesh network topologies by applying backward chaining. The rules identifies the end nodes (*line 4*) and the requirement defining the maximum allowed span between these nodes to guarantee the coverage property (*lines 5-7*). Moreover, to avoid duplication of existing nodes in the topology, an additional constraint is entered (*lines 8-9*). When the conditions are met, the rule will enter to the RHS, where four different mesh topologies (*lines 15, 18, 21, 23*) will be inserted in the working memory with the required nodes (*lines 12-14, 17-18, 20*) and requirements (*lines 16, 19, 22, 24*).

```

1 rule "Mesh Topo Decomposition"
2 ruleflow-group "Topo Mesh"
3 when
4   $topo: Mesh($n1: node1, $n2: node2, $n3: node3, $n4: node4,
5     $dist: distance)
6   $req: Req($topo:=topo, $pro: property,
7     $pro.name=="Coverage", $span: property.value,
8     $dist> $span, satisfied==false)
9   $graph: Graph(nodes contains $n1, nodes contains $n2,
10     nodes contains $n3, nodes contains $n4)
11   $count: Integer()
12 then
13   Node $n5 = new Node(++$count, "relay", new Point($n1,$n2));
14   $graph.addNode($n5);
15   Node $n6 = new Node(++$count, "relay", new Point($n1,$n3));
16   $graph.addNode($n6);
17   Node $n7 = new Node(++$count, "relay", new Point($n1,$n4));
18   $graph.addNode($n7);
19   Mesh $mesh1 = new Mesh($n1,$n5,$n6,$n7); insert($mesh1);
20   Req $req1=new Req($mesh1,new Coverage($span), false);
21   insert($req1);
22   Node $n8 = new Node(++$count, "relay", new Point($n2,$n4));
23   $graph.addNode($n8);
24   Mesh $mesh2 = new Mesh($n5,$n2,$n7,$n8); insert($mesh2);
25   Req $req2=new Req($mesh2,$new Coverage($span), false);
26   insert($req2);
27   Node $n9 = new Node(++$count, "relay", new Point($n3,$n4));
28   $graph.addNode($n9);
29   Mesh $mesh3 = new Mesh($n6,$n7,$n3,$n9); insert($mesh3);
30   Req $req3=new Req($mesh3,new Coverage($span), false);
31   insert($req3);
32   Mesh $mesh4 = new Mesh($n7,$n8,$n9,$n4); insert($mesh4);
33   Req $req4=new Req($mesh4,new Coverage($span), false);
34   insert($req4);
35   modify($req){satisfied = true};
36   update($graph); retract($count); insert($count);
37 end

```

Rule 4.4 Mesh Topo Decomposition Pattern Rule

4.2.2 Logical Topology Patterns

Compared to physical topologies, a logical topology defines how devices, components or processes interact or communicate with each other. A logical topology can include the

transmission of signals through a network. The communication of nodes can define the logical topology in a physical topology. Logical topologies describe the path of data between nodes through physical network topologies. The basic building blocks for forming logical network topologies can be the same to those identified for workflow patterns [137]. Flows on physical and cyber networks can be described as a composition of workflows, such as sequence, parallel-split, multi-choice, multi-merge and exclusive choice. More specifically, as it can be seen in Figure 4.7. Workflow patterns are suitable for graph analysis as presented in [166]. The composition of different service functions as chains can be based on different workflows such as the sequence, multi-choice, parallel and exclusive choice based on the predefined workflow patterns as follows:

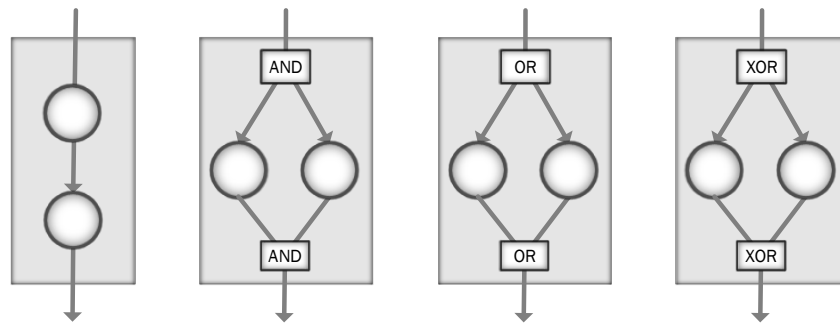


Fig. 4.7 Basic Logical Topologies: (a) Sequence (b) Parallel-split-join (c) Multi-choice-join (d) Exclusive-choice-join

- **Sequence** The sequence pattern defines that a process is enabled after the completion of a previous one. Especially for networks, the sequence topology depicts the sequential composition of nodes in a network. This topology is the fundamental approach for building network process blocks and the diameter/tiers of a network.
- **Parallel-split-join** The parallel-split-join topology (AND–AND) allows the parallel split into two or more branches. This pattern is able to provide load-balance in network transmissions.
- **Exclusive-choice-join** The exclusive-choice-join topology (XOR–XOR) diverges a branch into two or more exclusive branches. The latter topology can be used in networks in order to avoid flooding and for conditional routing.
- **Multi-choice-join** The multi-choice-join topology (OR–OR) provides the execution of a process to be diverged to two or more branches. This topology offers redundancy in network structures.

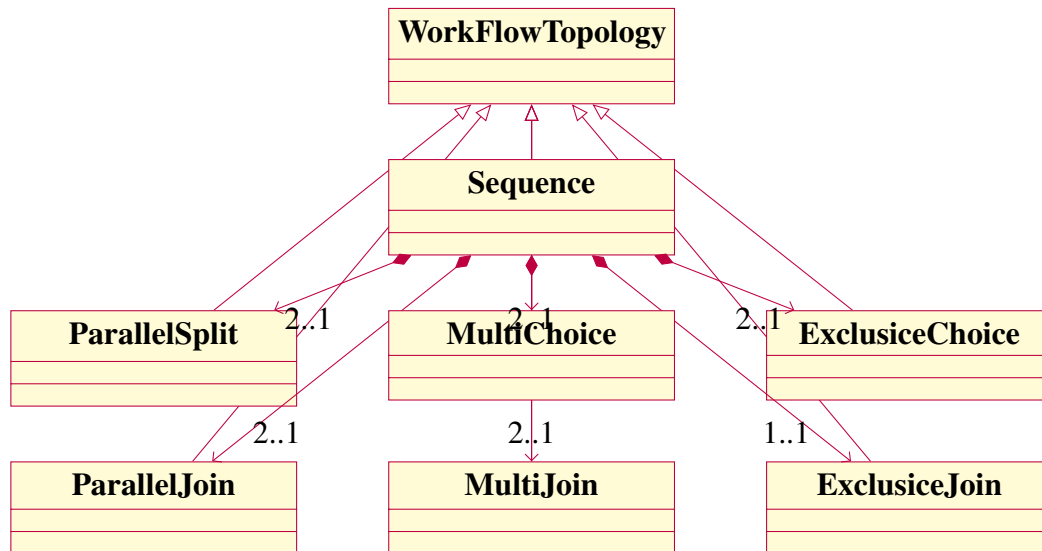
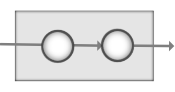
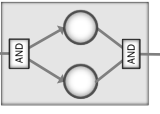
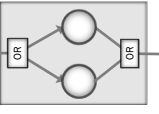
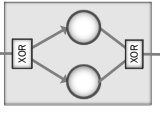


Fig. 4.8 Class Diagram of the Workflow Patterns

The described logical topologies can express the interaction between the components. This involves the different aspects of component compositions related to specific metrics such as the probability, the maximum and minimum costs/time and encryption size, as described in Table 4.1. The logical topologies can be used to design logical network topologies guaranteeing specific S&D properties as described in the next sections.

Table 4.1 Composition Metrics

Pattern Name	Structure	Probability	Max time/cost	Min time/cost	Boolean	Encryption Size
Sequence		$P = \prod_{k=1}^n (p_k)$	$\sum_{k=1}^n c_i$	$\sum_{k=1}^n c_i$	true	$\min\{key_1, \dots, key_n\}$
Parallel-split-join (AND-AND)		$P = \prod_{k=1}^n (p_k)$	$\max\{c_1, c_2\}$	$\min\{c_1, c_2\}$	true/false	$\min\{key_1, \dots, key_n\}$
Multi-choice-join (OR-OR)		$P = 1 - \prod_{k=1}^n (1 - p_k)$	$\min\{c_1, c_2\}$	$\min\{c_1, c_2\}$	true/false	$\min\{key_1, \dots, key_2\}$
Exclusive-choice-join (XOR-XOR)		$P = \min\{p_1, p_2, \dots, p_n\}$	$\max\{c_1, c_2\}$	$\max\{c_1, c_2\}$	true/false	$\min\{key_1, \dots, key_2\}$

4.3 Path Discovery Pattern

Network failures are related to misconfigurations including violation on properties such as reachability, security and dependability. Connectivity is the property that given a graph G and two vertices s and t determine if there is a path between the nodes. A graph is connected if every pair of nodes are connected through a path. Higher connectivity and average nodal degree and shorter average length path and network diameter can lead to higher network reliability [167]. Application reliability is related to network connectivity and routing protocols (Dijkstra, and BFS).

Connectivity plays a crucial role in reliable networks [168]. The connectivity between the different components is one important requirement of the component composition. Different parameters such as the distance between network nodes that is a topological constraint for a network may be expressed through pattern's constraints. For instance, in wired networks the connectivity can be satisfied using suitable interfaces and cables. However, in wireless networks, the connectivity is based on the coverage of each node and it can be classified into deterministic and probabilistic models. However, for a wireless link the following can be assumed: either a communication link is characterised as a component having specific properties (propagation, length, interference, noise, etc.) or a link is a connector of two components i.e. two wireless sensors. As a general statement, connectivity can influence the infrastructure or application communication based on the following factors:

- **Network Topology:** star, tree, mesh etc.
- **Network Characteristics:** connectivity, path length, nodal degree, network diameter.
- **Routing Algorithms:** short-path distance or hop (Dijkstra, BFS, DFS).
- **Delivery models:** unicast, multicast, anycast broadcast etc.
- **Other requirements:** related coverage and density focused on wireless communications.

When there it is necessary to discover a path between two points, the search of available links that is able to compose the path is required. As an example lets consider the path in which a *Link* can be defined also as a *Path* from x to y .

$$Link(x,y) \text{ or } ((Path(x,z) \circ Path(z,y)) \rightarrow Path(x,y) \quad (4.7)$$

When the decomposition of $Path(x,y)$ exists, then following can be assumed.

$$Path(x,y) \rightarrow Link(x,y) \text{ or } (Path(x,z) \circ Path(z,y)) \quad (4.8)$$

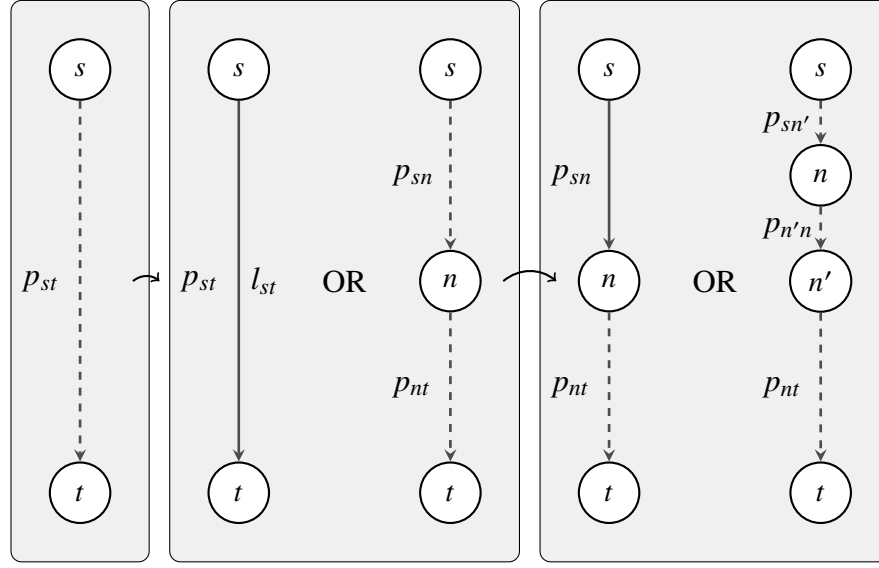


Fig. 4.9 Path Decomposition

Let assume that there is a path between nodes s and t , then the following proposition can be proved.

Proposition 1. *Let a Path between two nodes s, t with $s \in N$ and $t \in N$ exists if and only if there is a link between these two nodes or there is a node $n \in N$ so the following expression can be assumed:*

$$Path(s, t) \leftrightarrow Link(s, t) \text{ or } (Path(s, n) \circ Path(n, t)) \quad (4.9)$$

Proof. The above proposition requires both directions to be proved. Based on path definition, a path is a composition of consecutive edges. Therefore, when there is a link $Link(s, t)$ between two nodes s, t , then a least a $Path(s, t)$ with length 1 exists so $Link(s, t) \rightarrow Path(s, t)$. Furthermore, the composition of two paths $Path(s, n) \circ Path(n, t)$ defines a $Path(s, t)$ so $Path(s, n) \circ Path(n, t) \rightarrow Path(s, t)$. On the other hand, when the length of a $Path(s, t)$ is 1, then a link $Link(s, t)$ should exists. Furthermore, when the length of the path k is greater than 1, then at least one node n exists defining two new paths $Path(s, n)$ with length k_1 and $Path(n, t)$ with length k_2 , $k \leq k_1 + k_2$ so $Path(s, t) \rightarrow Path(s, n) \circ Path(n, t)$. \square

As described previously and depicted in the Figure 4.9, the procedure for path discovery is based on the path decomposition. The procedure is expressed by the **Path Discovery Pattern** which is able to guarantee end-to-end reachability and connectivity by the use of the functionality of Drools to support backward chaining. The pattern is expressed in Rule 4.5 to guarantee the discovery of a path between two nodes. The insertion of a query to express

```

1 query isPath(Node src, Node dst)
2   Link(src, dst;)
3   or
4   (Link(src,node;) and isPath(node, dst;))
5 end
6
7 rule "Path Atomic"
8 ruleflow-group "Path Find"
9 when
10   $link: Link($src:=src, $dst:=dst)
11   $req: Req($src:=src, $dst:=dst, $pro:=property,
12           $pro.name=="Connectivity", satisfied==false)
13   $path: Path()
14 then
15   $path.addLink($link);
16   update($path);
17   modify($req){satisfied = true};
18 end
19
20 rule "Path Decomposition"
21 ruleflow-group "Path Find"
22 when
23   $link: Link($src:=src, $node:=dst)
24   isPath($node, dst;)
25   $req: Req($src:=src, $dst:=dst, $pro:=property,
26           $pro.name=="Connectivity", satisfied==false)
27   $path: Path()
28 then
29   $path.addLink($link);
30   update($path);
31   Req req= new Req($node, $dst, $pro, false );
32   insert(req);
33   modify($req){satisfied = true};
34 end

```

Rule 4.5 Path Discovery Pattern Rules

the Equation 4.9 is described in the *lines 1-5*. In the LHS, when there is a link between the requested nodes (*line 10*) and the requirement is guaranteed (*lines 11-12*), the rule enters in the RHS and the identified link is added in the path (*line 15*). On the other hand, when there is no link between the end nodes, the rule can investigate available linked intermediate nodes (*line 23*). By the use of the query, the *isPath* is applied to walk the path *line 24* and identify the available intermediate node for satisfying the requirement of *line 25-26*. In the

RHS, the path decomposition can be applied to add the link in the path (*line 29*). Finally, the rule will insert a new requirement in order to continue the recursive procedure of secure path identification (*lines 31-32*).

4.4 Reliability Patterns

One of the most important issues for a system designer is to validate system reliability and identify the weakest components in order to replace, redesign or find alternative solutions. For that reason, the reliability is analysed as a critical property for the design of network infrastructures.

Reliability Pattern can guarantee reliability to avoid node failures by installing multiple nodes in parallel at design and assigning new paths at runtime. Reliability patterns can be used to recursively build reliable network topologies. Network reliability can be defined as a weighted graph where the weights represent the probabilistic approach for the node and the links. To extend this approach for complex systems, including spanning tree component compositions, a depth-first search based on a graph theory approach can be used adequately. Especially, for control flow analysis, a reverse postordering depth-first search can be used to produce natural linearisation of directed graphs.

4.4.1 Reliability in Compositions

System reliability depends on component's arrangements. The two basic arrangements include components in series and in parallel. Other arrangements can include parallel-series, k-out-of-n or non-series-parallel systems [169] as will be described in the next subsections.

4.4.1.1 Reliability in Serial Compositions

For components *in series*, the reliability quickly decreases as the number of components increases. In a serial system, a single failure results in entire assembly or system failure. The addition of new components in series decreases the reliability of system. For network topologies such as tree, star, linear and hybrid the logical topology follows the topology of components in series. Components in series may have arrangements either following the sequence or parallel-split workflow patterns. This occurs because a failure of a single component will result in the failure of the system. Reliability of systems in series can be defined as follows:

Definition 4. Let $C = \{c_1, c_2, \dots, c_n\}$ be a number of components in series and r_1, r_2, \dots, r_n be the reliability of each component, then the component composition c will have reliability r

equal to:

$$r = \prod_{k=1}^n (r_k) \quad (4.10)$$

Another topology of this pattern can be the parallel-split. The functional arrangement of this structure is similar to the parallel, where as the logical reliability arrangement is similar to the serial. In this case, the source input of c_1 is equal to the c_2 where as the outputs are different. In this case, both components should be functional in order to guarantee the property of the pattern.

4.4.1.2 Reliability in Parallel Compositions

On the other hand, in case of components *in parallel*, the reliability of the system exists only when at least one component is functional. The reliability of the system is 1 minus the probability that all fail. In parallel components, all redundant units failure causes system failure. Thus, the addition of components in parallel increases the reliability of the subsystem. In addition, actual network topologies, multi-path network topologies such as ring and mesh networking can be expressed as parallel compositions. We may associate the multi-choice pattern as a parallel arrangement because the failure of a single component does not cause system failure. Reliability of components in parallel can be defined as follows:

Definition 5. Let $C = \{c_1, c_2, \dots, c_n\}$ be a number of components in parallel and r_1, r_2, \dots, r_n be the reliability of each component, then the parallel component composition c will have reliability r :

$$r = 1 - \prod_{k=1}^n (1 - r_k) \quad (4.11)$$

Based on the serial and parallel definitions, it can be easily proven the following:

Definition 6. When c is the composition of components c_1 and c_2 with reliability r_1 and r_2 respectively, then (a) if a serial composition c preserves the reliability property, both c_1 and c_2 will satisfy the reliability property and (b) if both c_1 and c_2 preserve the reliability property, the parallel composition c will also satisfy the reliability property.

4.4.1.3 Reliability in Non Series-Parallel Compositions

Reliability in non-series-parallel logical arrangements can define complex network topologies. More precisely, fully-mesh physical network topology represents a specific case in which reliability follows the non parallel-series arrangements. Therefore, to measure the reliability of fully-mesh networks the division method is used [170]. In the fully-mesh topology

consisting of four nodes and six links, the split of the pattern in a non-reducible series of systems can be seen in Figure 4.10.

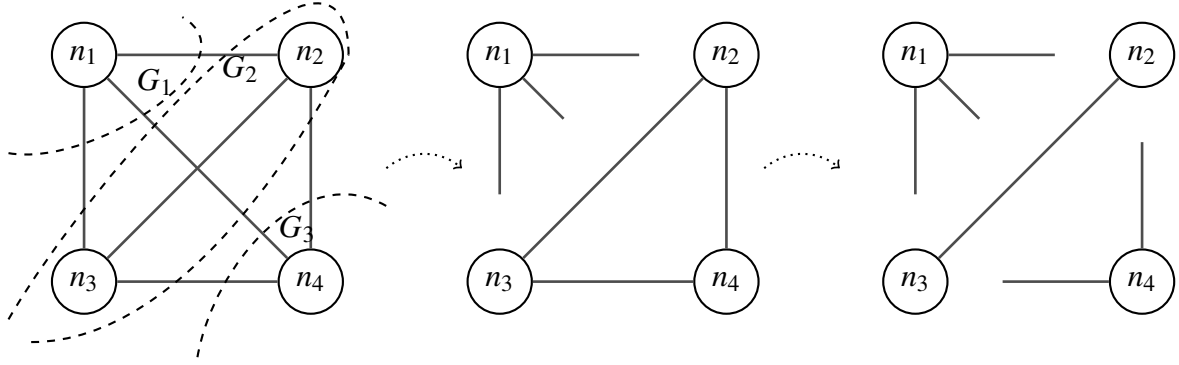


Fig. 4.10 Non-reducible Split of Fully Mesh Pattern

The reliability of the non series-parallel composition can be defined as follows:

$$r = \prod_{k=1}^n (r_k^s) \prod_{k=1}^{n-1} (r_k^p) \quad (4.12)$$

where r_k^s, r_k^p are the non-reducible series and parallel system respectively. In case of the non-reducible split of fully mesh pattern of the Figure 4.10, the equation is:

$$r_{G_1} = r_n \times (1 - (1 - r_l)^3), \quad r_{G_2} = (r_n)^2 \times r_l, \quad r_{G_3} = r_n \times (1 - (1 - r_l)^2)$$

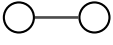
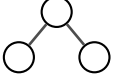
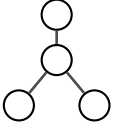
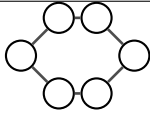
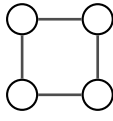
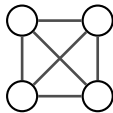
where G_1, G_2, \dots represent the subgraphs of the composition.

So the atomic reliability of the fully-mesh pattern will be:

$$\begin{aligned} r &= r_1 \times r_2 \times r_3 \\ &= (r_n \times (1 - (1 - r_l)^3)) \times ((r_n)^2 \times r_l) \times ((1 - (1 - r_l)^2) \times r_n) \\ &= ((r_n)^4 \times r_l \times (1 - (1 - r_l)^3) \times (1 - (1 - r_l)^2)) \end{aligned}$$

Finally, the summary of the reliability for the different network topologies can be evaluated as depicted in the table 4.2.

Table 4.2 Reliability in Physical Network Topology Patterns

Topo	Topology	Nodes	Links	System Reliability	E2E Reliability
Line		2	1	$(r_n)^2 \times r_l$	$(r_n)^2 \times r_l$
Tree		3	2	$(r_n)^3 \times (r_l)^2$	$(r_n)^3 \times (r_l)^2$
Star		3	3	$(r_n)^3 \times (r_l)^3$	$(r_n)^3 \times (r_l)^3$
Ring		6	6	$(r_n)^6 \times (r_l)^6$	$r_n(1 - (1 - ((r_n)^2 r_l)^3)^2 r_n)$
Mesh		4	4	$(r_n)^4 \times (r_l)^4$	$r_n(1 - (1 - ((r_n) r_l)^2)^2 r_n)$
Fully Mesh		4	6	$(r_n)^4 \times (r_l)^6$	$(r_n)^4 \times (r_l \times (1 - (1 - r_l))^3 \times (1 - (1 - r_l)^2))$

4.4.2 Serial and Parallel Reliability Pattern

In order to create reliable system design based on pattern, the following approach can be used. The pattern validates whether the serial composition satisfies the required reliability property. If this property is not satisfied, a component is added in parallel in order to increase individual component reliability. The procedure continues until the composition of all components guarantees the required property. In Figure 4.11 the execution order of pattern is depicted.

Let assume that a system with two placeholders in series provides the reliability property. Then both c_1 and c_2 should provide the reliability property given $c = c_1 \circ c_2$. If there is no atomic component to preserve the above reliability of c_1 , then a parallel composition of c_{1_1} and c_{1_2} : $c_1 = c_{1_1} \circ c_{1_2}$ may provide such property. The same procedure can be followed for c_2 as well. Based on such parallel composition, we are able to create a reliable composition of atomic components.

The **Reliability Serial and Parallel Pattern** can be expressed as two Drools production rules. These rules encode compositions corresponding to the structure of the logical reliability arrangements.

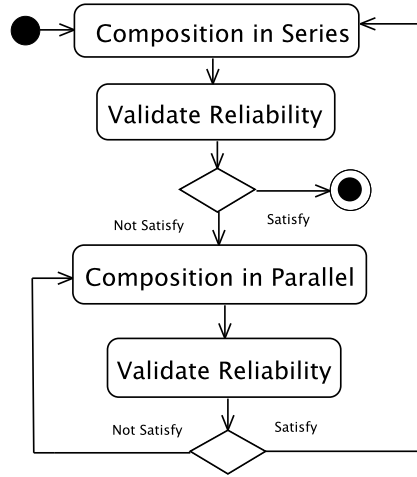


Fig. 4.11 Activity Diagram of Reliability Pattern

We may consider two components, the c_1 with source input v_1 and output v_2 and the c_2 having as source input v_2 and output v_3 and reliability r_1, r_2 . The composition of c_1 and c_2 will be described as a new component c with reliability r based on the components' arrangement. For the component composition in series, the control flow describes the serial arrangement of the components c_1 and c_2 . The data flow defines that for c_1 the output v_2 is the input of c_2 . The composition c will have as an input the v_1 and as an output the v_3 . In addition, the reliability property guaranteed by a serial component composition is equal to $r = r_1 \cdot r_2$. Therefore, the guaranteed reliability property r should satisfy the required reliability property $r^d \leq r$. The encoded pattern in Drools is depicted in Rule 4.6.

The *Reliability Serial Rule* defines three processes of the pattern: the composition of the components, the validation and the guarantee concerning the reliability of the serial component composition. In the LHS, the rule searches for suitable components in which the output of c_1 will be the input of c_2 in order to define a serial composition *lines 3-4*. The topology of the composition (in this case a serial) is required in *Line 5*. In *line 6* the required reliability property is given. If the condition of $r \leq r_1 \cdot r_2$ is met, the rule enters in the RHS. In the RHS, the rule creates a new component c with the input of c_1 , the output of c_2 and as reliability the product of r_1 and r_2 *line 10-11*. The new component c is inserted in the working memory. If the condition is met, the rule will modify the satisfied requirement as true indicating the end of the procedure as the pattern succeeds its goal.

```

1 rule "Reliability Serial"
2   when
3     $c1: Comp($v1:=input, $v2:= output, $r1:= rel)
4     $c2: Comp($v2:=input, $v3:= output, $r2:= rel)
5     $topo: Serial($c1:= src, $c2:= dst)
6     $req: Req($v1:=input, $v3:=output, $pro:=property,
7               $pro.name=="Reliability", $rel:$pro.value,
8               $rel<=$r1*$r2, satisfied==false)
9   then
10    Comp comp = new Comp($topo, $r1*$r2);
11    insert(comp);
12    modify($req){satisfied = true};
13 end

```

Rule 4.6 Reliability Serial Pattern Rule

```

1 rule "Reliability Parallel"
2   when
3     $c1: Comp($v1:=input, $v2:= output, $r1:= rel)
4     $c2: Comp($v1:=input, $v2:= output, $r2:= rel)
5     $topo: Parallel($c1:=comp1, $c2:=comp2)
6     $req: Req($v1:=input, $v2:=output, $pro:=property,
7               $pro.name=="Reliability", $rel:$pro.value,
8               $rel>=$r1+$r2-$r1*$r2, satisfied==false)
9   then
10    Comp comp = new Comp($topo, $r1+$r2-$r1*$r2);
11    insert(comp);
12 end

```

Rule 4.7 Reliability Parallel Pattern Rule

If the property is not satisfied, the components c_1 and c_2 should be replaced by a new component composition in parallel. The *Reliability Parallel Rule* is then fired to find new components which guarantee the required property. The control flow of this rule defines the multi-choice selection of components c_1 and c_2 . The data flow of the component c_1 (line 3) which is in parallel with the c_2 (line 4) must have both input the v_1 and as an output v_2 . The reliability property which should be guaranteed by a parallel component composition (line 6) is equal to $r \leq r_1 + r_2 - r_1 \cdot r_2$ (line 7). If the condition is met, the parallel component composition c will be inserted in the working memory as a new component having as an input v_1 and v_2 as an output and as reliability the $r_1 + r_2 - r_1 \cdot r_2$ (line 10-11). This will trigger a

new reinforcement of the serial rule to check whether new additional component composition satisfy the required by the sequential composition reliability function.

4.4.3 Serial-Parallel Reliability Pattern

Apart from the creation of reliable compositions based on the combination of serial and parallel compositions, the *Serial-Parallel Reliability Pattern* follows both the serial and the parallel simultaneously. The topology of the pattern consists of four nodes, the source n_1 , the destination n_2 and two nodes n_3 and n_4 placed in the middle of end nodes. It also includes four paths:

$$\begin{aligned} p_1 &= Path(src = n_1, dst = n_3) \\ p_2 &= Path(src = n_3, dst = n_2) \\ p_3 &= Path(src = n_1, dst = n_4) \\ p_4 &= Path(src = n_4, dst = n_2) \end{aligned}$$

The decomposition procedure (Figure 6.11b) can continue until atomic links are found as analysed below:

$$\begin{aligned} p = Path(src = n_1, dst = n_2) = \\ (Path(src = n_1, dst = n_3) \circ Path(src = n_3, dst = n_2)) \text{ or} \\ (Path(src = n_1, dst = n_4) \circ Path(src = n_4, dst = n_2)) \end{aligned}$$

The reliability guaranteed by this pattern is related to the serial and parallel path composition. When m is the number of parallel paths p and n is the number of sub-paths of each parallel path and r_{p_i} is the probabilistic reliability of each sub-path, the probabilistic reliability r of the composition can be given by the following formula:

$$r = 1 - \prod_{i=1}^m (1 - \prod_{j=1}^n r_{p_{ij}}) \quad (4.13)$$

Since the topology of Serial-Parallel pattern consists of two parallel paths with two sub-paths in sequence $((p_1 \circ p_2) \text{ or } (p_3 \circ p_4))$, the reliability r will be equal to:

$$r = 1 - (1 - r_{p_1} \cdot r_{p_2})(1 - r_{p_3} \cdot r_{p_4})$$

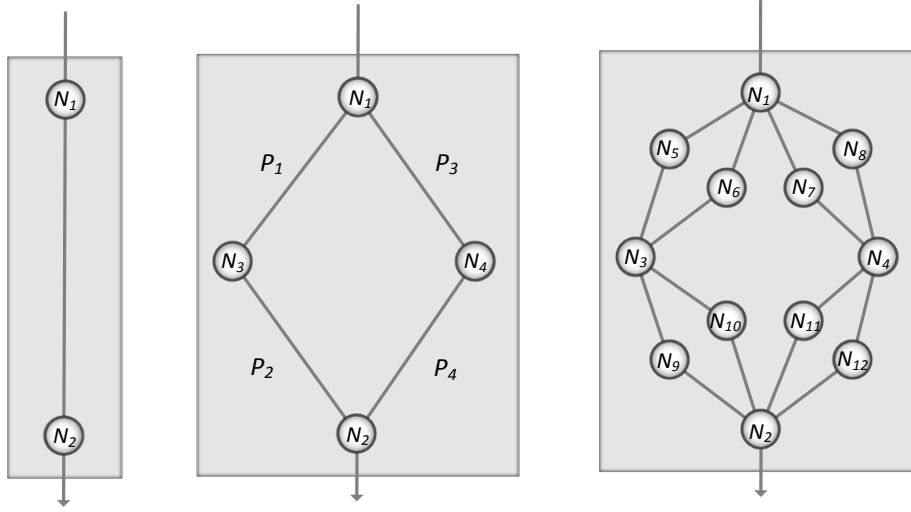


Fig. 4.12 Reliability Serial-Parallel Pattern Decomposition

When the required reliability property of the entire path is r^d , the network reliability r should satisfy the following condition: $r^d \leq r$. In case of equal uptime probability of each sub-path ($r_{p_1} = r_{p_2} = r_{p_3} = r_{p_4} = r_p$), the required reliability should satisfy the equation:

$$r^d \leq r = 1 - (1 - (r_p)^2)^2 \Rightarrow r_p \geq \sqrt{1 - \sqrt{1 - r^d}} \quad (4.14)$$

If there is not any atomic path with this reliability, the pattern will be executed by adding two new nodes and four new paths in the middle distance of each path p_i , as defined by the pattern topology. The new required reliability of each new path will be: $r_p = \sqrt{1 - \sqrt{1 - r^d}}$. it can be easily proven that the $r \leq r^d$ applies for requested path reliability greater than 62%. Finally, the recursive execution of the pattern will increase network reliability and will guarantee the required path reliability.

The pattern rule encodes the described reliability topology (Rule 4.8). In the *LHS* of this pattern, the rule matches the two nodes *src* and *dst* which the distance between them is *dist* with path reliability which assigned as the weight of the path *relP* (line 3). The requirement of the pattern defines that the reliability *relP* should be equal or greater than the *rel* and the constraint is that the distance *dist* between the nodes should be less or equal than the predefined range lines 4-7. When the topology constraint and the reliability are not satisfied the rule will enter in the *RHS* of the rule. In the *RHS*, two nodes n_1 and n_2 should be inserted in parallel between the *src* (line 9) and *dst* (line 14). Moreover, four new paths (lines 10,12,15,17) and requirements (lines 11,13,16,18) will be inserted in the knowledge base. Finally, the rule will modify the requirement satisfaction to true (line 19). The recursive


```

1 rule "Reliability Serial-Parallel"
2   when
3     $path: Path($src:=src, $dst:=dst, $dist:distance, $relP:
4       weight)
5     $req: Req($path:= path, $span:span, $pro: property,
6       $pro.name=="Reliability", $rel:$pro.value,
7       $rel>$relP, $span<=distance, satisfied==false)
8     $count: Integer()
9   then
10    Node $n1 = new Node(++$count, new Point($src,$dst)); insert($n1);
11    Path $p1 = new Path($src,$n1,$rel); insert($p1);
12    Req $req1 = new Req($p1, new Pro("Reliability",
13      Math.sqrt(1-Math.sqrt(1-$relP))), false); insert($req1);
14    Path $p2 = new Path($n1,$dst,$rel); insert($p2);
15    Req $req2 = new Req($p2, new Pro("Reliability",
16      Math.sqrt(1-Math.sqrt(1-$relP))), false); insert($req2);
17    Node $n2 = new Node(++$count, new Point($src,$dst)); insert($n2);
18    Path $p3 = new Path($src,$n2,$rel); insert($p3);
19    Req $req3 = new Req($p3, new Pro("Reliability",
20      Math.sqrt(1-Math.sqrt(1-$relP))), false); insert($req3);
21    Path $p4 = new Path($n2,$dst,$rel); insert($p4);
22    Req $req4 = new Req($p4, new Pro("Reliability",
23      Math.sqrt(1-Math.sqrt(1-$relP))), false); insert($req4);
24    modify($req){satisfied = true};
25    retract($count); insert($count);
26  end

```

Rule 4.8 Reliability Serial-Parallel Pattern Rule

procedure will be completed when the distance constraint and required availability property are satisfied.

4.5 Fault Tolerance, Detection and Restoration Patterns

Fault Tolerance, Detection and Restoration Patterns are able to avoid DoS attacks by creating paths based on the existing links and nodes at design and by creating new paths in case of attack or failure. Fault tolerance in network architectures requires the design of a network able to guarantee avoidance of single or multiple link failures, faulty end-hosts and switches, or attacks. In the control plane, faults appear in the controllers or in the east-west interface. Moreover, the softwarisation of SDN controllers makes the construction of fault tolerance mechanisms even more complicated but at the same time necessary because of

the multi-layer orchestrations. The development of a fault tolerance mechanisms in the northbound and southbound interfaces to avoid intermediate failures is required. The key technical solution of the problem includes the creation of a fault tolerance SDN pattern based on the defined pattern schema that can provide open-flexible design where existing fault tolerance solutions do not.

4.5.1 Fault Tolerance Pattern

Fault tolerance pattern enforces the design of fault tolerance architectures based on existing network infrastructures. The main purpose of the pattern is to provide proactively connectivity and path protection solutions for fast failure recovery against faults, extremely useful for large-scale SDN systems. The discovery of network topologies can be done by pattern matching of suitable interconnected nodes such as number of hosts, switches and the existing links between them. This includes the identification of all available paths and chooses the most appropriate ones in order to define the degree of the redundancy of the network topology. For instance, a path between source and destination will have degree of redundancy 1, where the existence of more than one paths will increase the level of redundancy. In addition, the insertion of suitable flow rules in the programmable switches will enable the prioritisation through the preplanned path identification. To define all paths from source to destination, suitable graph algorithms such as breadth-first algorithm can be applied. More specifically, the shortest path can be found from the Dijkstra algorithm which adapts breadth-first approach to find single source shortest path.

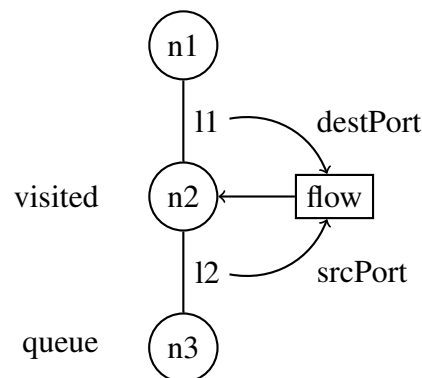


Fig. 4.13 Fault Tolerance Pattern

The describe pattern as Drools rule, is presented in the Rule 4.9. The two main core actions of this rule are:

- The pattern identifies the **shortest path** between source and destination.

```

1 rule "Fault Tolerance Pattern"
2 when
3     $n1: Node($port1: port)
4     $n2: Node($port2: port)
5     $n3: Node($port3: port, $n1!=$n3)
6     $l1: Link($n1:=src, $n2:=dst)
7     $l2: Link($n2:=src, $n3:=dst)
8     $req: Req($pro: property, $pro.src!=$n1, $pro.dest!=$n3,
9             $pro.name=="Fault Tolerance", $visited:$pro.visited,
10            $queue:$pro.queue, $visited contains $n2,
11            $queue not contains $n2, $visited not contains $n3)
12     $path: Path()
13 then
14     $queue.remove($n2); $queue.add($n3); $visited.add($n3);
15     update($queue); update($visited);
16     $path.addLink($l1);
17     update($path);
18     Flow flow = new Flow($n2.id, $l1.dst.port, $l2.src.port);
19     insert(flow);
20 end

```

Rule 4.9 Fault Tolerance Pattern Rule

- To **preplan** and **reserve paths**, the actions of the pattern contains the installation of suitable flow rules in the OpenFlow-enabled switches.

More specifically, the pattern rule in the LHS identifies three nodes (*lines 3-5*) and two links between these nodes (*lines 6-7*). As defined in the *lines 8-10*, the requirement of the pattern to provide fault tolerance is related to the path finding. To find the shortest path the visited/queue approach of the breadth first search is followed. When the requirement is satisfied, the rule enters in the RHS, where the pattern is able to walk the path (*lines 14-15*), store the path *lines 16-17*) and insert the appropriate flow rule in the switch, guaranteeing the priority of the traffic forwarding through this path (*lines 17-18*).

4.5.2 Fault Detection and Restoration Pattern

Apart from the proactive definition of the respective flows, the fault detection and restoration pattern provides a reactive mechanism to dynamically allocate path flows for fast fault detection and restoration of network availability. The pattern topology includes the source node, the destination nodes and the active path for data transmission. To enable the fault/failure detection condition different triggering events should be activated:

- In case of dropped packets between the two nodes.
- In case of a link failure.
- Other monitoring mechanisms including ping/echo, heartbeats, ack, time interval and by request.

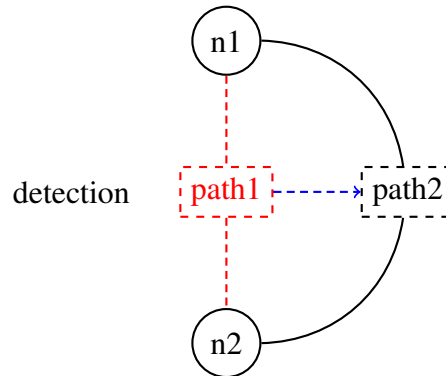


Fig. 4.14 Fault Detection and Restoration Pattern

```

1 rule "Fault Detection and Restoration Pattern"
2 when
3   $n1: Node($tx: txPkt)
4   $n2: Node($rx: rxPkt)
5   $path: Path($n1:=src.id, $n2:=dst)
6   $req: Req($topo:$path, pro.name == "Fault Detection", $tx>$rx)
7 then
8   retract($path);
9   Req req = new Req($n1, $n2, "Fault Tolerance");
10  insert(req);
11 end

```

Rule 4.10 Fault Detection and Restoration Pattern Rule

After the detection of such failure, different actions are described in Rule 4.10 to achieve the following results:

- **Path removal** including all the related links and node from the inventory list.
- **Flow rule removal** from the inventory list (in case of SDN networks).
- **Recover path** by finding alternative paths via the insertion of a new requirement as a new fact to the knowledge base to activate the *Fault Tolerance Flow Pattern*.

The pattern is able to guarantee fault detection when there is data transmission from n_1 to n_2 (lines 3-5). This can be done by identifying the fault detection based on the statistics retrieved by the programmable switches. When lost packets are detected, the rule will enter in the RHS (line 6). In this case, the path will be removed (line 8) and a new requirement for triggering the fault tolerance pattern to instantiate a new path will be inserted (lines 9-10).

4.6 Security Patterns

Security Patterns are able to guarantee confidentiality and integrity in network infrastructures. At design, encryption patterns can be used to avoid man in the middle attacks by providing encryption (i.e. SSL, IPSec) at runtime by updating the encryption keys frequently. Data confidentiality and integrity is guaranteed either by the use of symmetric or asymmetric encryption. The two most common encryption methods are the link and end-to-end encryption as described below.

4.6.1 Link Encryption Pattern

Link Encryption protects traffic flows from monitoring since all data (payload and headers) are encrypted/decrypted in every hop. However, the link encryption is not related to the network protocols. All transmitted data will be encrypted reducing the overhead and the required bandwidth compared to IPSec by as much as 40%¹. When two neighbouring nodes src and dst are connected, the path is encrypted when both nodes are able to encrypt and decrypt the exchanged data. Further constraints of the link encryption pattern relates to the connectivity between edge nodes. More specifically, the link encryption can guarantee confidential data transmission as expressed in the following definition:

Definition 7. *Let a source node s transmit data p to a destination node t , link encryption can guarantee confidentiality when every pair of neighbouring nodes from s to t such as $\{(s, n_1), (n_1, n_2), \dots, (n_i, t)\}$ share keys $k_1, k_2, \dots, k_i \in K$ and are able to encrypt data in source with algorithm E and decrypt in destination with decryption algorithm D .*

When there is a connected Graph such $G = (N, L)$ and there are two nodes $s, t \in N$ connected with a link, without sharing an encryption key k , confidential data exchange cannot be guaranteed until a key generation procedure is applied as described in Rule 4.11. The *Link Encryption Instantiation Pattern* is able to guarantee secure link paths between nodes by creating a set of keys to share the interconnected nodes as described in the Rule

¹<https://blog.finjan.com/what-is-link-layer-encryption/>

4.11. The pattern is able to validate whether two nodes connected with a link (*lines 3-5*) share the same encryption key to provide link encryption (*lines 6-8*). If not, the rule will enter in the RHS where a key generation is applied (*lines 10-13*). The symmetric key is stored in both nodes for enabling the link encryption property and transit secure data (*lines 14-15*).

```

1 rule "Link Encryption Instantiation"
2 when
3   $s: Node($id1:=id, $key1:=key)
4   $t: Node($id2:=id, $key2:=key)
5   $link: Link($s:=src, $t:=dst)
6   $req: Req($s:=src, $t:=dst, $pro:property,
7             $pro.name=="Link Encryption", $value:=$pro.value,
8             $key1 not contains $key2, satisfied==false)
9 then
10  KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
11  SecureRandom secureRandom = new SecureRandom();
12  keyGenerator.init($value, secureRandom);
13  SecretKey key = keyGenerator.generateKey();
14  $s.addKey($key);
15  $t.addKey($key);
16  modify($req){satisfied = true};
17 end

```

Rule 4.11 Link Encryption Instantiation Pattern Rule

When two nodes s and t are connected with a link and both share the same symmetric key k (or k_{st} and k_{ts} in asymmetric cryptography), the transmission can be assumed as confidential.

$$s \xrightarrow[\text{Link Encryption}]{\text{Key } k} t \text{ (Secure link)} \quad (4.15)$$

In addition, when two nodes s and t are connected with a path but no with a link, then for every link of the path, a shared key between the nodes of the links should exist.

$$s \xrightarrow[\text{Link Encryption}]{\text{Key } k_1} n_1 \xrightarrow[\text{Link Encryption}]{\text{Key } k_2} n_2 \dots n_i \xrightarrow[\text{Link Encryption}]{\text{Key } k_i} t \text{ (Secure path)}$$

To identify the secure path in which traffic should be forwarded, the Path Discovery Pattern (Rule 4.5) is extended to fulfill not only the connectivity property but also the link encryption property. More specifically, when there is a requirement to find an encrypted path between two nodes, then the following procedure should be followed.

$$Req(s, t, \text{Link Encryption}) \rightarrow (s == t) \text{ or } (s \rightarrow t) \text{ or } (s \rightsquigarrow t) \text{ with Link Encryption} \longrightarrow$$

```

1 rule "Link Encryption Path Atomic"
2 when
3     $link: Link($src:=src, $dst:=dst)
4     $req: Req($src:=src, $dst:=dst, $pro:=property,
5             $pro.name=="Encryption",
6             $src.key contains $dst.key, satisfied==false)
7     $path: Path()
8 then
9     $path.addLink($link);
10    update($path);
11    modify($req){satisfied = true};
12 end
13 rule "Link Encryption Path Decomposition"
14 when
15     $link: Link($src:=src, $node:=dst)
16     isPath($node, dst;)
17     $req: Req($src:=src, $dst:=dst, $pro:=property,
18             $pro.name=="Encryption",
19             $src.key contains $node.key, satisfied==false)
20     $path: Path()
21 then
22     $path.addLink($link);
23     update($path);
24     Req req= new Req($node, $dst, $pro, false);
25     insert(req);
26     modify($req){satisfied = true};
27 end

```

Rule 4.12 Link Encryption Path Pattern Rules

- $(s == t) \rightarrow$ data are secure
- $(s \rightarrow t)$, *Link Encryption* is *True* when they share the same key k_1
- $(s \leftrightarrow t)$, *Link Encryption* is *True* when there is a node n_1 so
 - $(s \rightarrow n_1)$, *Link Encryption* is *True* when they share the same key k_2
 - $(n_1 \leftrightarrow t)$, *Link Encryption* is *True* when there is a node n_2 so
 - * $(n_1 \rightarrow n_2)$, *Link Encryption* is *True* when they share the same key k_3
 - * $(n_2 \leftrightarrow t)$, *Link Encryption* is *True*...

The pattern Rule 4.12 express the described condition to identify secure paths between two nodes. If there is a link between the nodes (*lines 3*), the requirement is satisfied when

both nodes share the same key (*lines 4-6*). The identified link is added the secure path identification (*lines 9-10*). When there is no link between the nodes (*lines 15-16*) and there is an intermediate node that shares the same key with the source as satisfied by the requirement (*lines 17-19*), the path decomposition can be applied to identify the secure path *lines 22-23*. Then the rule will insert a new requirement to continue the recursive procedure of secure path identification (*lines 24-25*).

```

1 rule "Link Encryption"
2 when
3     $s: Node($id1:=id, $key1:=key)
4     $t: Node($id2:=id, $key2:=key)
5     $link: Link($s=src, $t:=dst)
6     $pkt: Message(packet==plaintext)
7     $req: Req($s:=src, $t:=dst, $pro:property,
8             $pro.name=="Encryption",
9             $key1 contains $key2, satisfied==false)
10 then
11     Encrypt $cph = new Encrypt($pkt, $key);
12     modify($req){satisfied = true};
13 end
14 rule "Link Decryption"
15 when
16     $s: Node($id1:=id, $key1:=key)
17     $t: Node($id2:=id, $key2:=key)
18     $link: Link($s=src, $t:=dst)
19     $cph: Message(packet==ciphertext)
20     $req: Req($s:=src, $d:=dst, $pro:=property,
21             $pro.name=="Decryption",
22             $key1 contains $key2, satisfied==false)
23 then
24     Decrypt $pkt = new Decrypt($cpr, $key);
25     modify($req){satisfied = true};
26 end

```

Rule 4.13 Link Encryption/Decryption Pattern Rules

Finally, when there is runtime data exchange, the source s can encrypt data p to cipher c by the use of E_k and t can decrypt them in the destination by the use of D_k .

$$c = E_k(p) \circ D_k(c) = D_k(E_k(p)) = p \quad (4.16)$$

The Rule 4.13 describes the *Link Encryption and Decryption Pattern* where both rules can express the procedure of encrypting and decrypting data in the source and in the destination respectively. More specifically, *link encryption* rule forwards the packet *pkt* (line 6) from *s* to *t* (lines 3-5). The requirement validates whether both nodes share the same key to enter in the RHS of the rule (lines 7-9). If true, the packet *pkt* is encrypted to cipher by the use of an encryption function (line 11). Similarly to the *link encryption* rule, the *link decryption* rule is able to decrypt cipher text (line 19) by the application of the decryption function (line 24).

4.6.2 End-to-End Encryption Pattern

E2E encryption can guarantee confidentiality and authentication between source and destination based on the composition of different components or applications for secure E2E data transfer, as expressed in the following definition:

Definition 8. Let *s* a source node of transmitting data *p* to destination *t* through intermediate nodes $n_i, i \in \{1, 2, \dots, k\}$, end-to-end encryption can guarantee security when transmitted data is encrypted to *s* by the use of an encryption algorithm *E* and a key *k* and the cypher *c* is decrypted in destination node *t* by the use of a decryption algorithm *D*.

$$s \rightarrow t : \{E\}_k(p) \circ \{D\}_k(c) \quad (4.17)$$

The main property that ensures E2E secrecy is the **Perfect Forward Secrecy (PFS)**. PFS is the property of key-exchange in case that the long-term session keys exposure used for authentication and negotiation does not compromise the secrecy of established keys before the exposure [171]. This can be accomplished by enforcing the creation for each and every session with a new key [172]. Different transport layer security protocols such as IPSec, SSH and TLS are able to satisfy E2E security and the PFS.

IPSec can be used to authenticate and/or encrypt messages for satisfying the goal of E2E security. To instantiate an IPSec E2E session the following should be defined:

- **Security Associations (SAs)** should be generated to encrypt and/or authenticate traffic. The SA can be defined as follows:

$$SA : \langle src, dst, protocol, algorithm, key \rangle$$

where *src* is the source and *dst* is the destination of the association, the protocol can be *ESP* with encryption algorithms such as *3DES/AES* or *AH* algorithm for authentication (*MD5/SHA*) and key is a string.

- **Security Policies (SPs)** define how data can be handled by the devices and implemented following ACL rule structure.

$$SP : \langle match : \{src\{IP, MAC, Port\}, dst\{IP, MAC, Port\}, proto\{ip.icmp, tcp, udp\}\}, \\ action : \{bypass, drop, encrypt, create/lookupSA\} \rangle$$

- **Flow Rules** The creation of the respective SA and SP rules to establish the security property are:

$$F : \langle action : \{encrypt, bypass, drop\}, protocol\{ESP or AH\}, SA\ src, SA\ dst, property \rangle$$

When the SA and the SP have been established, the next step includes the encryption/authentication of traffic as defined by IPSec policies in SA. There are two modes, the transport mode where the data are protected and the tunnel mode where both the header and the payload are protected. However, in order to satisfy property, the need for forwarding the traffic through IPSec or other specific mechanisms such as firewall or NAT may create network unreachability issues. This happens when there is not path or available ACL rules that are able to forward the traffic between two end points. In order to satisfy reachability/connectivity property, the creation of a new requirement for defining the path between source and destination based on the connectivity pattern should be given.

The followed procedure to guarantee encryption for confidentiality and authentication for integrity is depicted in the activity diagram of Figure 4.15. The diagram presents the process to achieve E2E security requirement between two end nodes. Based on the requested encryption or authentication, if the respective SA does not exist, it is instantiated applying either the ESP protocol for encryption (ie. 3DES/AES) or AH for authentication (MD5/SAH). Furthermore, the required SP is instantiated and is inserted to respective flow rules for E2E path instantiation.

The E2E security patterns are able to express the proposed security procedure based on the IPSec mechanism to satisfy secure E2E message transmission. Based on the above, two different E2E security patterns are defined as pattern rules (Rule 4.14 and 4.15) supporting the IPSec protocol and the respective enabling mechanism as described earlier. Moreover, the pattern rules are able to instantiate not only the respective SA to support encryption and authentication introducing a new requirement for path discovery under existing network topologies. More specifically, the *E2E Encryption Pattern* defines the requirement for E2E encryption (*lines 5-6*) between the end nodes (*lines 3-4*). When there is no SA instantiated between these nodes (*line 7*), the rule can enter to the RHS. In the RHS, a new SA is

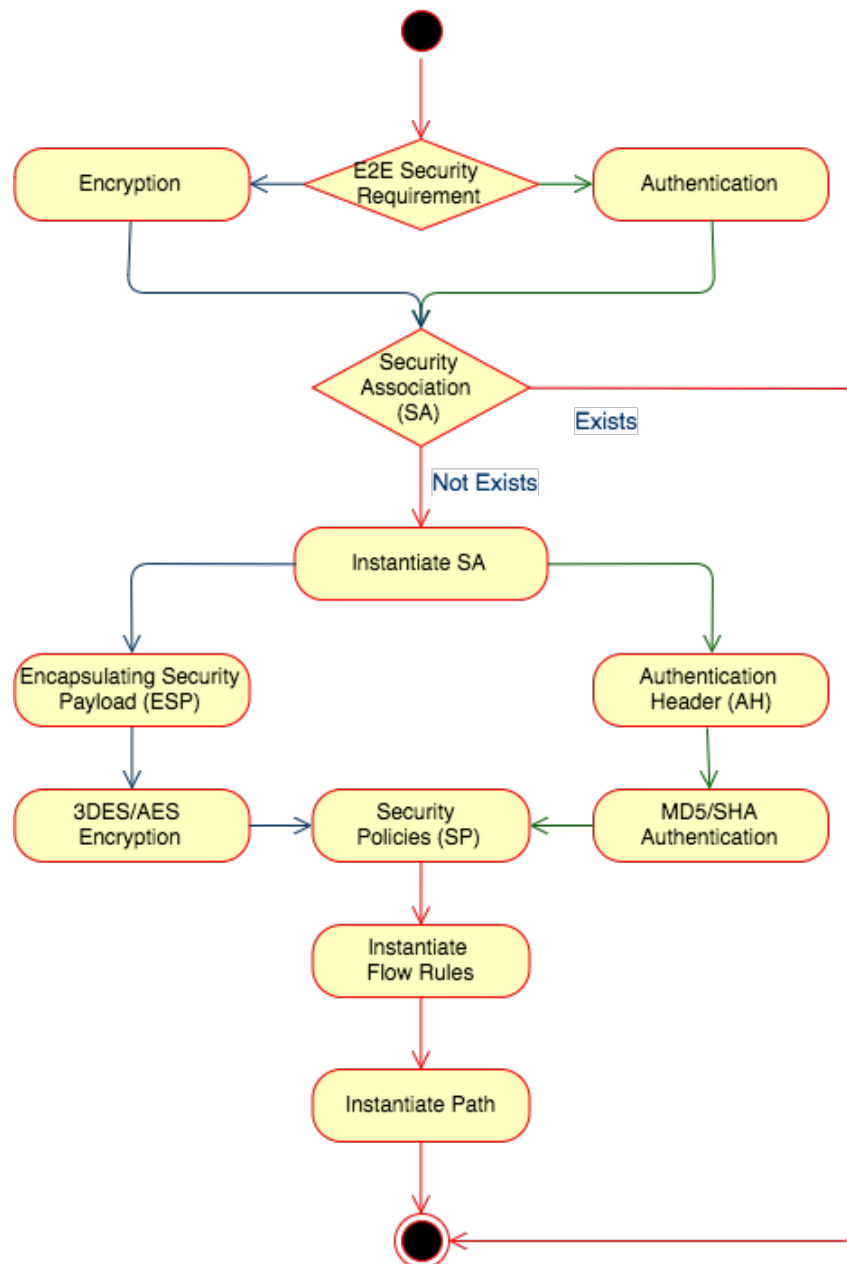


Fig. 4.15 Activity Diagram of the E2E Security Procedure

instantiated and inserted in the memory to establish IPSec encryption by enabling the ESP and AES (*lines 9-10*). Moreover, a new flow rule is created and installed in the memory (*lines 11-12*). The last step of this rule contains the insertion of a new requirement to trigger the *Pattern Discovery Pattern* to identify the shortest path between the end nodes (*lines 13-14*). On the other hand, the Rule 4.15 is able to guarantee authentication. The rule is similar to

```

1 rule "End to End Encryption Pattern"
2 when
3     $n1: Node($ip1: address)
4     $n2: Node($ip2: address)
5     $req: Req($n1:=src, $n2:=dst, $pro:=property,
6             $pro.name == "E2E Encryption", satisfied==false)
7     not (SecAs($ip1:=src, $ip2:=dst))
8 then
9     SecAs sa = new SecAs($n1, $n2, new IPSec(ESP, AES, new
10         Key(IKE)));
11     insert(sa);
12     Flow flow = new Flow($n1, $n2, sa);
13     insert flow;
14     Req req = new Req($n1, $n2, new Connectivity($n1,$n2);
15     insert(req);
16     modify($req){satisfied = true};
17 end

```

Rule 4.14 End to End Encryption Pattern Rule

```

1 rule "End to End Authentication Pattern"
2 when
3     $n1: Node($ip1: address)
4     $n2: Node($ip2: address)
5     $req: Req($n1:=src, $n2:=dst, $pro:=property,
6             $pro.name == "E2E Authentication", satisfied==false)
7     not (SecAs($ip1:=src, $ip2:=dst))
8 then
9     SecAs sa=new SecAs($n1,$n2,new IPSec(AH, SHA2,new Key(IKE)));
10    insert(sa);
11    Flow flow = new Flow($n1, $n2, sa);
12    insert(flow);
13    Req req = new Req($n1, $n2, new Connectivity($n1,$n2);
14    insert(req);
15    modify($req){satisfied = true};
16 end

```

Rule 4.15 End to End Authentication Pattern Rule

Rule 4.14 with the main difference in the instantiation of different security association by enabling the AH and the SH2 authentication mechanism (*lines 9-10*).

```

1 rule "E2E Encryption"
2 when
3     $s: Node($id1:=id, $key1:=key)
4     $t: Node($id2:=id, $key2:=key)
5     $pkt: Message(packet==plaintext)
6     $req: Req($s:=src, $t:=dst, $pro:property,
7             $pro.name=="Encryption",
8             $key1 contains $key2, satisfied==false)
9 then
10    Encrypt $cph = new Encrypt($pkt, $key);
11    modify($req){satisfied = true};
12 end
13 rule "E2E Decryption"
14 when
15     $s: Node($id1:=id, $key1:=key)
16     $t: Node($id2:=id, $key2:=key)
17     $cph: Message(packet==ciphertext)
18     $req: Req($s:=src, $d:=dst, $pro:=property,
19             $pro.name=="Decryption",
20             $key1 contains $key2, satisfied==false)
21 then
22    Decrypt $pkt = new Decrypt($cpr, $key);
23    modify($req){satisfied = true};
24 end

```

Rule 4.16 E2E Encryption/Decryption Pattern Rules

Finally, the *E2E Encryption and Decryption Pattern* (Rule 4.16) aims to guarantee that no message p sent by a source node s through intermediate nodes and links to destination node t can be revealed by an adversary. The rule is similar to the Rule 4.13 but in this one, the link existence between source and destination is not required.

4.7 Service Function Chaining Patterns

Network service deployments are often coupled to network topology, whether it is physical, virtualised, or a hybrid of the two. The problems which SFC is trying to solve are: topology dependencies, configuration complexity, constrained high availability, consistent ordering of service functions etc. SFC aims to address the aforementioned problems associated with service deployment including also the instantiation of a SFC that can be either static or dynamic [173]. In case of a static instantiation, the instances are predefined based on the

assigned configuration of policy of the network administrator. However, the static approach includes the danger of vulnerable paths in case of failure or overload. On the other hand, a dynamic approach can avoid such dangers by introducing the instantiation of SFI according to states or attributes at initial classification or at the time of demand of the SFP intermediate traversal. Based on the above, one of the most important issues in SFC is the current service delivery model that is usually bound to static topologies and manually configured resources.

Service Function Chaining Patterns provide the ability to define an ordered list of security network services (e.g. firewalls, DPIs, IDS) for security in network infrastructures by creating chains at design and by updating function in chains based on available ones at runtime. SFC patterns should cover the placement, security and scalability aspect of SDN/NFV-enabled network infrastructures. More precisely, SFC patterns can be used as follows:

- Instantiate virtual network function to address SFC requests.
- Instantiate service function chains
- Design and optimisation of paths

Our definition is based on formal analysis of the composition of service function chaining and the network functions as presented in [174, 175].

4.7.1 VNF Instantiation Pattern

The design of SDN/NFV-enabled networks includes the placement of network functions either in a physical network function (PNF) in a VNF. PNFs can be also called Service Node (SN) containing of one or multiple VNF instances. NFV is focused on the replacement of PNFs with VNFs that is a virtual version of a network function and can be located in physical nodes. They can be located on physical appliances or virtual machines, running in virtualisation infrastructures.

The physical network can be defined as $G = (N, L)$ while the virtual network can be defined as $\bar{G} = (\bar{N}, \bar{L})$. Virtual graph may share or not some common elements with the physical graph G . The physical placement of network functions in the physical nodes includes also the partition of the network and the placement of functions on different locations as can be seen in Figure 4.16. More specifically, the different cases of the VNFs inside the PNFs connected on different SDN topologies can be seen below:

1. Different network functions on the same service node, switch and controller (domain)

2. Different service node with different network functions on the same switch and controller (domain)
3. Different network switches holding different service nodes and functions on the same controller (domain)
4. Different controller connected on different switches holding different service nodes and functions on different domains and controllers

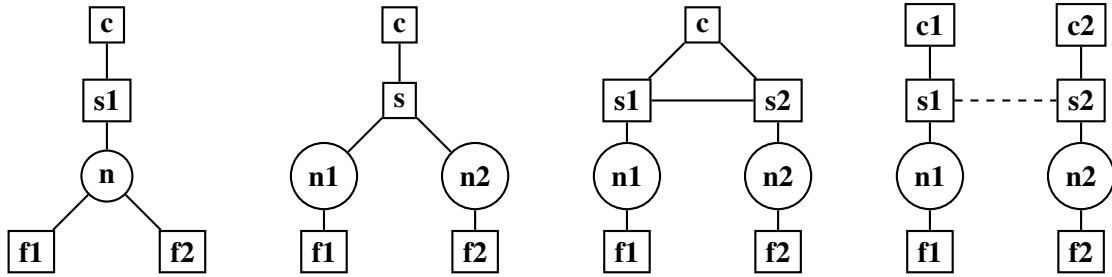


Fig. 4.16 Service Functions (i) on Same Service Node, (ii) on Same Switch (iii) on Same Domain (iv) on Different Domains

The applications and services that make use of the network are crucial factors for the design of a network as they can affect the limitations of the device such as computational power, available memory, storage and networking capabilities. Other constraints which may be defined include the quantity and type of nodes, interfaces per nodes, cost and energy consumption, distance and range. More specifically, each node of the network has specific characteristics and constraints as follows:

- Each physical node $N = \{n_1, n_2, \dots\}$ has specific resource (or capacity) capabilities so

$$R = \{CP, M, T, E\} \quad (4.18)$$

where computation power/CPU (CP), memory (M), storage (T), energy (E).

$$R_{n_j} = \{CP_n, M_n, T_n, E_n\} \quad (4.19)$$

- Each physical node $n \in N$ can hold a number of virtual network functions $F = \{f_1, f_2, \dots\}$

$$F \subseteq N$$

- Each network function requires a certain amount of physical resources at the physical node such as CP_{f_i} , M_{f_i} , T_{f_i} and E_{f_i} . The required capacity by each function can be defined as follows:

$$R_{f_i} = \{CP_{f_i}, M_{f_i}, T_{f_i}, E_{f_i}\} \quad (4.20)$$

- Each function node should not exceed the capacity of physical nodes. This includes also the function placement based on the resource capabilities of each service node.

$$R \geq \sum_{i=1}^k R_{f_i} = \sum_{i=1}^k \{CP_{f_i}, M_{f_i}, T_{f_i}, E_{f_i}\} \geq \left\{ \sum_{i=1}^k CP_{f_i}, \sum_{i=1}^k M_{f_i}, \sum_{i=1}^k T_{f_i}, \sum_{i=1}^k E_{f_i} \right\} \quad (4.21)$$

- Each function node f_i can be instantiated (i) once, (ii) at most u times, i.e. due to the limited number of licenses (iii) unlimited times.

$$\sum_{i=1}^k \sum_{u=1}^z f_{iu} \quad (4.22)$$

On the other hand, each physical or virtual link of the network has specific characteristics and constraints as follows:

- Each physical link $l \in L$ between two nodes $n_1, n_2 \in N$ so $l_{n_1 n_2}$ has specific constraints such as (i) limited bandwidth $BW_{l_{n_1 n_2}}$ and (ii) introducing a link propagation delay $D : D_{l_{n_1 n_2}}$ so $l_{n_1 n_2} : R_{l_{n_1 n_2}} = \{BW_{l_{n_1 n_2}}, D_{l_{n_1 n_2}}\}$
- Each virtual link \bar{l} between virtual functions f_i and n_i .

In order to allocate virtual resources in physical nodes, the requested capacity of N^d of the virtual node \bar{n}_i is characterised by the requested resources R^d including the requested node capacity regarding CPU, memory, storage and energy. In addition, the requested link capacity of L^d of the virtual link \bar{l}_{ij} is characterised by the requested resources in bandwidth BW^d and the allowed delay D^d between the end points.

$$G^d = (N^d, L^d) \quad (4.23)$$

When there is a request for an SFC instantiation containing service functions, the depicted in Figure 4.17 procedure should be followed. If the SFC does not exist, the instantiation of the respective SFC is deployed through the identification of the requested VNFs. If the VNFs exist in the service nodes, the SFC is updated including these VNFs. If the VNFs do not exist, the service node with the available resources is requested to instantiate the respective

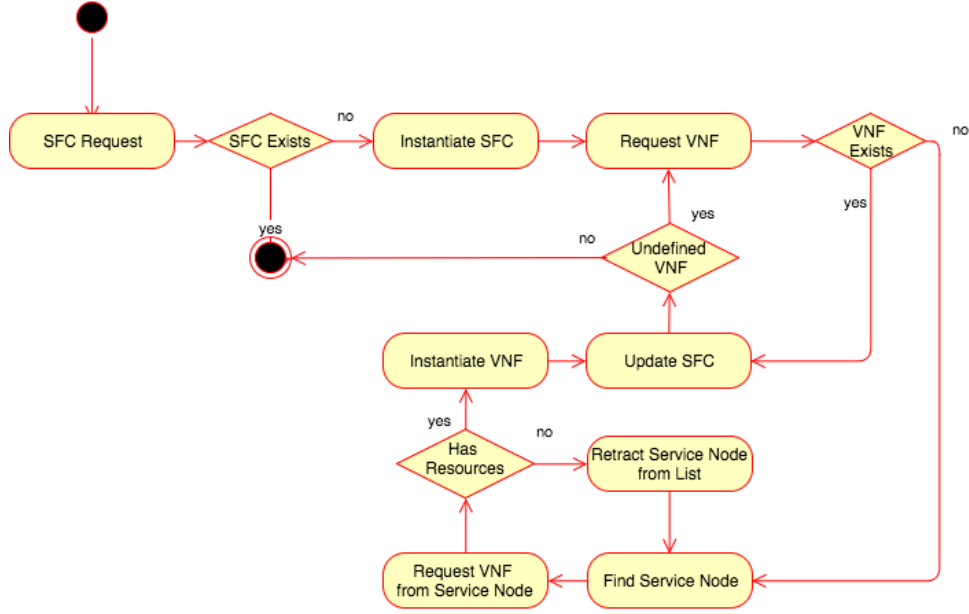


Fig. 4.17 VNF Instantiation based on SFC Request

Algorithm 4 Recursive Procedure of SFC Pattern

```

1: procedure SFC ( $G, S^d$ )
   Input:  $s^d = (u, v, s)$  is the requirement from  $u$  to  $v$  and SFC  $s = (f_1, f_2, f_3)$ , and  $G = (N, L)$ 
   defines the service node  $N$  nodes and the links  $L$  of the network
   Output: Shortest path from  $u$  to  $v$  passing from the chain  $s$ , satisfying the constraints
   bandwidth  $b$  and delay  $d$ 
    $\bar{G} = (\bar{N}, \bar{L})$ 
2:   for all  $SF$  do
3:     for all  $SN$  in  $N$  do
4:       if  $SF$  does not exist in  $SN$  then
5:         if available  $SN$  resources can satisfy required  $SF$  resources then
6:           instantiate a  $SF$  in  $SN$ 
7:           update resources of  $SN$ 
8:           add  $SN$  to  $SFP$ 
9:         else break
10:        end if
11:      else
12:        add  $SN$  to  $SFP$ 
13:      end if
14:    end for
15:  end for
16: end procedure

```

```

1 rule "Virtual Network Function Instantiation"
2   when
3     $vnf: Function($type: type,
4       $resources:resources,undefined==true)
5     $graph: Graph($node: node, $node not contains $function,
6       $node.hasResources($resources) }
7     $req: Req($sfc: chain, $sfc not contains $vnf,
8       satisfied==false)
9   then
10    $node.add($vnf);
11    update($node);
12    modify($vnf){undefined = false};
13    $sfc.add($vnf);
14    update($sfc);
15  end
16 rule "Virtual Network Function Find"
17   when
18     $vnf: Function($type:
19       type,$resources:resources,undefined==true)
20     $graph: Graph($node: node, $node contains $function)
21     $req: Req($sfc: chain, $sfc not contains $vnf,
22       satisfied==false)
23   then
24     modify($vnf){undefined = false};
25     $sfc.add($vnf);
26     update($sfc);
27  end

```

Rule 4.17 Virtual Network Function Placement Pattern Rules

VNFs. The procedure is ended when all the requested VNFs are included in the SFC. The respective algorithm is described in the Algorithm 4.

The *Virtual Network Function Placement Pattern* is able to process the previously described procedure as presented in the abstract form of the pattern Rule 4.17. The pattern consists of two parts: the *VNF Find Rule* and the *VNF Instantiation Rule*. The rule is triggered when a VNF does not exist in any service node, there are resources available to instantiate it (lines 3-4) and there is a SFC request (line 5). The rule will enter in the RHS, where the node will instantiate a VNF (lines 7-8) and will be added in the SFC (lines 10-11). On the other hand, when the VNF exists (lines 15-16) and the SFC does not include it (line 17), it will be added in the SFC without the need for any additional instantiation (lines 20).

4.7.2 SFC Path Finding Pattern

The dynamic creation of SFC chains according to available or instantiated function instances and security policy requirements can be described as virtual graph $\bar{G} = (\bar{N}, \bar{L})$.

Definition 9. Let the forwarding graph \bar{G} defines an SFC graph so $\bar{G} = (\bar{N}, \bar{L})$, where \bar{G} is the service chain graph, $\bar{N} = \{f_i | i = 1, 2, \dots, k\}$ can be considered as the service functions representing the virtual nodes of the graph $S = \{s_k | k = 1, 2, \dots, k\}$ are the set of chains that hold different service functions types and \bar{L} are the virtual links that interconnect the service functions.

An SFC includes all the services that are stitched together. However, the path of the chain can include all the actual components that are contained in the path such as source u and the destination v nodes, the classifiers, the forwarders, the service nodes where the functions are located and the function nodes. As depicted in Figure 4.18, the conceptual and extensive SFC diagram consisting of a two service functions, the following backward chain decomposition can include the following steps:

- Service Function Chain : $s = (f_1, f_2, \dots, f_k)$
- Service Function Path: $p = \{u, f_1, f_2, f_3, v\}$, where u, v are the ingress and the egress nodes.
- Service Function Path can include apart from the ingress u and the egress v , the service nodes:

$$\{u, n_1, f_1, f_2, n_2, f_3, v\}$$

- Service Function Path can include apart from the ingress u and the egress v , the physical nodes n^p and the service functions f , the classifiers n^c and the forwarders n^f :

$$\{u, n_1^c, n_1^s, n_1^p, f_1, f_2, n_2^s, n_2^p, f_3, n_2^c, v\}$$

The design and the optimisation of service function chains and the respective paths include the finding of the optimal routes. The position of the classifier and forwarders can improve network performance (by minimising the delay and number of physical hops) based on capacity constraints. In addition, the order, the placement and the selection of the service functions can satisfy the security requirements of the chain as described below:

- **Inspect encrypted data:** For instance, encrypting functions such as VPN or IPSec should be placed after firewall (and DPI) or just before decryption functions [176].

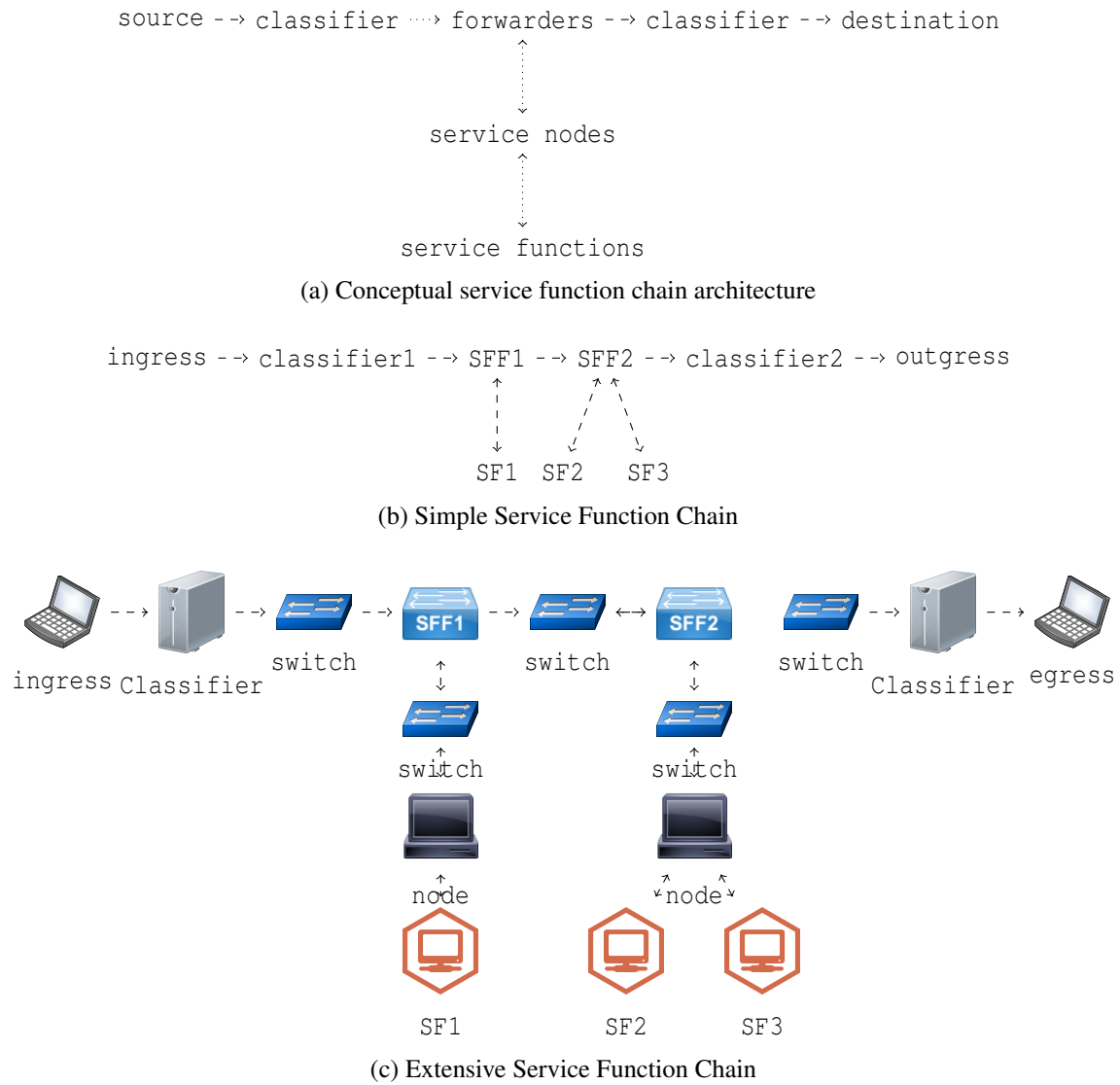


Fig. 4.18 Conceptual, Simple and Extensive Service Function Chaining Example

- **DDos attacks avoidance** based on an IDS should be placed close to the source
- **End-to-end encryption** should be done before the untrusted networks.

When there is a request S^d consisting of two or more end points of a path S_p^d , a set of virtual functions S_f^d that need to be mapped and the set of virtual links s^d between endpoints and functions should be defined. Let S^d represent all SFC requests that have to be embedded in the network, the service request may include the ingress and egress through a list of intermediate services as described below:

$$s^d = (u, v, s, bw, t, d) \quad (4.24)$$

where u is the source, v is the destination, s is the set of required VNFs for the SFCs, bw is the bandwidth requirement; t is the lifecycle; and d is the maximum allowed delay.

```

1 rule "SFC Verification"
2 when
3     $src: Node($src: src)
4     $dst: Node($dst: dst)
5     $chain: Chain($functions: functions)
6     $req: Req($src:=src, $dst:=dst, $pro: property,
7             $pro.name:= "Chaining", $chain:=$pro.chain,
8             satisfied==false)
9 then
10    modify($req){satisfied = true};
11    Req req = new Req($src, $dst, new Pro("Chain Path", $chain),
12        false); insert(req);
13 end
14 rule "SFC Instantiation"
15 when
16     $src: Node($src: src)
17     $dst: Node($dst: dst)
18     $req: Req($src:=src, $dst:=dst, $pro: property,
19             $pro.name:= "Chaining", $chain:=$pro.chain,
20             satisfied==false)
21     not Chain($chain.functions:=functions)
22 then
23     Chain $chain = new Chain($functions); insert($chain);
24     Req req = new Req($chain, false); insert($req);
25     modify($req){satisfied = true};
26 end
27 rule "SFC Path Discovery"
28 when
29     $req: Req($src:=src, $dst:=dst, $pro:= property
30             $pro.name=="Chain Path", $chain:= $pro.chain,
31             satisfied==false)
32 then
33     Req req1 = new Req($src, $chain.getFirst(),
34         new Connectivity(true), false); insert(req1);
35     $chain.removeFirst(); update($chain);
36     Req req2 = new Req($chain.getFirst(), $dst, $pro, false);
37         insert(req2);
38     modify($req){satisfied = true};
39 end

```

Rule 4.18 SFC Pattern Rules

The *SFC Pattern* abstract definition of the pattern rule is depicted in Rule 4.18. The pattern contains three different rules in which each one implements specific actions. The first action of the pattern is to identify in the *SFC Verification Rule* if the requested SFC exists in order to forward the traffic from source to destination (*lines 3-5*). If the requirement is true, the rule will enter in the RHS where a new requirement regarding the identification of the SFC path will be inserted (*line 11*). If the chain does not exist (*line 20*), the *SFC Verification Rule* will be triggered to instantiate the requested chain (*line 22*) and a new requirement for function instantiation will be inserted in the knowledge base (*line 23*). This requirement will also trigger the previously described *Virtual Network Function Placement Pattern* to identify or instantiate any required network function by the chain. When the chain is created including the required functions, the *SFC Path Discovery* will be enforced. Two additional requirements will be inserted in the knowledge based. The first one will trigger the *Path Discovery Pattern* to identify the shortest path between the source and the first function of the chain (*lines 32-33*). After the removal of the first function *line 34* from the chain list, a new requirement for a *SFC Path Discovery* will be inserted (*line 35*). The recursion will be completed when all the requested chains will be instantiated with respective network functions. To conclude, the procedure described above represents a very accurate example on how the composition of a number of different patterns can provide a complete solution for the design of SDN/NFV-enabled networks.

4.8 Summary

In this chapter, a set of pattern instances was presented. A number of different patterns encoded as rule were presented in order to design network topologies and guarantee S&D properties in SDN/NFV-enabled networks. More specifically, the design of network topologies is based on the enforcement of topology patterns, such as the line, the mesh and the tree, able to guarantee functional properties such as connectivity coverage and scalability respectively. In addition, the presented reliability patterns can assist the design and verification of reliable network topologies. The development of fault tolerance patterns in SDN/NFV-enabled architecture can be used for enabling proactive and reactive fault tolerance and restoration mechanism. Moreover, the security patterns can guarantee link encryption and end-to-end encryption in network topologies. Finally, the SFC patterns are able to provide service provisioning for secure and traffic forwarding through different network service functions.

Chapter 5

Implementation of the Pattern Framework

5.1 Overview

As discussed in the previous chapters, design patterns can be used for the design, configuration and verification of network topologies to guarantee some S&D properties. To give a proof of concept of the approach, the implementation of a prototype pattern framework is proposed and developed to evaluate the applicability of this pattern schema and the potentiality in the SDN/NFV-enabled network environments. The system architecture and the interaction between blocks are presented in the Figure 5.1. The developed pattern framework consists of different layers:

- **Application Layer:** This layer includes the translation of network monitoring and management requirement of applications as expressed by the proposed pattern schema.
- **SDN/NFV Layer:** This layer contains two main components: the SDN controller which enables the different network capabilities and the NFV MANO for the virtualisation of network infrastructures.
- **Infrastructure Layer:** The infrastructure layer is responsible to host the different network elements required for deploying network topologies. This layer can include components that are responsible for forwarding or interacting with the data flow such as switches, routers, hosts, servers, IoT devices, in a physical infrastructure, a virtual emulated or simulated environment.

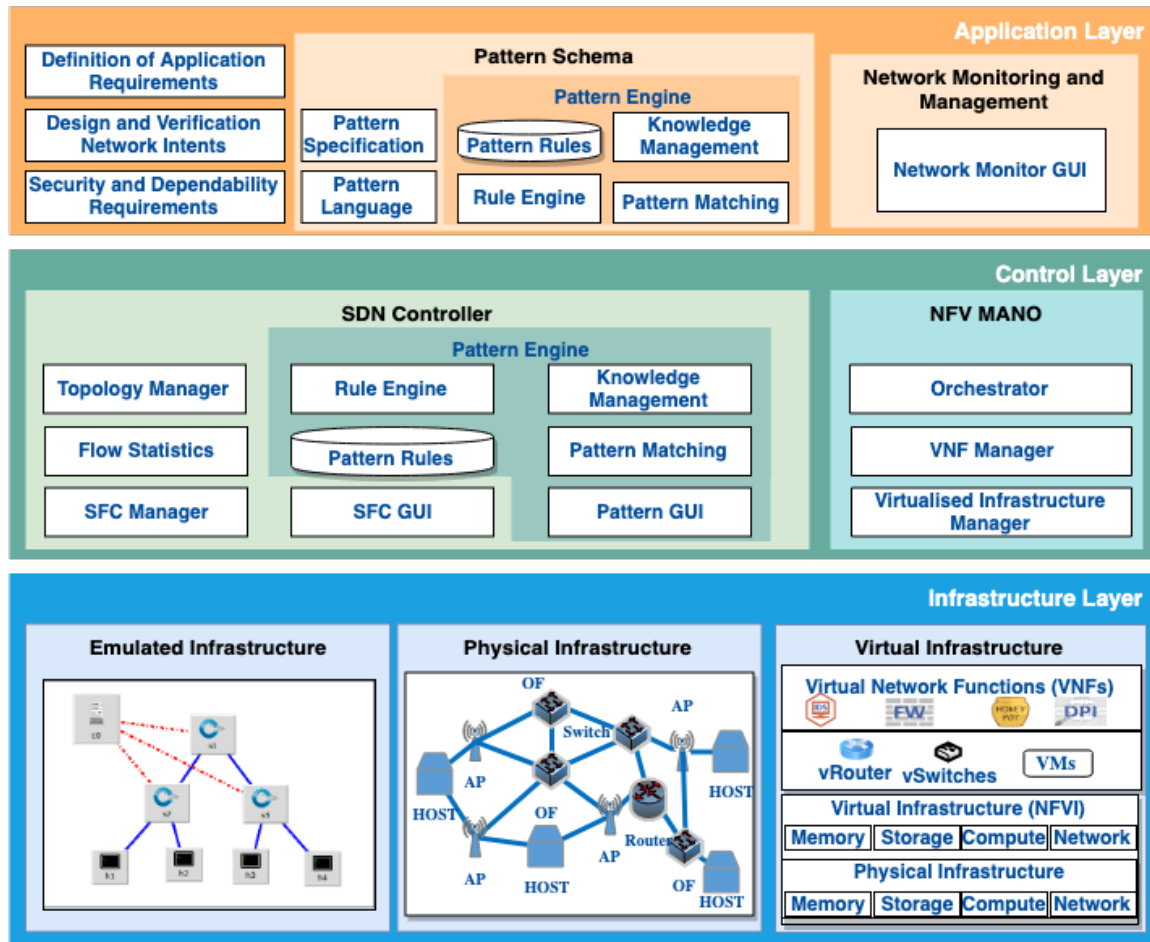


Fig. 5.1 Architecture of the Pattern Framework

5.2 Pattern Framework

The pattern framework is implemented in order to provide a framework able to host and enforce design patterns for the design and verification of network infrastructures. The pattern framework is deployed in both the application and the network layer.

5.2.1 Pattern Schema and Requirements

The insertion of the requirements in the pattern schema to trigger the respective design patterns is one of the most important factors for the design and management of networks. The specification of the different pattern schema and requirements regarding the applications, the design and verification of networks and the S&D management are described in the next subsections.

- **Pattern Schema:** The schema describes the structure of the pattern including the structure of the design patterns into the predefined specification and language. The *Pattern Specification* can be described by a template able to provide the name of the pattern, the problem which tries to be solved, the differences between the proposed solution and the existing solutions. It can provide also the evaluation of the proposed solution and the contributions related to the state of the art. Regarding the *Pattern Language*, this follows the language proposed in Chapter 3. This can translate the property that should be guaranteed by the patterns. Moreover, the structure of these patterns are similar to the previous described S&D patterns.
- **Definition of Application Requirements:** The definition of the application requirements is crucial for the network management and monitoring. This is related to the intent that should be guaranteed by the application. For instance, the sentence *design an IoT network with coverage 100km for deploying monitoring applications* describes the application domain. This can be translated to the placement of a number of different IoT devices able to collect measurements of the covered area. However, each device has specific characteristics, such as range, network interfaces, energy constraints that should also be considered.
- **Design and Verification of Networks:** The design of network infrastructures includes the expression of application requirements in network designs. That involves the placement of network nodes into specific locations in order to guarantee the required properties. In addition to the above, the number of nodes and links is related to the available type of nodes, type of links wired and wireless). Moreover, the network topologies is correlated with the type of application that is required to run upon this network. Finally, the verification of existing networks should also evaluate and monitor the characteristics of network designs on whether they have satisfied the predefined application requirements. This is related to definition of monitoring conditions and elements in order to provide the required network verification and validation.
- **Security and Dependability Management:** The management of network designs requires the definition and provision of predefined S&D properties. This properties can be guaranteed by the determination of specific network designs and by the addition of dedicated elements that satisfy these properties. Furthermore, the addition of specific service function chains in order to forward traffic to specific chains is required for guaranteeing the above S&D challenges.

It is crucial for all the above requirements to be defined not only during the network design phase but also during the validation and runtime phase. Thus, high quality network analysis,

evaluation and procedure may be involved in actual implementation. The use of design patterns in the pattern framework is important to express all the above requirements. In the next subsections, a detailed description of the cross-layer pattern engines, the implemented components and the technology used during the design, integration and deployment of the developed framework are described.

5.2.2 Pattern Engine in the Application Layer

The *Pattern Engine* is based on the Drools rule engine as an enabler to insert design patterns as production rules. Pattern Engine is able to store the pattern rules and conflict the facts from the knowledge management by pattern matching. In this way, it can enable the capability to insert, modify, execute and retract patterns at design or at runtime. The production rules are being stored in the production memory and are used to process data inserted in the working memory (Knowledge Base) as facts by pattern matching (Figure 5.2).

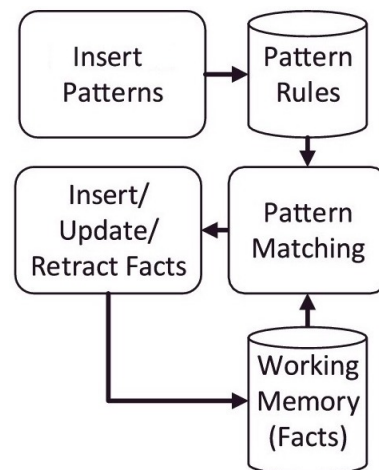


Fig. 5.2 Drools Patterns

Variants of pattern engines can be found at the application and at SDN controller. More specifically, each pattern engine is able to handle different type of components and the satisfaction of specific properties. Compared to the pattern engine in the SDN controller, pattern engine in the application layer is able to instantiate components and handle different network properties. In addition, the pattern engine can be used to design and verify network infrastructure by the insertion of S&D patterns and requirements. More specifically, pattern can be used for designing scalable networks with respect to network pattern topologies. In addition, the pattern engine is able to reason on the S&D properties of aspects pertaining to the operation at design time. Then, the tool uses Drools engine to apply *Pattern Matching*

and identify if a network can be formed out of the available types of nodes that satisfy the required S&D property.

The framework is able to convert facts to network topologies which can be used either to create physical custom topologies or to import in a network emulator as shown in the process diagram in Figure 5.3. Especially in case of exposing custom emulated network topologies to Mininet a converter is required to be created in order to convert pattern engine outputs (Java based facts) to a Mininet understandable format (python based).

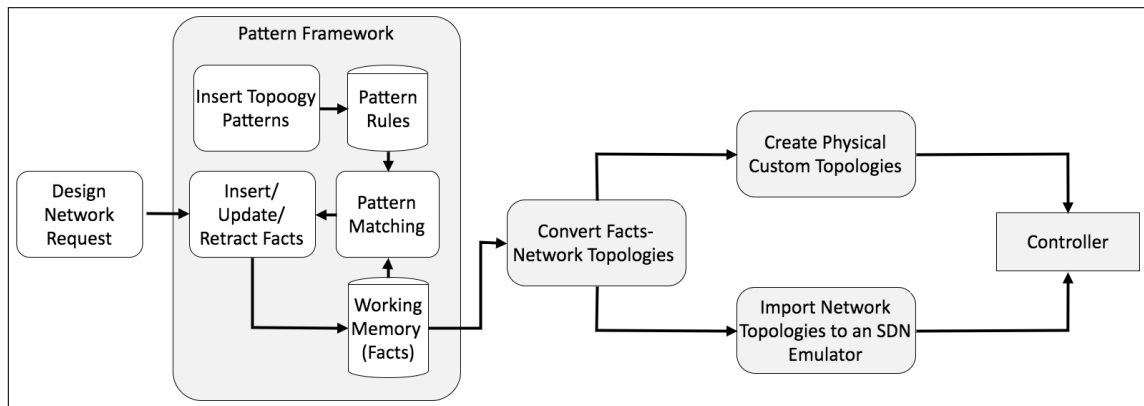


Fig. 5.3 Network Design within the Pattern Framework

Apart from the network designs, pattern engine is also usable to interact with the NFV Mano in order to instantiate VNFs based on the predefined descriptors. This is also related to the instantiation of specific VNFs if requested by the respective service function chains. For runtime adaptation, the pattern engine may receive fact updates from the SDN Pattern Engine, allowing it to have an up-to-date view of the S&D state of said layers and the corresponding components at runtime. This can also involve the collecting of monitoring conditions from more than one controller in inter-domain scenarios.

5.2.3 Pattern Engine in the SDN Controller

ODL can receive application requests and translate the high-level APIs into low-level internal ones as exposed interfaces (e.g. function calls using YANG data structures) or inside in the controller requests as exposed by Java classes. ODL is based on Apache Karaf¹ and uses Apache Maven². Apache Maven is a software management tool that allows developers to define project's build lifecycle (compile and deploy the source code), dependencies, phases, goals, dependencies, build plug-ins and profiles, in order to provide a variety of builds for the

¹<https://karaf.apache.org/>

²<https://maven.apache.org/>

project. ODL exposes its pre-compiled modules as artifacts in the Maven Central Repository. The modular architecture adheres to OSGi specification³, which aids to modules' lifecycle management enabling dynamic state at runtime. For example, the modules can be loaded and unloaded, updated, started or stopped without influencing other running services. At core, ODL solution promotes modular and extensible controller platform design, with the controller instances contained in their own JVM and thus deployable on any Java -supported systems. In OSGi, a module is called bundle, and it is realised as an executable Java Archive (JAR) which contains module metadata such as bundle name, activators, version, as well as information about exported packages and required imports which are references to other module packages. The role of Apache Karaf and Equinox, in ODL project, is to act as containers of OSGi bundles as well as launching the contained bundles on top of its implementation. Finally, ODL supports a number of south-bound protocols such as OpenFlow; this capability enables the interaction with physical infrastructures.

Based on the above, ODL is selected as an open source controller which has attracted a lot of attention in the networking area with more active contributors compared to other open-source solutions [14]. It is chosen to build the pattern framework, as it provides a wide range of abstractions and functionalities that facilitate controller application development and numerous built-in modules that can be either reused or even extended. Furthermore, to implement functionalities in ODL, certain purpose-built modules as well as enhancements to existing SDN controller modules need to be installed and set up appropriately. Based on that, a number of different components were implemented, integrated and used in order to provide network designs with S&D properties guarantees.

In order to support S&D monitoring and management of network infrastructures, pattern engine is also integrated in the controller to enable the capability to insert, modify, execute and retract patterns at design or at runtime. Since pattern engine is based on Maven and is OSGi-enabled, one core part of this framework is the integration of all the dependencies in the ODL controller, as well as the integration of the entities that interact with the controller to run Drools rules at design and at runtime. As ODL does not support all Drools Maven libraries by default, some modifications must be done in order to import the required packages (*knowledge-api*, *drools-core*, *drools-compiler*, *drools-templates*, *drools-decisiontables*).

The verification of an existing SDN network with regards to S&D, is supported by our S&D pattern framework as shown in Figure 5.4. In particular, a designer can *Insert S&D Patterns* in the *Pattern Rules* production memory of the framework and S&D requirements the *Working Memory* as *Facts*. After specifying or importing the network to be verified (*Get Network Topologies/Flows*), S&D patterns are executed to realise the verification process

³<https://www.osgi.org/developer/specifications/>

and new paths can be inserted (*Put Flows*) or current paths can be deleted (*Delete Flows*) or modified (*Post Flows*) in the *Controller* and consequently in the *Programmable Switches*. Through the use of verification patterns, suitable paths can be found in order to pre-plan and reserve paths with respect to S&D properties. In addition, the proposed framework can be used not only for the verification of network paths but also at runtime i.e. following a network link failure or when an S&D property is not guaranteed. Finally at runtime, the framework is able not only to verify a network but also to reconstruct it and restore the required S&D properties in cases where such properties have been violated.

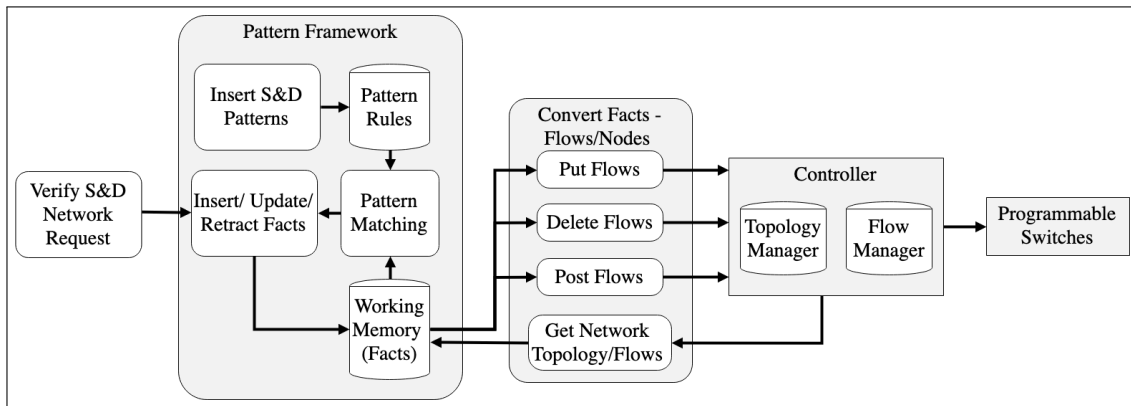


Fig. 5.4 Network Verification and Adaptation within the Pattern Framework

5.2.3.1 Pattern Engine NBI

To support insertion, modification and deletion of facts and rules in the knowledge base by administrators or users, suitable northbound interfaces (YANG APIs and the respective REST APIs) are implemented. Moreover, a number of different YANG interfaces are implemented to interact with the different components including also network components such as switches, service functions and end-hosts, active links and statistics from the controller as required by the pattern rules. In addition, pattern engine is able to import existing network topologies and flows from the inventory list of the ODL. It is capable to export produced OpenFlow rules as generated by the Drools rules. These topologies and flows are imported/exported in a REST/XML format by the use of the NBI interface. Finally, both transformers are developed in Java as part of our framework.

5.2.3.2 Pattern GUI

The Pattern GUI module is implemented in the SDN Controller as an additional module on the ODL controller to monitor, manage and assess the proof of concept implementation of

the SDN Pattern Framework. Pattern rules can be inserted as a plain text or as an external file in Json format. Suitable Javascripts and html files have been implemented in order to support the insertion of patterns either at design or at runtime. More precisely, angularJS⁴ is used as a Javascript-based library for front-end web application. The use of predefined pattern/rule templates is considered to be enhanced as a more convenient way to manage patterns. The imported patterns are previewed automatically in an interactive table as presented in the Figure 5.5. Appropriate APIs have been implemented in order an administrator to be able to insert, modify, delete and enable/disable imported rules using the implemented controls.

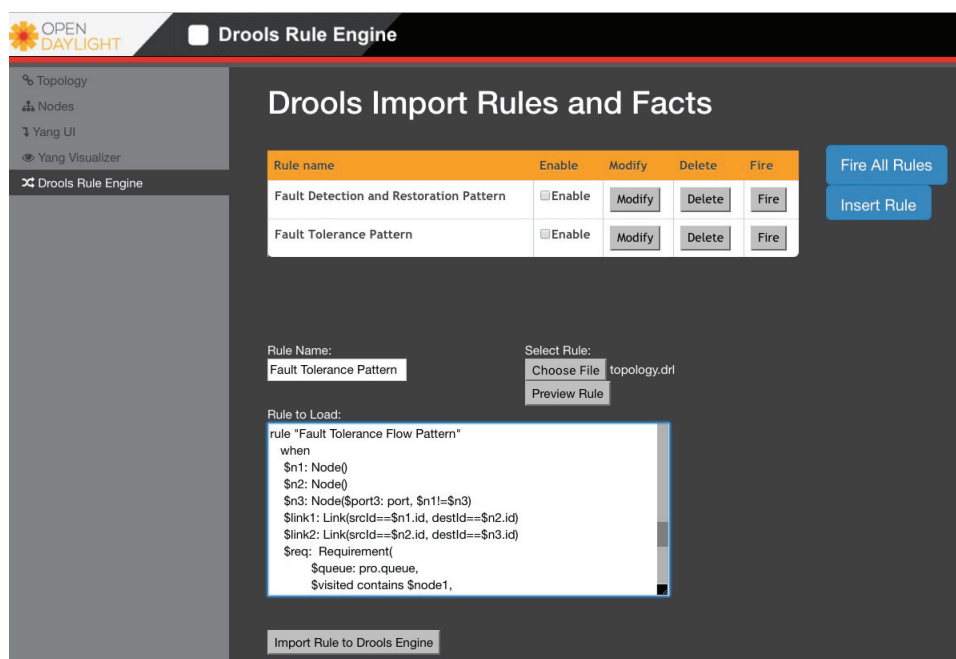


Fig. 5.5 OpenDaylight SDN Pattern Framework GUI

5.3 SFC Reactive Security

The implementation of reactive security leveraging SFC is one core topic of this work. The SFC reactive security is able to forward the traffic to the respective chains. This chains may consist of the instantiated virtual network functions as instantiated by the respective patterns deployed in the developed pattern framework. Moreover, the interaction with the network is also available. In addition, the traffic traversing said network, e.g. to automatically mitigate attacks, block malicious entities, route them to specific, dummy network components can be done through through the SFC. Furthermore, the allowance for enhanced monitoring or

⁴<https://angularjs.org>

even trigger the deployment of new security functions alleviate the effects of an ongoing attack. By leveraging the flexibility of SDN-based deployments and the concept of SFC, a service-specific overlay creates a service-oriented topology, on top of the existing network topology, thus providing service function interoperability.

The aim of this service chaining is to overcome constraints and inefficiencies, as mentioned previously. This can be used to fulfil the target of providing security profiles per application classification based on the originating application. The per tenant classification serves multiple virtual tenant networks with the chaining of vital security functions. Moreover, the per traffic classification, for both intra- and inter- domain deployments, can forward the traffic following predefined service function paths for each traffic type. By instantiating of security network functions such as Firewalls, IDS, DPI, Honeypots and Honeynets, a number of service function chains can be created to forward traffic based on the type of traffic or running application.

Important parts of the implementation of this functionality are the Classifier and the DPI service function. The Classifier is responsible for classifying and forwarding incoming packets based on predefined rules, exploiting pattern matching and tags found on the packet headers, and forwards the packets through one of the predefined function chains. In the case of packets already carrying a tag classifying, the traffic is forwarded via the associated chain providing faster packet transmission. In case of a malicious type of packets, the classifier will forward the packets to the honeypot (or honeynet, depending on the deployment), to isolate and investigate the attack. When there is no previous acquired knowledge about the packet's classification (i.e. no tag on the packet header), the classifier will forward the traffic to the nDPI aiming to detect any malicious activity, assess its impact, and attach the associated tag, to help form the system's response and enhance the attack mitigation effectiveness. The nDPI disassembles the traffic packets, assesses their content and decides on their traffic type. Then, the packet is repackaged by assigning the appropriate headers to allow for its routing through the corresponding service chain. However, even if this chain protects SDN network from malicious attacks, the procedure will add delay to the transmission.

The reactive security is based on the following components:

- **Virtual Network Functions**
- **SFC Manager** The SFC Manager to stitch security service functions
- **SFC GUI** The SFC GUI to monitor and manage the reactive security
- **Dynamic SFC instantiation**

5.3.1 Virtual Network Functions

The preparation of an incident detection and response mechanism contains a generic incident handling of a reactive security for cyber-physical system. Additionally, the incident response, vulnerability and artefact handling include analysis, support and coordination. In the same way, the protection, detection and response are a combination of monitoring and incident detection, mitigation and trace-back and audit mechanisms. One of the goals of this effort is to provide a secure industrial networking infrastructure, via the associated security mechanisms, such as network monitoring and intrusion detection for industrial SDN networks. To achieve this objective, the reactive security presented herein includes a number of different VNFs for continuous network monitoring and intrusion detection for identification of attacks and run-time network adaptation for attack response and mitigation mechanisms. The placement of exemplary security VNFs in the NFV architecture is depicted in Figure 5.6. That includes the implementation of the following virtual switches and the proactive service functions:

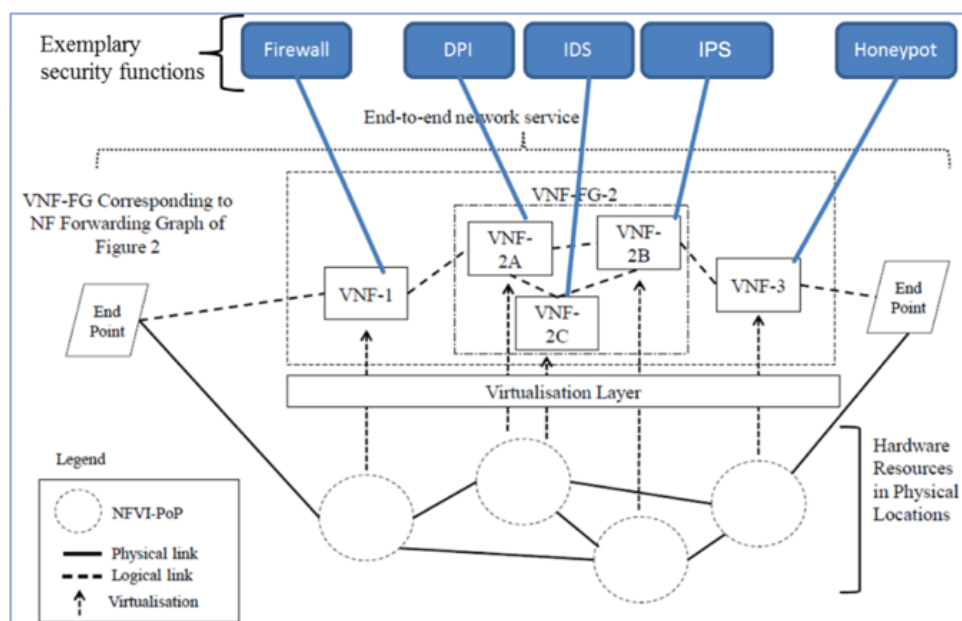


Fig. 5.6 Security Functions in NFV Architecture

5.3.1.1 IDS and SCADA IDS

The security mechanisms include continuous network monitoring and intrusion detection for identification of attacks and run-time network adaptation for attack response and mitigation mechanisms. More specifically, IDS instances of Snort [177] are deployed, with scripts to ensure that the most up-to-date rules are constantly active. A database for event monitoring

is present, while provisions are made to allow for future extensions to transmit relevant information to a security backend (e.g. for more sophisticated pattern matching). Moreover, a SCADA-specific instance of Snort [178] is deployed, where SCADA traffic is routed. This limits the delay imposed on the SCADA traffic by the IDS functionality (a delay that significantly depends on the number of rules/patterns in the IDS's database, which is significantly lower in the case of the IDS with only SCADA-specific rules installed).

5.3.1.2 Honeynet

Network-based Honeypots have been widely used to detect attacks and malware. A Honeypot is a decoy deployment that can fool attackers into by making them believe that they are hitting a real network whereas in the same time it collects information about the attacker and attack method. A Honeynet contains a set of functions, emulating a production network to attract and detect attacks, acting as a decoy or dummy target. In the protected wind park network, a Honeynet can consist of Honeypot network elements, as well as emulating operational systems of the wind park and more specifically elements, such as the SCADA systems and data historian. Simple Honeypots [179] and SCADA-specific Honeypots [180] are deployed to emulate the exact network and SCADA system setup present in the SDN-enabled wind park. Moreover, passive Honeypots (Early Warning Intrusion Detection Systems, (EWIS), in specific [181]) are part of the Honeynet, acting as a network telescope on the production part of the industrial network, to monitor all activity in normally unused parts of the network. Such activity is a good indicator of malicious entities operating on the network (such as an attacker probing/foot-printing the network), thus providing early warning of incoming attacks.

5.3.1.3 Firewall

A software or hardware firewall instance is also deployed on the wind park's network use case to implement network perimeter security. This is a software firewall (instance of pfsense [182]), but a hardware (legacy) firewall appliance already present in the industrial network or even a virtualised commercial firewall appliance (such as the VM-Series from Palo Alto [183]) could also be used. The type of firewall, as well as its placement, is irrelevant to the context of the reactive security employed to protect the industrial network, as the service plane view of the framework focuses on the type of service and not the underlying technology. This allows the use of any type of firewall and placement in any place on an SDN network deployment.

5.3.1.4 DPI

As a proof-of-concept implementation of a DPI function, nDPI [184] is employed to monitor incoming traffic and to forward it to the corresponding service chain. In order to have an up to date view, the SFC-related information is fetched from the ODL controller via constantly running scripts (e.g. information about the various chains or information on which chain IDs correspond to reverse chains, i.e. return traffic). Firstly, the NSH header is extracted to check if the packet is already processed (in this case, the packet chain ID would be that of a reverse chain). If it is already processed, it is simply forwarded to the attached forwarder. If the packet has not been processed before, the developed application encapsulates the packet and sends it to the *nDPI engine* for processing. As soon as a response is received from the nDPI engine, the appropriate chain IDs and the appropriate next hops from rendered service chains are fetched from the controller. Based on the received information from the controller, the NSH header is generated and the packet is encapsulated appropriately, before being forwarded to the appropriate SFC to support dynamic packet flow change. The response of the *nDPI engine* is based on a set of rules that the engine has compiled to classify traffic types, and can be extended by writing additional rules (for instance TCP and UDP ports 502 associated with Modbus traffic can be defined as being malicious, if such traffic is not expected in the specific part of the network). The response from the *nDPI engine* classification of each packet is either the protocol/application/framework ID that the above mentioned rules define, or *UNKNOWN* if it cannot be determined.

5.3.1.5 Open vSwitch

Finally, the last required virtual component is Open vSwitch (OVS) which is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). In addition, it is designed to support distribution across multiple physical servers similar to VMware's vNetwork distributed vswitch or Cisco's Nexus 1000V ⁵. Finally, OVS Switch is able to support IPSec encryption for the tunnel traffic to prevent the traffic data from being monitored and manipulated ⁶.

⁵<https://www.openvswitch.org>

⁶<http://docs.openvswitch.org/en/latest/tutorials/ovn-ipsec/>

5.3.2 SFC Manager

An SFC determines an abstract set of service functions and their ordering constraints that should be applied to packets and/or frames selected as a result of classification. Service instances may include Firewall, IDS, DPI, and HoneyPot. These services may be the physical appliances or virtual machines running in network function virtualisation infrastructures. They may be composed of one or multiple instances. Each SFC configuration includes a set of service nodes, a set of service functions, a set of service function forwarders, a set of service chains, a set of service paths and a set of configurations for classifiers (ACL/NSH). If the Service Function Nodes are not OpenFlow-speaking or SFC-aware, or are in different domains, SFC Proxies are needed.

To implement the above functionality, certain purpose-built modules as well as enhancements to existing SDN controller modules are needed. The SDN controller programs the underlying forwarding elements that do the actual packet forwarding. In essence, the SDN Controller is converting commands from the high-level SFC language to the low-level flow filters expressed in the OpenFlow semantics. The SDN Controller provides an abstraction view of the network topology. This significantly simplifies the configuration. However, the SDN Controller does the necessary transformations to put the paths (sequence of service instances where the packet traverses) and filters (associate user based on his profile to its respective service chain) in the forwarding devices (OF-enabled).

The SFC Manager is able to handle service function chaining of network functions. This allows the SFC Manager to focus on the chaining itself and not on the internal topology of the Network Controller. This means that SFC Manager manages forwarding rules and flow filters on external ports. In particular, the job of SFC Manager is to register external ports of the SDN transport network (which is being used for SFC) and to declare and associate service instances to the external ports. In more detail, the SFC Manager controller module exposes a number of interfaces. The various components can use the interfaces to provide and receive information about service chains that need to be built. This may include which tenants want to use them, which destinations are being accessed, what applications the traffic pertains to and about the service instances of the network functions. The SFC Manager aggregates this information, combines it, and sends service chains in commands to the SDN Controller.

At the Management and Control planes, the SFC Manager and the SFC-enabled SDN controllers are responsible for administrating the services chains, i.e., for mapping the operator's/tenant's/ application's requirements into service chains. At the Data plane, Classifiers assign traffic to their intended service chain (based on pre-defined criteria) and Service Forwarders and Proxies (where needed) are responsible for steering traffic accordingly, in order to realise said Service Chains. In the windpark case, such service instances may include

vFirewall, IDS, DPI, and Honeypot. These services may be composed of one or multiple instances. These may be the physical appliances or virtual machines running in NFVI. Finally, the syntax of the respective SFC files and the templates are presented in Table 5.1

Table 5.1 Abstract SFC Component Structure

Service-nodes	Syntax
Service Function (SF)	"service-function": [{ "name", "ip-mgmt-address", "rest-uri", "type", "nsh-aware", "sf-data-plane-locator": [{ "name", "port", "ip", "transport", "service-function-forwarder" }] }]
Service Function Forwarder (SFF)	"service-function-forwarder": [{ "name", "service-node", "service-function-forwarder-ovs:ovs-bridge": { "bridge-name", "sf-data-plane-locator": [{ "name", "port", "ip", "transport", "service-function-forwarder" }] }, "service-function-dictionary": [{ "name", "sff-sf-data-plane-locator": { "sf-dpl-name", "sff-dpl-name" } }]
Classifier	"service-function-classifier": [{ "name", "scl-service-function-forwarder": [{ "name", "interface" }], "acl": { "name", "type" }]
Service Function Chain	"service-function-chain": [{ "name", "symmetric", "sfc-service-function": [{ "name", "type", { "name", "type" } }
Service Function Path	"service-function-path": [{ "name", "service-chain-name", "starting-index", "symmetric", "context-metadata", "service-path-hop": [{ "hop-number", "service-function-name" }]

5.3.3 SFC GUI

To assess and manage the proof of concept implementation of the Reactive Security, a Graphical User Interface (GUI) is developed, as an additional module on the ODL SDN Controller. The GUI displays instantiated VMs/Service Chains and traffic paths, based on the chains seen in the bottom of Figure 5.7. Based on this classification, SCADA traffic goes to the SCADA IDS and then to its intended SCADA system at the wind park. HTTP traffic goes to normal IDS and then to its intended system at the wind park. Malicious traffic (e.g. nmap port scan) is detected and goes to Honeypot/Honeynet instead of its intended target wind park

system. Finally, unknown traffic is routed to DPI for classification, where a modification in the header of the packet, can forward the traffic to the respective active chain (legitimate, SCADA or malicious). One additional capability of the implemented GUI module depicts realtime network traffic monitoring interfaces and functions and Service Function resource monitoring interfaces and functions custom of the imported data.

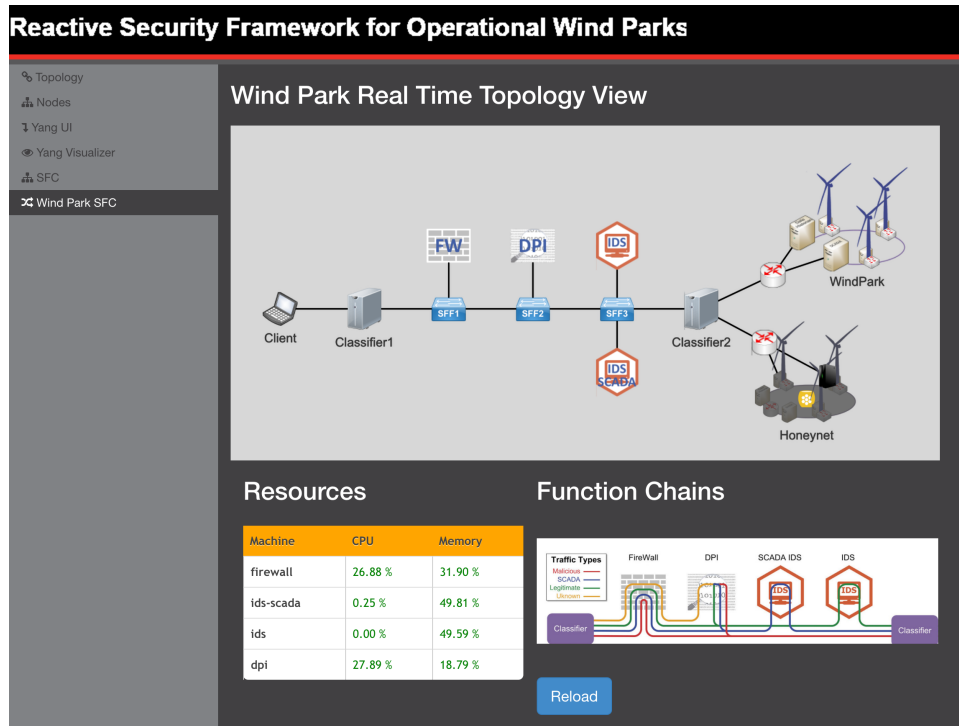


Fig. 5.7 Reactive Security SFC GUI

To preview the topology of the network, nodeJS library [185] is used to present network topology at real-time. Suitable REST interfaces were implemented to import network components such as switches (i.e. forwarders and classifiers), security functions (FW, DPI, IDS and SCADA-IDS) and end-hosts (i.e. windparks, scada servers). Moreover, the condition of service functions with respect to CPU and memory utilisation of the various security service functions (i.e. the VMs running said service functions), is imported automatically by the use of implemented REST interfaces presented in real-time on a separate table.

5.3.4 SFC in the NFV MANO

In today's legacy network infrastructure security-related functionalities and applications are running on dedicated locations in the network architecture. Nowadays, these locations have to be decided in the network planning phase and are very difficult to change during

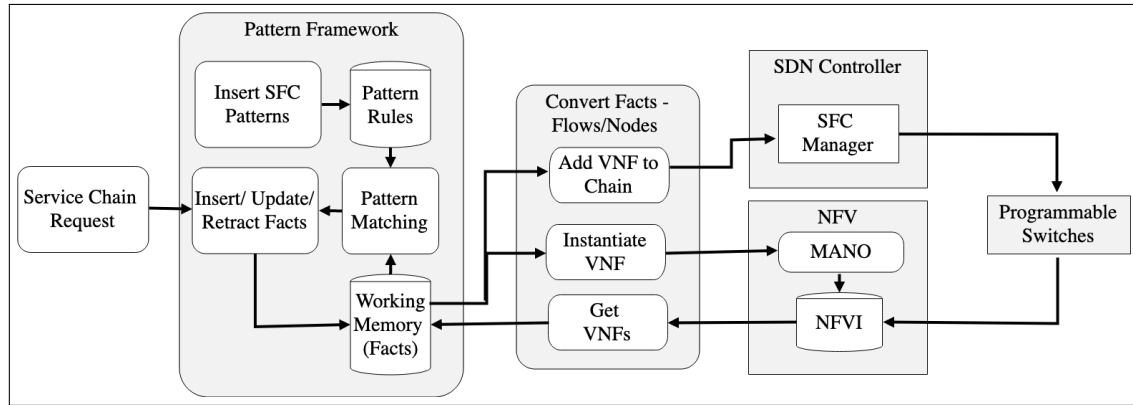


Fig. 5.9 SFC Requests in the Pattern Framework

5.4 Network Simulator

To evaluate the proposed pattern framework, it was necessary an open and extensible environment for deploying the design networks and its integration with the pattern network. Therefore, to assess and manage the proof of concept implementation of the pattern framework, a network simulator is developed in order to monitor and manage network infrastructures deployed by the pattern framework and used by the reactive security. The main scope of the simulator is to present existing or designed topologies including different kind of nodes and links. In addition, the simulator is able to depict and update network topology at design or at runtime. The structure of the simulator includes different deployed and implemented parts.

- **Simulator Library:** The simulator is based on the `vis.js`⁷ visualisation Javascript library. `Vis.js` is able to handle dynamic data in order to enable data manipulation and interaction. Different type of components are included in the `Vis` library such as `Network`, `DataSet`, `Timeline`, `Graph2d` and `Graph3d`. For the development of the network simulator, the `network` library is used to display networks, consisting different nodes and link interactions. It supports custom shapes, images, styles colors and clustering for large number of nodes.
- **Simulator Functions:** To be able to develop the simulator to fulfil the framework requirements, a number of different additional Javascript functions are introduced. Javascript functions are developed to enable the import and export of the network topology files externally. Furthermore, the addition of functions to support the additional required attributes (ie. resources, ports resources etc.) of nodes for the different import and export of topologies are also developed. Moreover, the capability to interact

⁷<http://www.vis.js>: A dynamic, browser based visualisation library.

with the depicted in the simulator nodes are included. These functions enable the different component interaction with the mouse (ie, click, doubleclick, leftclick) the node conditions such as address and resources can be previewed. Apart from the basic functions, some other capabilities are introduced such as zoom in/out and move/drag of the nodes.

- **Network Component Descriptions:** Apart from the different functions, to express all the required components for the design of physical and network infrastructures, a number of additional components are included in the description file. Each of the below nodes are inserted as different groups enabling the depiction of different nodes having a unique image. More specifically the following type of nodes are defined:
 - **End Hosts:** client, host, server, controller historian, scada, windpark, printer, pda, laptop, internet, network
 - **Forwarding Devices:** router, switch, accesspoint, forwarder, classifier, servicenode
 - **Service Functions:** function, firewall, dpi, ids, scadaids, loadbalancer, honeypot, honeynet
- **Network Topology Json Format:** All the deployed network topologies should be formatted in a suitable Json format as required in order to be inserted and depicted in the network simulator. The structure of the Json files following the format of:

$$\{ "data" : \{ "nodes" : [\{ \dots \}, \dots], "edges" : [\{ \dots \}, \dots] \} \}$$

As an example of a Json network topology, Figure 5.10 depicts the expression of a interconnection between a client and a server. Both group of nodes are introduced previously where the additional required information for the deployment of the proposed topologies such as address, resources, etc. are also presented.

- **Simulator GUI:** The GUI of the simulator is developed by the creation of an HTML web page as a canvas for rendering the network topologies. This canvas enables the capability for runtime interaction between the user and the web page. In the Figure 5.11, the variety of the previously defined network elements supported by the simulator is presented. The simulator provides the capabilities to import network topologies from a file or to export them to a file by the use of the deployed buttons and the developed functions. Moreover, the addition of a node on the fly is possible. Some additional options are enabled to provide easier interaction with the output of the simulator, such


```
1 {
2   "data": {
3     "nodes": [
4       {
5         "id": 1,
6         "label": "Client",
7         "group": "client",
8         "address": "192.168.1.1",
9         "ports": 80,
10        "position": ["x": 30, "y": 40],
11        "resource": ["cpu": 30, "mem": 40, "storage": 40, "energy": 50]
12      },
13      {
14        "id": 2,
15        "label": "Server",
16        "group": "server",
17        "address": "192.168.1.2",
18        "ports": 80,
19        "position": ["x": 60, "y": 60],
20        "resource": ["cpu": 20, "mem": 50, "storage": 30, "energy": 40]
21      },
22    ],
23    "edges": [
24      {
25        "from": 1,
26        "to": 2,
27        "label": "Client to Server",
28        "width": 1,
29        "color": ["color": "blue"]
30      }
31    ]
32  }
33 }
```

Fig. 5.10 Network Topology Json Example

as arrows to move the topology or zoom in/out properties. Finally, the simulator is able to update the topology of the network based on the latest received information stored externally in a file since the auto refresh option is enabled.

- **Pattern Framework Interaction:** The last part of this development includes the interaction between the pattern framework and the simulator. To fulfil the initial

Network Topologies.

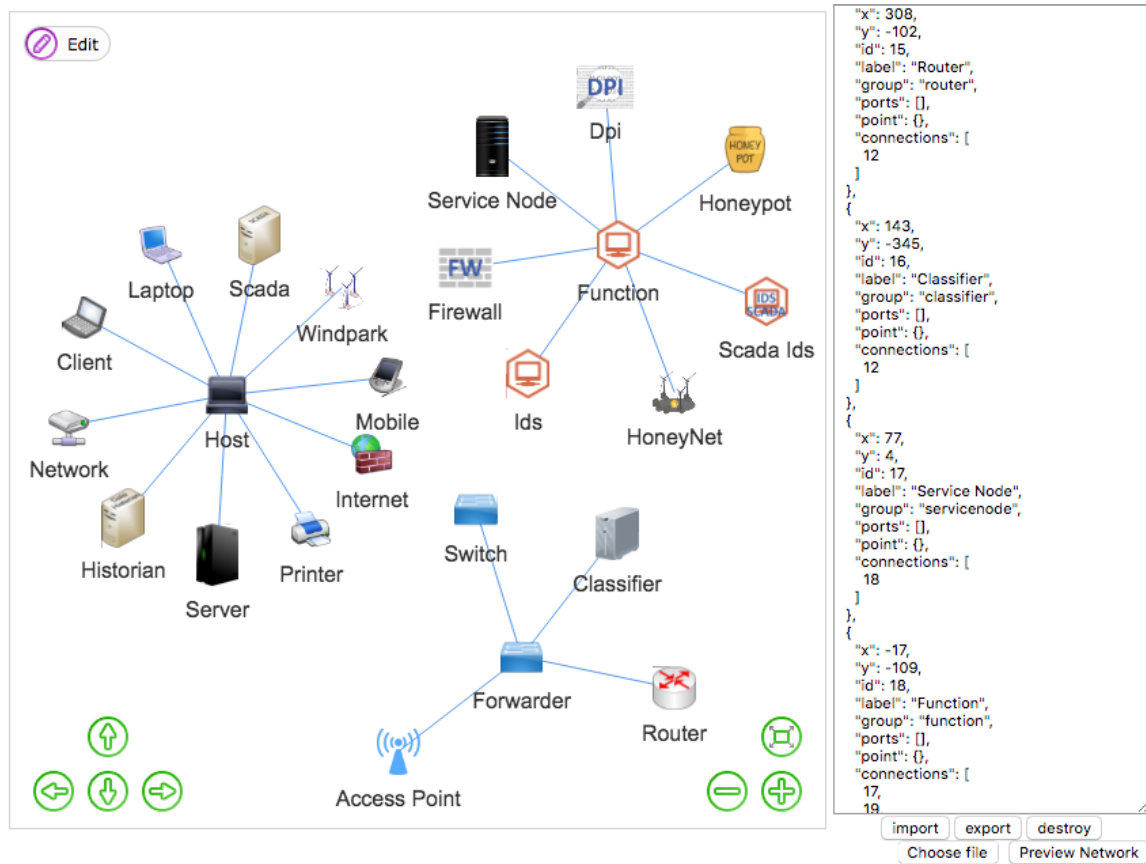


Fig. 5.11 Implemented Network Simulator

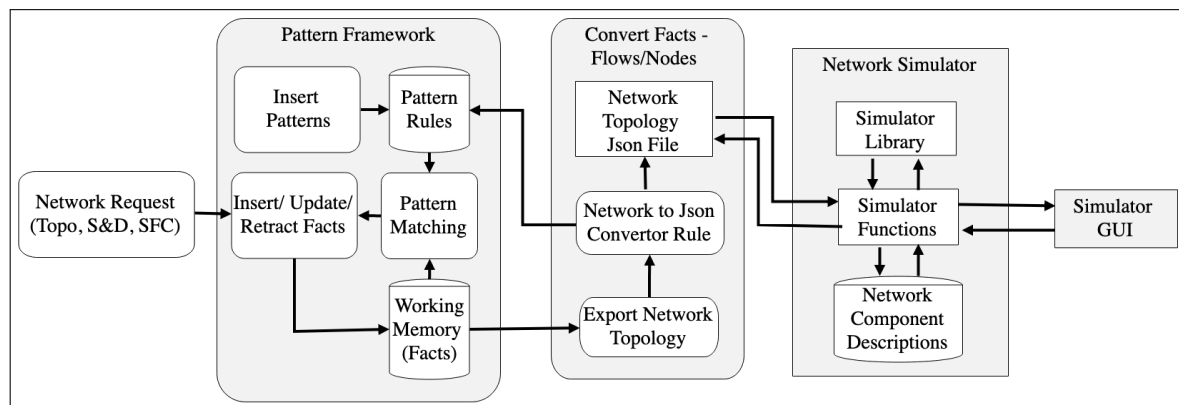


Fig. 5.12 Network Simulator and Pattern Framework Interaction

requirement for runtime representation of the designed or updated network topologies as retrieved by the pattern framework, suitable rules are developed to enable this capability. For instance the initial design of a network including nodes and links has

to be converted in the Json format as required by the simulator. The pattern engine is applied to make this conversion by the development of suitable rules for this reason. The converted network topology format is exported automatically as a Json file able to be inserted in the network simulator. The identification of suitable paths, service chain paths and secure paths as exposed by the pattern framework can be depicted by enabling the modification of the default color of the respective links. The procedure of the interaction between the pattern framework and the simulator is depicted in Figure 5.12

5.5 Testing and Evaluation Environment

5.5.1 Emulated Infrastructure

Network emulators such as Mininet⁸ platform are suitable for testing and emulation. Especially by the use of Mininet-WiFi [187] and NS3⁹, it is possible to include not only switches and hosts, but also OpenFlow-enabled access points. In the proposed framework, Mininet is used to build network topology based on the predefined patterns. Moreover, with the extension of Mininet-WiFi the possibility to extend the proposed scenarios for hosting OpenFlow-enabled access points is also examined. In addition, the capability of Mininet to create custom network topologies based on the predefined configuration files is applied.

5.5.2 Virtual Infrastructure

Virtual infrastructures represent a core part of NFV architectures. The possibility to instantiate Virtual Machines (VMs) based on the use case requirements, emphasise also the great capability of virtualisation. To instantiate VMs running vSwitchs, security VNFs and SDN controller, a hypervisor is required. Hypervisor is a computer software or hardware component that is able to create and run VMs. There are a number of hypervisors such as:

- VirtualBox VM¹⁰ is a free and open-source hosted hypervisor for x86 virtualisation.
- Proxmox Virtualisation Environment¹¹ is an open source server virtualisation management software.

⁸<https://www.mininet.org>

⁹<http://www.nsnam.org/>

¹⁰<https://www.virtualbox.org>

¹¹<https://www.proxmox.com>

- OpenStack¹² is an open-source software platform for building private and public cloud.

In the context of this work, all the above platforms are evaluated in order to instantiate VNFs for the different SFCs. VirtualBox was used initially to experiment locally the concept of SFC. However, the resource requirements made necessary to develop the reactive security in a dedicated machine running native Proxmox. Therefore, Proxmox is used as the standard hypervisor to run static topologies, where the Openstack is used as an experimental platform to apply dynamic topologies in NFV environment. In Figure 5.13 a screenshot of the tested hypervisors (Proxmox and OpenStack) is presented.

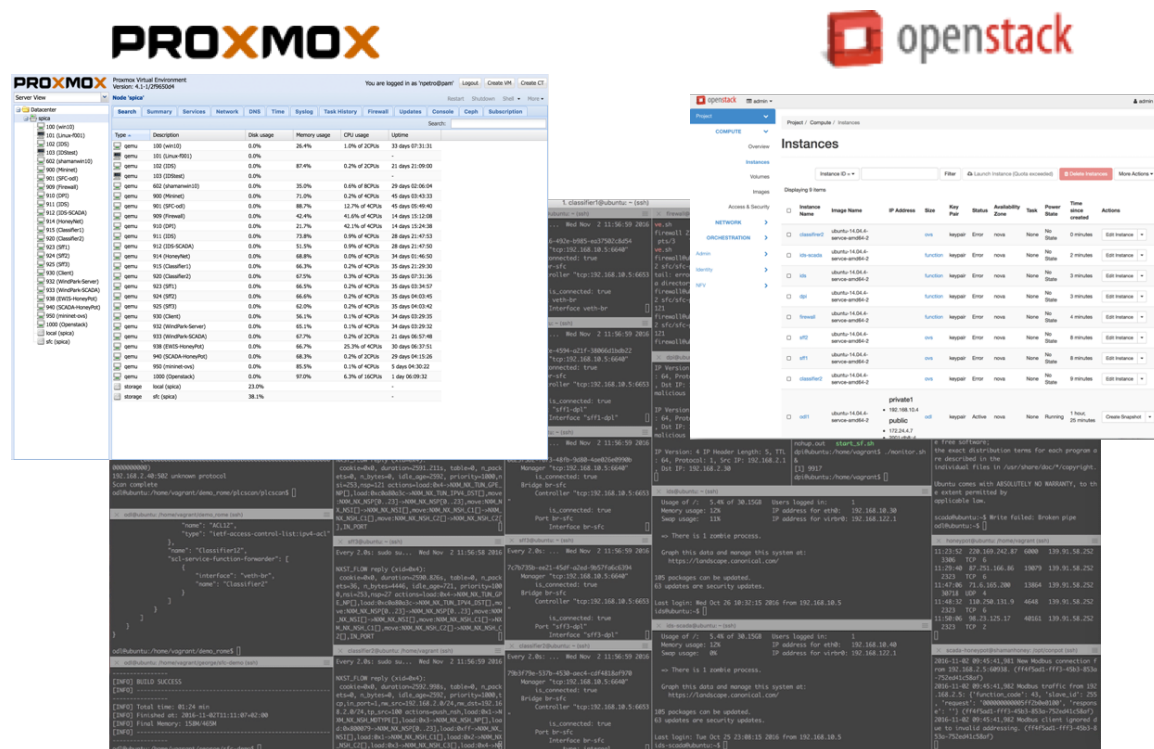


Fig. 5.13 Hypervisors

The reactive security through SFC is developed in an experimental testbed. The testbed featuring multiple VMs is developed and deployed on Proxmox which runs on a server system (featuring 2 x Intel Xeon E5-2630 v2 6-core/12-thread CPUs, at 2.6GHz, with 32GB RAM). The following VMs are required in order to implement the described scenario; three different types of virtual instances were created (in parentheses the resources dedicated to each VM):

- **Controller:** 1x OpenDaylight Controller instance (Boron release (4 CPU cores, 4GB RAM))

¹²<https://www.openstack.org>

- **Forwarding Devices:** 5x Open vSwitch [188] (v2.59) instances (2 x Classifiers (4 CPU cores, 1GB RAM), 3 x Service Function Forwarders (4 CPU cores, 1GB RAM))
- **Service Functions:** 4x Security Service Functions: 1x Firewall instance (4 CPU cores, 2GB RAM), 1x DPI, i.e. the custom nDPI-based implementation (4 CPU cores, 4GB RAM), 1x Snort-based IDS with all generic rules (4 CPU cores, 1GB RAM), 1x Snort-based SCADA IDS, with SCADA-only rules (4 CPU cores, 1GB RAM)
- **End-Hosts:** 4x End-hosts: 1x Emulated Data Historian (4 CPU cores, 1GB RAM), 1x Emulated SCADA system (4 CPU cores, 1GB RAM), 1x Passive EWIS Honeypot (4 CPU cores, 1GB RAM), 1x SCADA Honeypot (4 CPU cores, 1GB RAM)

5.5.3 Software Setup

Finally, for the evaluation and testing different software tools were used:

- **Eclipse Modelling Tool**¹³ with the JBoss Drools is used to evaluate and test drools rules. Eclipse is used to develop the Java classes as required by the definition and the specification of under-development modules and blocks.
- **Git**¹⁴, is a tool that provides versioning and parallel development of source code. The developers can have access to all versions of the files and exchange code modifications via a git server. When a developer makes local changes to the code, these changes can be pushed onto the git server. Other developers are able to pull these modifications from the server. As git keeps track of all the modifications done to each and every file, line by line, it is possible to retrieve any previous versions of the code at any time.
- **Postman**¹⁵ is a powerful GUI platform for testing building APIs and sharing REST API calls exposed by ODL modules.
- **JBoss Drools** is used to express patterns and the respective Java classes in the rule Engine.

5.6 Summary

In this chapter, the development of a pattern framework able to handle faults and failures in SDN/NFV-enabled network infrastructures was presented. The flexibility of the framework

¹³<http://www.eclipse.org>

¹⁴<https://git-scm.com/>

¹⁵<https://www.getpostman.com>

to insert patterns as Drools rules in the pattern engine, shows the capability of the controller to guarantee properties and handle incidents. The cross layer distribution of the different pattern engines in the application and in the SDN controller is described. Especially in the pattern engine developed in the SDN controller, there is no need to modify internal controller modules since the interaction is applied through pattern rules. Thus, the framework can be easily extended to interact with multiple controller capabilities and functionalities. In addition, the reactive security leveraging SFC and the integration with the pattern framework in order to provide traffic classification through different security network functions was also presented. Finally, the developed network simulator and the testbed setup description was given.

Chapter 6

Evaluation of Design Patterns in the Pattern Framework

6.1 Overview

To evaluate the applicability, usability and performance of our patterns on the pattern framework, three different main use cases are proposed each of them containing different usage scenarios as depicted in the Figure 6.1.

- **Design Networks:** The first use case includes the design of different types of network topologies based on the proposed topology patterns able to offer connectivity, coverage and scalability.
- **Design and Verify S&D Network Topologies:** The second use case includes the design and verification of existing network topologies with respect to different network properties such as reliability, fault tolerance and security.
- **Service Provisioning:** The third use case includes the application of the pattern framework to offer reactive security leveraging the SFC concept in a actual windpark use case to ensure service provisioning and chaining.

These use cases and the outcomes of the evaluation are presented in the following subsections.

6.2 Design Network Topologies

One of the main goal of network designers is to provide network topology designs able to satisfy different application requirements and infrastructure properties. This may include the

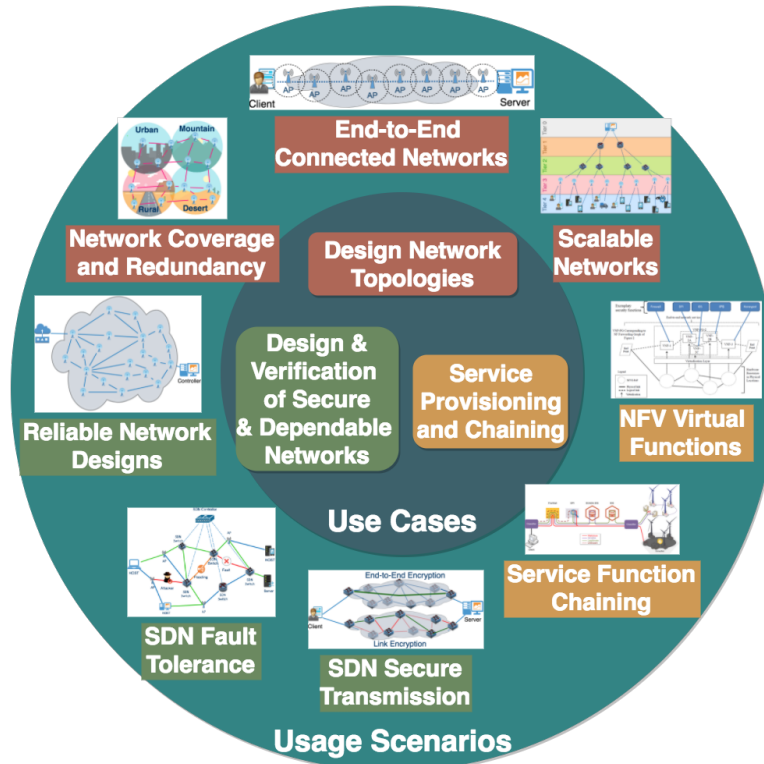


Fig. 6.1 Evaluated Use Case and Scenarios

number of users, the covered area, the degree of redundancy, the type of network technology or finally the available resources.

The first use case focuses on the design of network topologies based on the previously defined topology patterns. After the execution of these topology patterns under the pattern framework, network design are available either for physical deployment or under an emulated entronement for experimenting reasons. The three different network topology patterns (line, mesh and tree) can be used for the evaluation of three usage scenarios that can demonstrate the existing capabilities of the framework as provided below:

- End-to-end connected network topologies based on the *Line Pattern*.
- Maximum coverage and redundancy network topologies based on the *Mesh Pattern*.
- Designing scalable network topologies based on the *Tree Pattern*.

6.2.1 End-to-End Connected Network Topologies

The first usage scenario of network topology includes the design of a wireless network able to guarantee the *connectivity* property between two end-points. In this scenarios, the number

of intermediate nodes is related to the maximum distance between the nodes. This distance is also related to the range of each node as depicted in Figure 6.2.

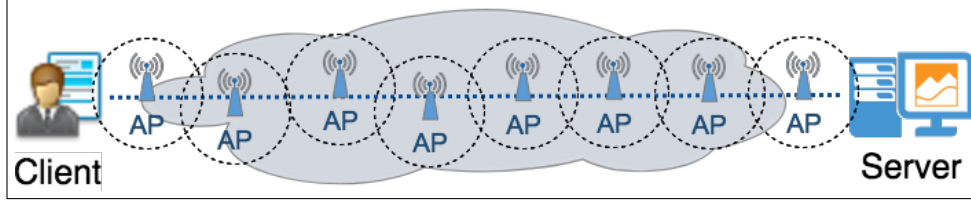


Fig. 6.2 End to End Connected Networks

The main scope of this scenario is to find the number of nodes between source and destination. The role of the pattern framework in this usage scenarios, involves the design of a connected E2E network able to satisfy the connectivity property based on the execution of the *Line Pattern*.

Table 6.1 Experimental Results of the Line Pattern Execution

Distance (metres)	Required Relay Nodes	Exec. Time (msec)
500	4	44
1.000	9	58
2.000	17	60
5.000	33	85
10.000	65	101

To evaluate the line pattern, different requirements are inserted in the working memory following the syntax: $Req(src, dst, new Pro("connectivity", d), new Range(r))$, where src is the source node, dst is the destination node, $Pro("connectivity", d)$ defines the required *connectivity* property as guaranteed by the *Line Pattern*, d is the distance between src and dst , and r is the constraint that defines the maximum distance (range) between nodes in order to establish connectivity. Different requirements, expressing a variety of distances d between a src and dst are inserted in the working memory. Line pattern is executed based on the inserted requirements for defining the required number of relay nodes for five different distance values (500m, 1km, 2km, 5km, 10km) between src and dst with maximum range of 100m between nodes. After the execution of the line pattern in the pattern framework, the experimental results are presented in Table 6.1. Based on the different distances of src and dst (column 1), the minimum number of required relay nodes (column 2) is presented. Moreover, the execution times of line pattern to solve the design problem, are presented in column 3. As expected and verified by the results, the execution time is increased as the distance being increased. In addition, the number of relay nodes, necessary for preserving

the required connectivity property, is growing having as a result the CAPEX of such network design deployment. Finally, together with the experimental results, Figure 6.3 depicts the constructed network for the distance of 1 km in the developed network simulator.

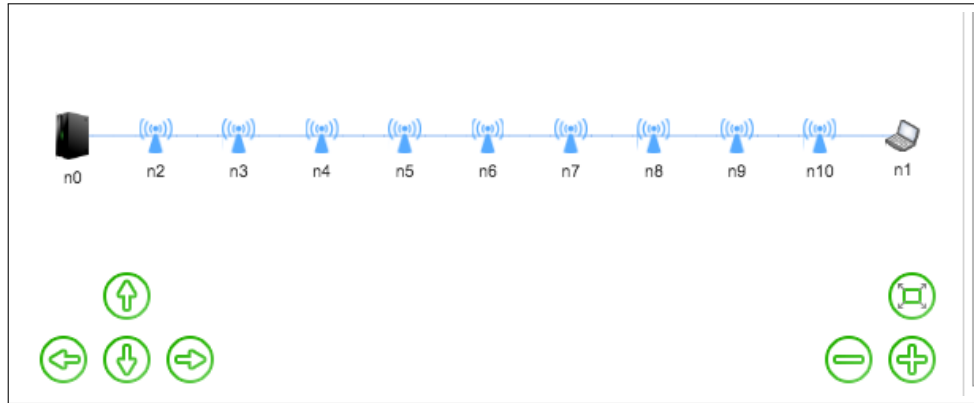


Fig. 6.3 Line Pattern Output on Network Simulator

6.2.2 Maximum Coverage and Redundancy in Network Topologies

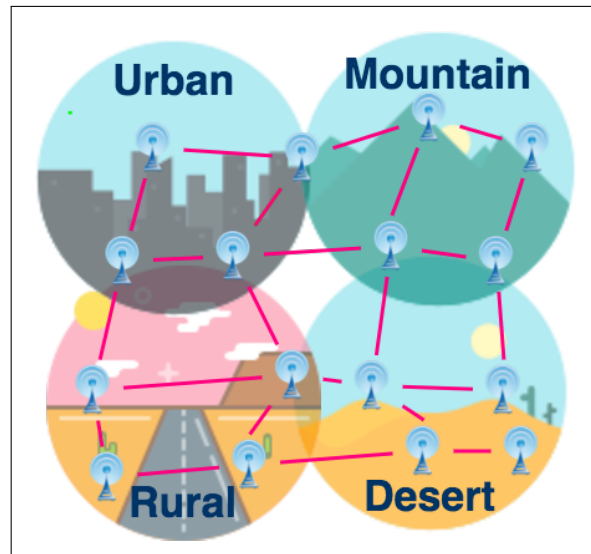


Fig. 6.4 Mesh Networking in Rural and Urban Environments

The second usage scenario for the design of a network topology includes an IoT mesh network able to monitor a large-scale area including the reception of monitoring measurements. In order to offer maximum coverage, a mesh network must be designed to enable path redundancy for failure avoidance. The topology should enable the maximum covered

area considering also the number and kind of IoT devices. As an extension of this approach it could be the definition of the minimum number of nodes to cover the maximum area for nodes with different wireless ranges and terrains as depicted in Figure 6.4.

Table 6.2 Experimental Results of the Mesh Pattern Execution

Distance (metres x metres)	Required Nodes	Exec. Time (msec)
100 x 100	4	139
200 x 200	9	154
300 x 300	16	203
400 x 400	25	268
500 x 500	36	533
800 x 800	81	854
1.000 x 1.000	121	1206
2.000 x 2.000	441	3945
5.000 x 5.000	2601	6438

To design an IoT mesh network, the *Mesh Pattern* is applied. The main scope of the mesh pattern is to define the placement of nodes for different distances. Different sets of covered areas ($\{d, d'\}$) are inserted as requirements in the pattern framework and the results can be seen in Table 6.2. After the execution of the mesh pattern in the pattern framework, a sample of the pattern for coverage of 800×800 can be previewed in the network simulator as can be seen in the Figure 6.5.

6.2.3 Scalable Network Design

The third usage scenarios of network topology includes the design of a multi-tier tree network topology as depicted in the Figure 6.6. This topology can represent an SDN-supported architecture where in Tier 0 is the SDN controller, Tier 1-3 are the intermediate switches and Tier 4 is the host layer. Tree topologies are able to support scalable network designs. The main concept that this scenario is able to investigate is the design of a flexible and extensive network topology for supporting a number of clients under an SDN-enabled network topology.

Table 6.3 Experimental Results of the Tree Pattern Execution

End-Hosts	Tie Number	Required Switches	Exec. Time (msec)
2	2	0	124
4	3	2	136
32	6	30	319
256	9	256	857

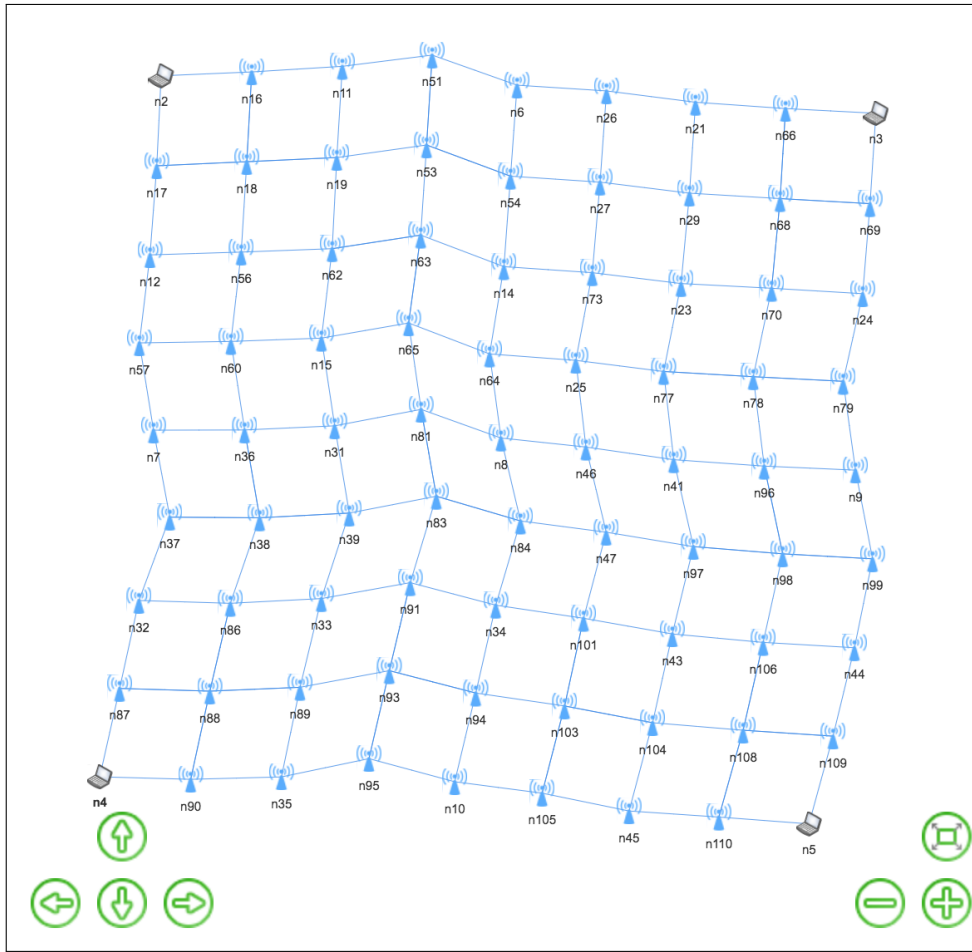


Fig. 6.5 Mesh Pattern Output on Network Simulator

To design scalable SDN-enabled topologies, the *Tree pattern* is applied. The main scope of this pattern is to identify the tiers, the number of necessary switches in order to satisfy a number of end-hosts. After the evaluation of the tree pattern for the different number of clients, the results can be found in Table 6.3. The depicted outputs in the network simulator for 32 clients can be seen in the Figure 6.7.

6.3 Design and Verification of S&D Networks

The second use case, investigated in this chapter, is the design and verification of network infrastructures with respect to S&D properties. Based on that, three different usage scenarios are described as follows:

- Reliable IoT Networks based on the developed *Reliability Patterns* as presented in Section 4.4.

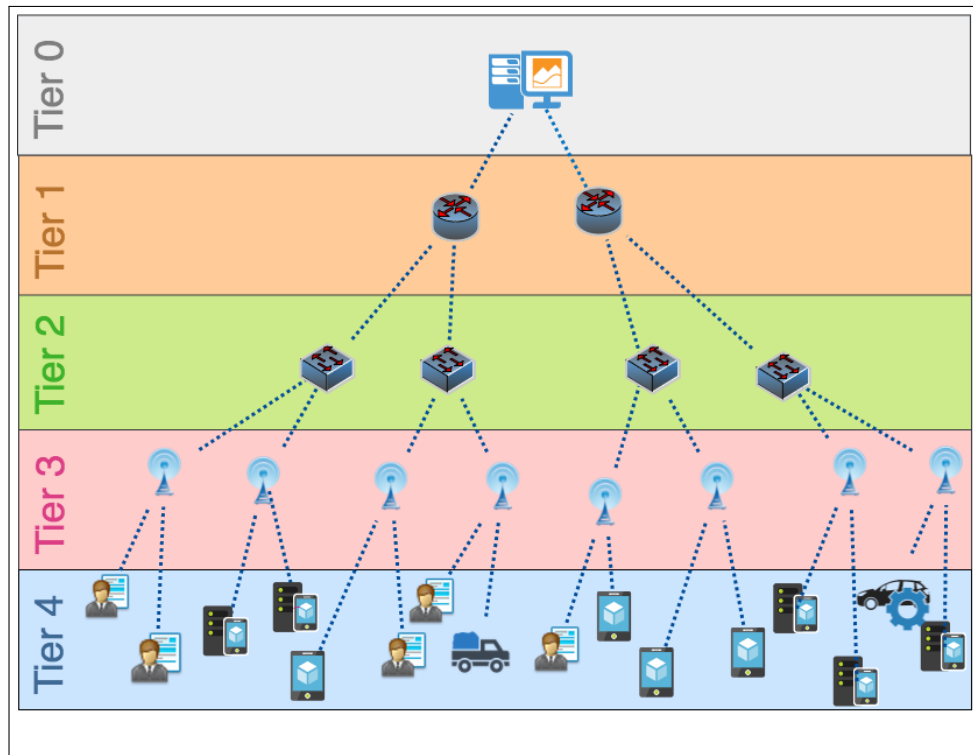


Fig. 6.6 Scalable Network Designs based on Tree Topologies

- Fault Tolerance SDN based on the developed *Fault Tolerance Patterns* as presented in Section 4.5.1.
- Confidential Transmission in SDN based on the developed *Confidentiality Patterns* as presented in Section 4.6.

6.3.1 Reliable Network Designs

System reliability is related to the reliability of cyber and physical components and to the connectivity between these components. Failures or attacks on the IoT network consisting of wireless IoT-enabled sensors may have as a result that possible anomalies disable the data transmission to the central controller.

In this usage scenario, a network is deployed to send monitored data to a central controller through relay nodes and paths. The transmission between end-hosts is applied through the relay nodes that forward the received data continuously. For instance, source can be the location of a monitoring mechanism and sink can be the location of a central controller as depicted in Figure 6.8. The main scope of this scenario is the design of a reliable IoT network.

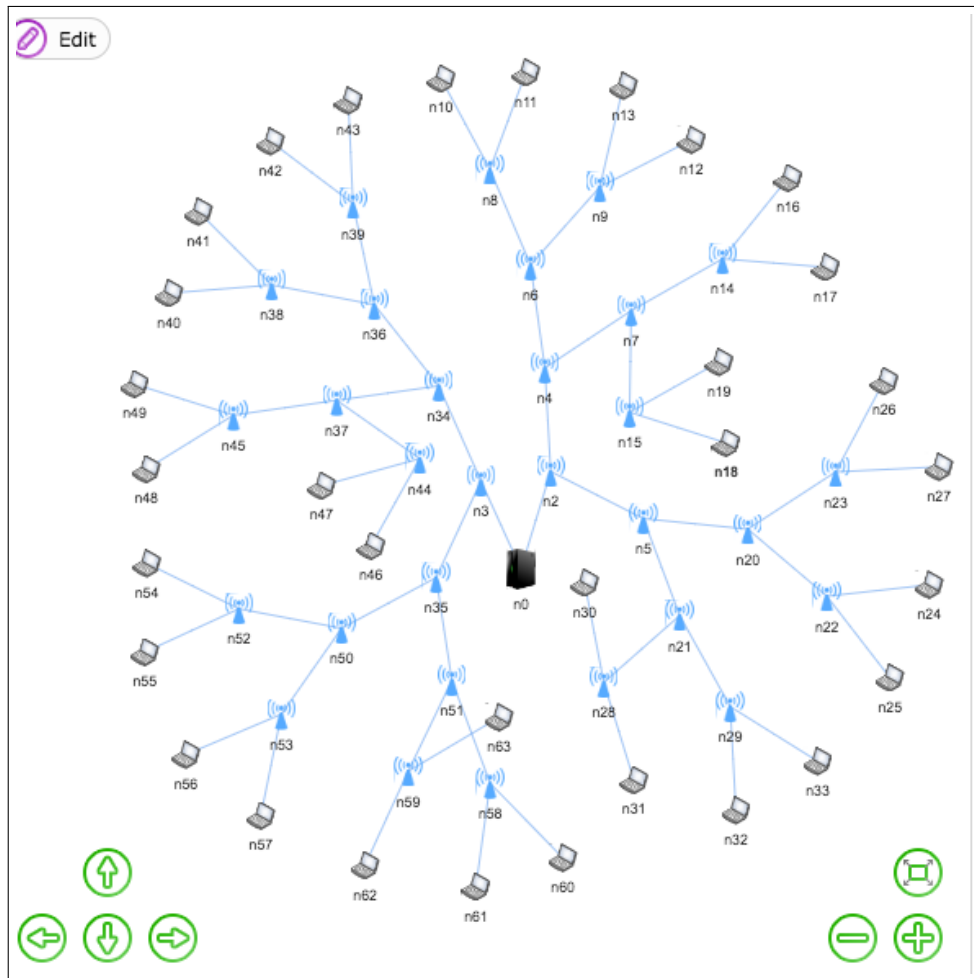


Fig. 6.7 Network Simulator Outputs on Tree Pattern

In order to satisfy the required reliability property, a number of different relay components which compose the system should be installed to enable a reliable monitoring mechanism.

The role of the pattern framework is to design reliable network infrastructures based on reliability patterns to avoid failure or attacks on the communication medium. This includes the definition of the number and the location of relay IoT devices that should be installed in order to ensure a reliable monitoring network. The pattern implies that the path reliability is related to the probability of an attack or failure. In order to validate the component composition of the monitoring mechanism, reliability patterns are used (a) to compose components (b) to validate system reliability and (c) to guarantee system reliability.

The inputs of the described scenarios include (a) the distance between source and destination, (b) the maximum range of communication link IoT sensors (c) the reliability of each component and (d) the required system reliability. The outputs of the described scenarios include the following (a) the number of IoT sensors, (b) their position and (c) the number of

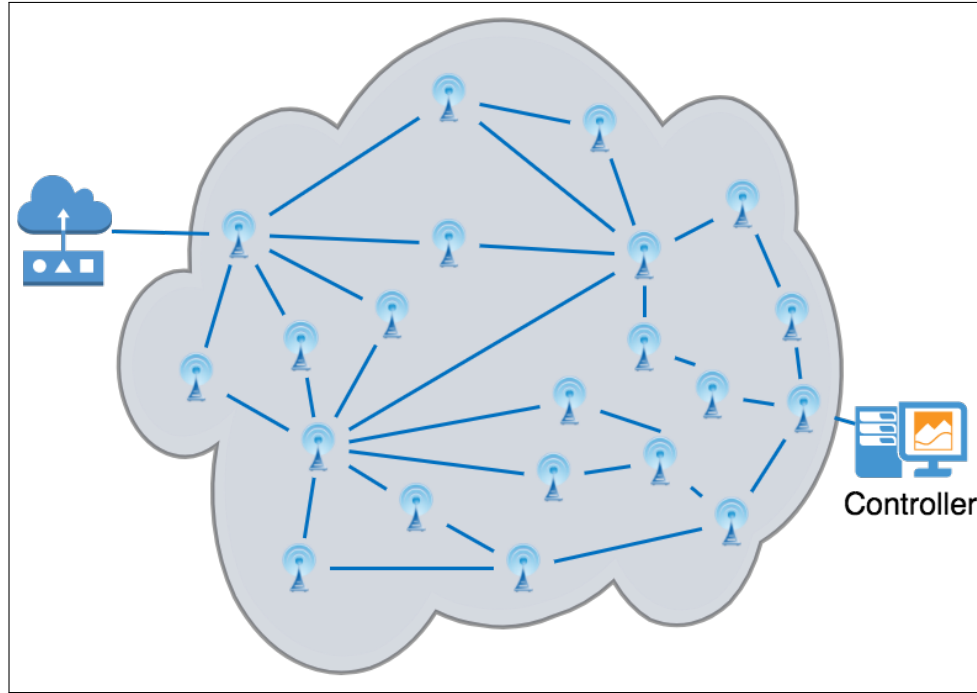


Fig. 6.8 Reliable Network Designs

paths and links. To evaluate the performance of the pattern, different scenarios are defined where reliability patterns can be used satisfactorily.

6.3.1.1 Serial and Parallel Reliability Pattern

The *Serial and Parallel Reliability Pattern* is able to validate whether the reliability of the system satisfies the required reliability and if not, it will add sensors in parallel in order to guarantee the required reliability. This will produce solutions concerning the number of sensors and their location. We may consider a system with c components as placeholders where in each placeholder an IoT sensor must be installed. The number of placeholders is related to the distance between the source and the sink is d . If the maximum range between two wireless sensors is r , the minimum number of relay nodes can be calculated by $d/r - 1$. The reliability of the system is equal to the composition of components in series: $r = \prod_{k=1}^n (r_k)$. The deterministic approach is followed to verify the connectivity between two sensors. If the distance between the two nodes is greater than the maximum distance d then the reliability of a link is 0. On the other hand, if the distance is less or equal than d , the link reliability is 1. This is an assumption to our approach because in reality the reliability of a link, particularly in case of wireless links, is probabilistic where other factors such as interference, path loss and propagation can influence the reliability of the connectivity.

Let assume that a system includes 2 placeholders c_1 and c_2 , the reliability of each sensor is 98% and the required reliability is 99%. The pattern will validate system reliability by placing two sensors s_1 and s_2 in series at placeholder c_1 . Calculating the reliability of the system it will give: $r = r_1 \cdot r_2 = 0.98 \cdot 0.98 = 0.96$ which is lower than the required reliability. Since the reliability is not guaranteed, the pattern will add an s_3 at second placeholder c_1 . A new validation occurs giving: $r = (1 - (1 - r_1) \cdot (1 - r_3)) \cdot r_2 = 0.98$. The reliability is close to 99% but even now the property is not satisfied. Therefore, the pattern will add a new sensor s_4 in placeholder c_2 . Finally, the $r = (1 - (1 - r_1) \cdot (1 - r_3)) \cdot (1 - (1 - r_2) \cdot (1 - r_4)) = 0.999$. The described procedure is depicted in Figure 6.9. This example shows the procedure which is followed for designing a reliable system containing 2 placeholders. However, in case of multi-hops networks the solution is not so easily provided. Reliability pattern is able to provide solutions for multi-hops networks as presented in the next subsection.

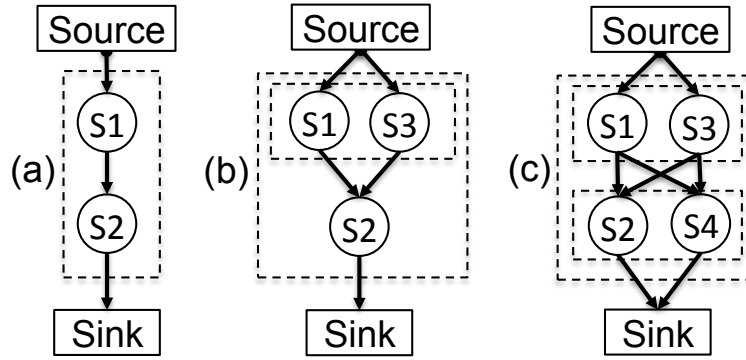


Fig. 6.9 Design Phases of a Sensor Network with Reliability (a) 96% (b) 98% (c) 99.9%

The previously described procedure is followed automatically by the *serial and parallel reliability pattern*, as expressed in Drools rules (Rule 4.6 and 4.7), and is used for constructing a reliable monitoring network consisting of wireless sensors. The number of placeholders is based on the distance between the source and the sink node and the wireless transmission range of each sensor. Let assume that each sensor has reliability factor 98% and transmission range 100m. As different factors of the experiments, we may consider the distance between the source and sink, which reflects the number of placeholders based on $d/r - 1$. The number of sensors, the execution times for different distances between source and sink of the described scenario are presented in Table 6.4. As we can observe from the results, the execution time is increased as the number of distance is increased. In addition, the number of sensors necessary for preserving the required reliability property is growing exponentially.

Table 6.4 Experimental Results of the Serial and Parallel Pattern Execution

Distance (metres)	No Placeholders	Reliability	No Sensors	Execution Time (msec)
1.000	9	99.6%	18	17
2.000	19	99.2%	38	19
3.000	29	99.8%	108	26
4.000	39	99.4%	128	35
5.000	49	99.0%	148	47
6.000	59	99.9%	336	50
7.000	69	99.9%	376	58
8.000	79	99.8%	508	66
9.000	89	99.4%	528	75
10.000	99	99.0%	548	78

6.3.1.2 Serial-Parallel Reliability Pattern

The evaluation of *Serial-Parallel Reliability Pattern* is also presented below. The results of the pattern evaluation, after applying the above pattern, is presented in Table 6.5. The table shows the number of nodes of each pattern and the time that was needed to execute each pattern. The requirement of 99.999% reliability suggests that a great number of nodes should be installed, especially for long distance links.

The developed pattern topology by the Serial-Parallel Reliability Pattern can be converted to an SDN architecture based on the developed mechanism as discussed in Chapter 5. Two different capabilities are examined below: i) the conversion of topologies to custom Mininet topologies and ii) the developed network simulator. The created SDN infrastructure can include hosts and OpenFlow-enabled (wired or wireless) switches as obtained by the reliability patterns. Then, the emulator is able to forward the topology to a remote controller such as ODL. Figure 6.10 depicts the outputs (nodes and links) of the reliability pattern when the distance between the source and the destination is 500m, the range is 100m and the uptime probability is 99%.

Table 6.5 Experimental Results of the Serial-Parallel Pattern Execution

Distance (metres)	No Nodes	Exec. Time (msec)
500	14	56
1.000	46	81
2.000	172	192
5.000	686	1487
10.000	2736	7530

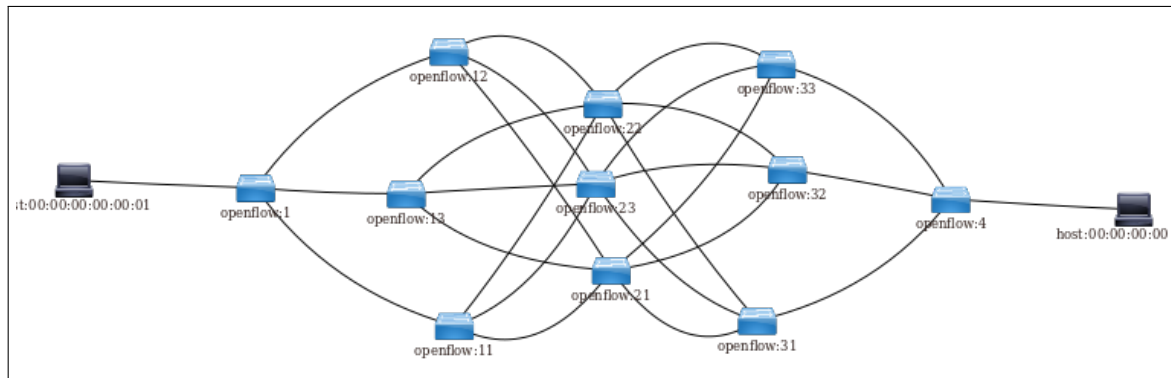


Fig. 6.10 OpenDaylight SDN Infrastructure Topology

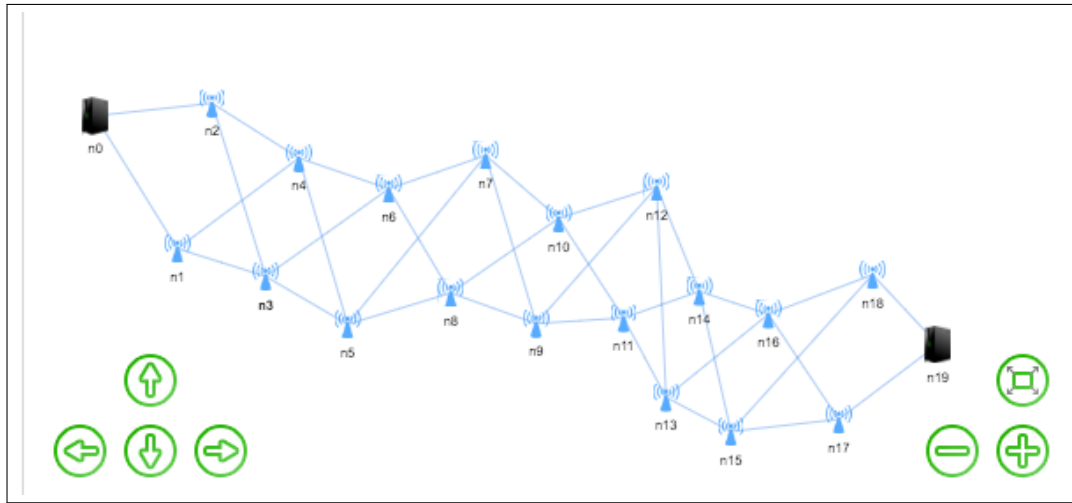
Finally, the comparison between the *Serial and Parallel Reliability Pattern* and the *Serial-Parallel Pattern* outputs, as presented in the network simulator, are depicted in Figure 6.11.

6.3.2 Fault Tolerance, Detection and Restoration in SDN

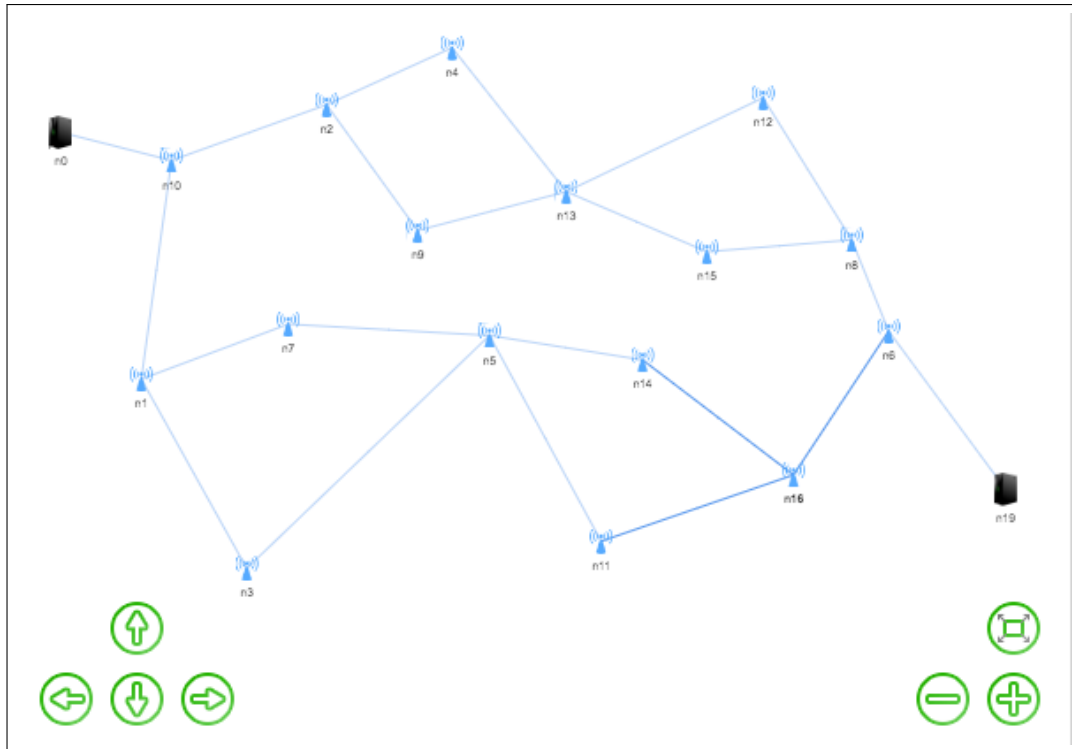
The use of SDN networks imposes the necessity to provide higher fault and intrusion tolerance compared to legacy networks as new threats are being introduced. More specifically, the ability to control networks by means of software and centralisation of network control makes SDN vulnerable to failures, attacks or overloads. In a network fault for example, new alternative network paths must exist or should be found as depicted in Figure 6.12. However, the most important factors for runtime adaptation appear to be both the detection of an attack or failure and the reaction time to transfer the new flow rules to the controller and the switches. The *Fault Tolerance Pattern* and the *Fault Detection and Restoration Pattern* will be presented and evaluated below. The role of the pattern framework is to insert the patterns in order to evaluate its performance and the applicability regarding fault tolerance in the SDN-enabled networks.

6.3.2.1 Fault Tolerance Pattern

The *Fault Tolerance Pattern* is applied to preplan and configure suitable paths at design time in order to provide a fault tolerance adaptation in case of fault or attack. To evaluate the performance of the *Fault Tolerance Pattern* a number of executions are made regarding the needed time to identify the available paths in order to preplan alternative solutions. Based on the pattern, a proactive mechanism is presented that is able to install the required flow rules inside the SDN switches in order to provide fault tolerance in case of attacks or faults. The



(a) Serial and Parallel Output



(b) Serial-Parallel Output

Fig. 6.11 Network Simulator Outputs on Reliability Patterns

results of the experiments for identification and installation of the required flows for different custom topologies including different number of hosts, switches and links are presented in Table 6.6.

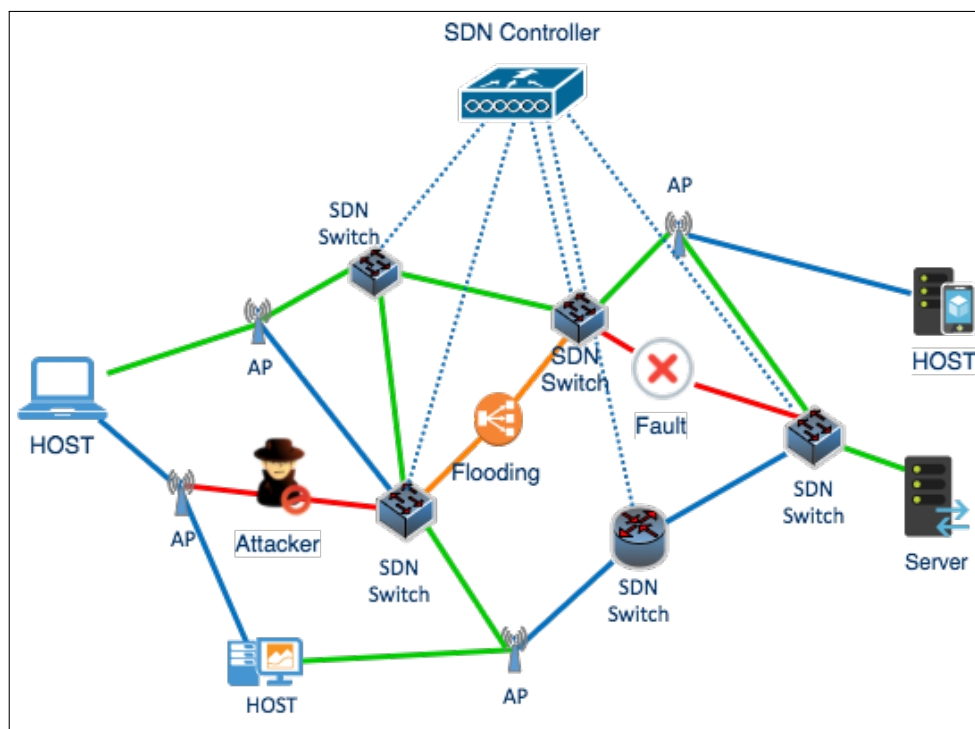
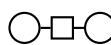
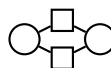
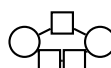
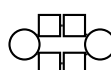
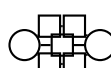
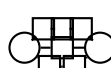


Fig. 6.12 Fault Tolerance in SDN

Table 6.6 Experimental Results of Fault Tolerance Pattern Flow Configurations

Topo	Hosts	Switches	Links	Config Time	Degree	Topology
1	2	1	2	0.1sec	1	
2	2	2	4	0.3sec	2	
3	2	3	5	0.4sec	2	
4	2	4	6	0.6sec	2	
5	2	5	8	1.1sec	3	
6	2	6	9	1.3sec	3	

6.3.2.2 Fault Detection and Restoration Pattern

Fault Detection and Restoration Pattern can be executed to support runtime SDN adaptation in cases of DoS attacks. For the detection, the pattern is based on the node connector statistics as fetched from the OpenFlow-enabled switches to the ODL controller. The statistics include

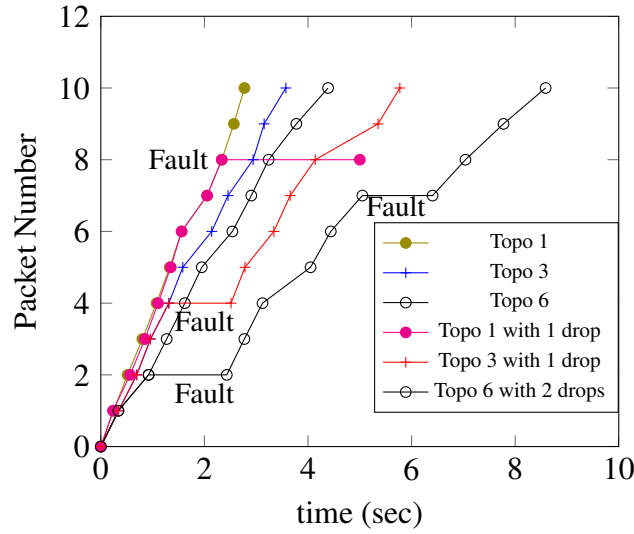


Fig. 6.13 Experimental Results of Fault Detection and Restoration Pattern

receive and transmit packets, errors, drops CRC errors and collisions. The pattern can retrieve these statistics and instantiate a new requirement for immediately reaction as an intrusion detection mechanism. Suitable also to be used in case of malicious adversaries that create DoS attacks and forward traffic to different secure paths based on the restoration capability.

The performance evaluation of the *Fault Detection and Restoration Pattern* is focused on the time needed to detect and restore path, if possible, in case of link failures. The purpose of these experiments, as conducted in the Mininet, is to send 10 packets from source to destination for the different network topologies as were defined in the previous experiments. Since the fault restoration pattern defines the shortest path between source and destination, the purpose is to measure the detection and restoration time in case of a failure. Therefore, six different experiments with and without faults are conducted to evaluate three different topologies from Table 6.6; the results are presented in Figure 6.13. *Topo 1* is the simplest one, containing two hosts and one switch. In this case, pattern cannot guarantee fault restoration since there is no alternative path to forward the traffic in case of a failure. So, when there is a fault, the transmission of the packets just stops. The results of the transmission of ten packets when there is no failure and in case of a fault can be seen in Figure 6.13. *Topo 3* contains two hosts and three switches, one switch in the first path and two switches in the second path creating two alternative paths. Initially, the fault tolerance pattern will choose the shortest path that contains only one switch and the packet transmission will be done through this path. When there is a fault in this path, the restoration pattern detects the failure and identifies an alternative path through longer path as can be seen in the transmission time in the Figure 6.13. In this case, the pattern is able to detect the failure and identify an alternative path that

requires less than a second for the detection and the restoration, including also the installation of the required flow rules. Finally, *Topo 6* includes two hosts again and 3 alternative paths with one, two and three switches per path respectively. The procedure is similar to *Topo 3*, however the existence of a third alternative, enables the fault tolerance (detection and restoration) of two failures, firstly in the path 1 (with the one switch) and secondly in path 2 (with the two switches). All the results of the experiments can be found in Figure 6.13.

6.3.3 Secure Transmission in SDN-enabled Networks

One of the most important issues especially in the SDN-enabled networks is to guarantee secure data transmission. This includes the protection of transmitted data either in data or in control plane but also through intermediate interface (ie. SBI). Different solutions for data protection have been proposed in the literature as discussed in the Chapter 2. These include the transmission of data through a number of different network devices such as routers, firewalls, NAT and IPSec-enabled gateways by applying access control policies based on predefined filtering rules able to forward discard or encrypt incoming traffic.

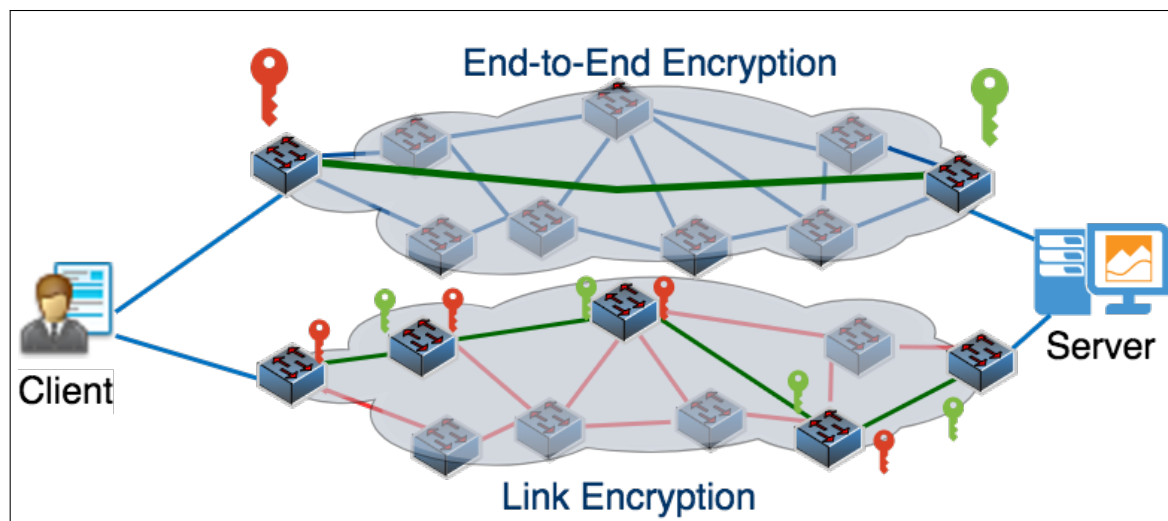


Fig. 6.14 End to End and Link Encryption Use Case

This use case deploys different approaches on how to enable secure data transmission between a client and a server. The main scope of this use case is to identify suitable paths able to support the confidential transmission of data between end hosts, as presented in Figure 6.14. Two different scenarios are described where the first one considers that each node should be able to encrypt/decrypt data by applying link encryption and the second one defines the E2E secure data transmission.

The role of pattern framework is to design secure SDN-enabled networks based on security patterns able to avoid attacks on the communication medium such as eavesdropping. These patterns guarantee that the exchanged data on the communication channel should be secure either at the source and destination or between each step of the transmission hops.

6.3.3.1 Link Encryption Pattern

The procedure regards the satisfaction of confidentiality between end hosts in network architectures based on link encryption. In order to guarantee confidentiality, the *Link Encryption Pattern* is applied. The pattern configures network paths, where the intermediate nodes share the same key and are able to guarantee link encryption. The evaluation of this pattern is done by the use of an existing SDN infrastructure such as the network topology shown in Figure 6.10). However, in this scenario, network nodes and links have different encryption level as presented in Table 6.7.

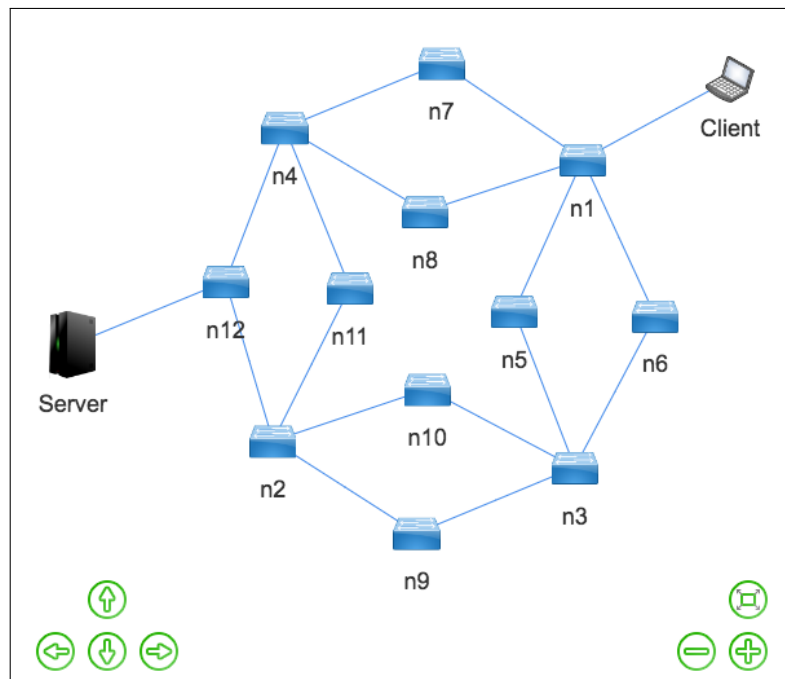
Table 6.7 Link Encryption on a Network Topology

Links	$l_{1,5}(n_1, n_5)$	$l_{1,4}(n_1, n_6)$	$l_{1,7}(n_1, n_7)$	$l_{1,8}(n_1, n_8)$	$l_{5,3}(n_5, n_3)$	$l_{6,3}(n_6, n_3)$	$l_{7,4}(n_7, n_4)$	$l_{8,4}(n_8, n_4)$	$l_{3,9}(n_3, n_9)$	$l_{3,10}(n_3, n_{10})$	$l_{4,11}(n_4, n_{11})$	$l_{4,12}(n_4, n_{12})$	$l_{9,2}(n_9, n_2)$	$l_{10,2}(n_{10}, n_2)$	$l_{11,2}(n_{11}, n_2)$	$l_{12,2}(n_{12}, n_2)$
Encrypted	✓	✓	X	✓	X	X	✓	✓	✓	✓	✓	✓	X	X	✓	✓

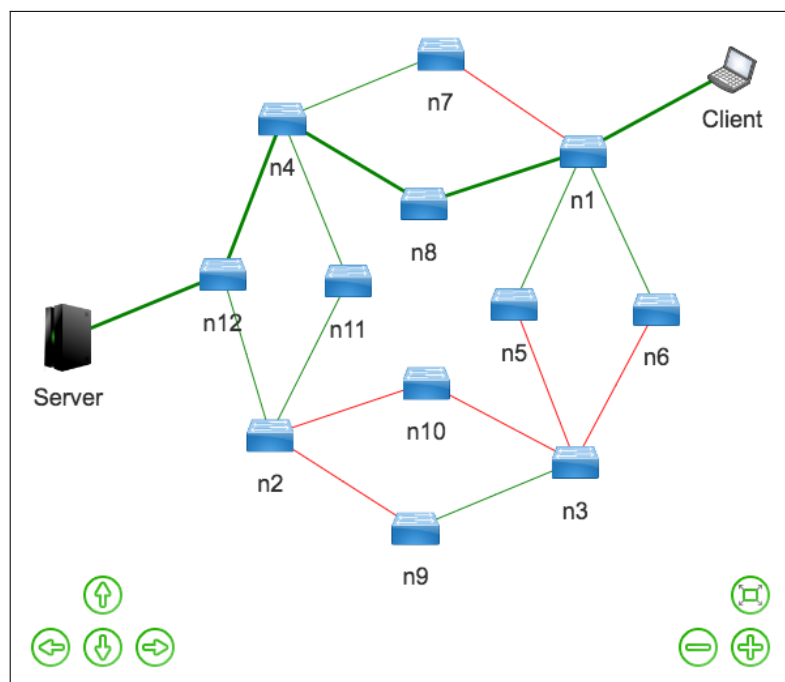
The evaluation of proposed schema is provided based on the three following phases:

- To preview the different conditions of the network topology regarding the security level of each link and relay node, the topology of the Figure 6.10 is inserted in the Figure 6.15a. The defined security levels (enabled/disabled) of each link are inserted in the knowledge base of the pattern engine as defined in 6.7.
- The *Link Encryption Pattern* can find the most suitable path(s) for providing link encryption from a client to a server. More specifically in the described scenario, the pattern is able to verify and identify the following sequential and parallel paths that guarantee the required link encryption property based on the pattern definition as follows: $\mathcal{R}\{s, t, \mathcal{P}\}$, where \mathcal{R} defines the requirement for link encryption property to guarantee \mathcal{P} from s source/client (n_1) to t destination/server (n_2). The results of this pattern execution provides the required encryption property is provided below:

$$\mathcal{R}\{\mathcal{P}(\text{Path}(s, t))\} =$$



(a) Network Topology



(b) Secure Path Selection

Fig. 6.15 Link to Link Encryption in Network Topologies

$$\mathcal{R}\{Path(n_1, n_2), encryption\} =$$

$$\begin{aligned}
& \mathcal{R}\{Link(n_1, n_4), encryption\} \text{ and } \mathcal{R}\{Link(n_4, n_{11}), encryption\} \\
& \text{and } \mathcal{R}\{Link(n_{11}, n_2), encryption\} \\
& \text{or} \\
& \mathcal{R}\{Link(n_1, n_4), encryption\} \text{ and } \mathcal{R}\{Link(n_4, n_{12}), encryption\} \\
& \text{and } \mathcal{R}\{Link(n_{12}, n_2), encryption\}
\end{aligned}$$

- Finally, the last procedure involves the instantiation of the respective OpenFlow rules to be installed in the intermediate switches, reflecting also the selected paths as can be previewed in the Figure 6.15b.

6.3.3.2 End to End Security Pattern

One of the crucial parts of a designer is to instantiate paths and configure the respective nodes for providing E2E security property guarantees. This scenario focuses on the instantiation and the configuration of respective security properties to support E2E secure path transmission in the SDN data plane. The main mechanism to provide such E2E paths are based on the IPsec enabling security mechanisms as proposed by the *E2E Security Patterns*. Two different steps are included in this scenarios as described below:

- The first step includes the instantiation of the required security associations (SA) and security policies (SP) for the specific end-nodes to provide the requested security requirements configuration.
- The second step is the identification of the IPsec-enabled forwarding devices that are able to encrypt/decrypt or authenticate the incoming and outgoing data. The role of this step in the scenario is to identify the most efficient route for connecting source and destination to these IPsec-enabled switches. After the identification of the respective switches, the instantiated configuration from *step 1* will be inserted in the respective switches.

To evaluate the performance of the E2E Security Pattern, a custom topology is developed including end-hosts, IPsec-enabled and non IPsec-enabled (with the red stripes) switches. The custom topology is depicted in Figure 6.16 where apart from the hosts, the switches with green stripes are the IPsec-enabled ones and with red stripes are the non IPsec-enabled ones. Based on this topology, different requirements are identified for enabling E2E secure transmission between the end-hosts and the intermediate nodes. As discussed previously, the first step includes the IPsec configurations (SA and SP) that should be inserted in the OVS

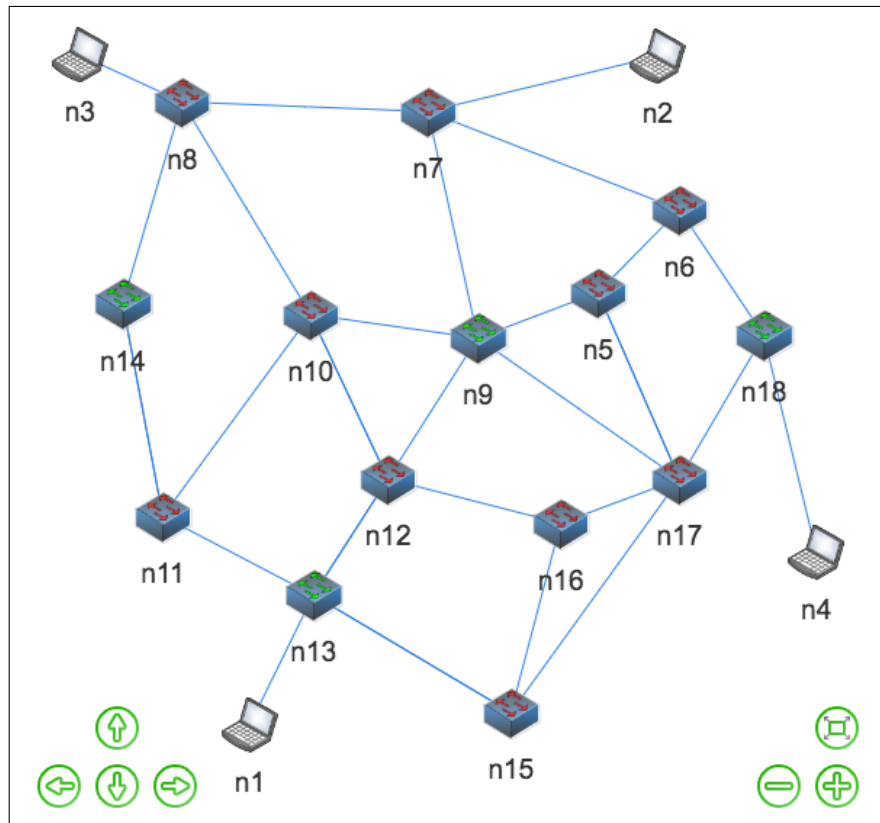


Fig. 6.16 End to End Encryption in Network Topologies

switches where the IPSec protocol is enabled. The next step includes the identification of two IPSec-enabled switches, one close to the source node and another close to the destination. The insertion of the required configurations is applied to ensure confidentiality and integrity through the IPSec encryption/decryption and authentication mechanisms.

Table 6.8 End to End Security on a Network Topology

Source	Destination	Hops	Secure Links
n1	n2	5	2
n1	n3	5	2
n2	n3	3	0
n2	n3	7	3
n2	n4	5	2
n1	n4	5	3

For different E2E secure paths requests, the pattern is able to identify the provided available paths to forward the traffic. This can be done by defining the available paths related to the IPSec-enabled switches close to the source and the destination. In Table 6.8 the

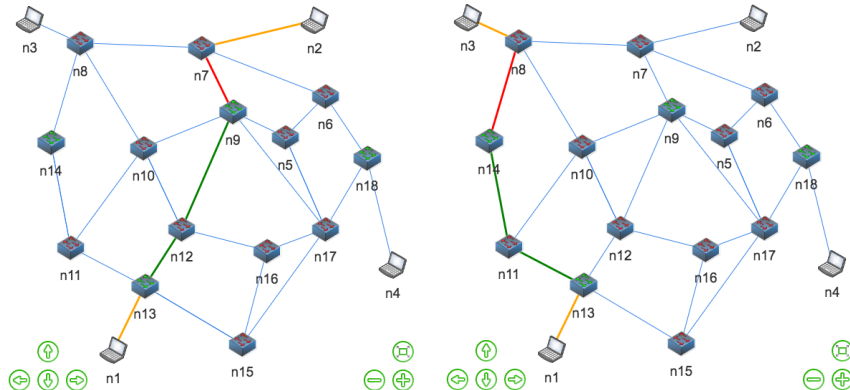
different E2E requests and the outputs of the path identification are provided. In the Figure 6.17, the fully or partially secure E2E paths are depicted in the network simulator by the identification of different path colors. The green path defines a secure transmission, where an insecure one is presented. Finally, orange path can express an undefined one, corresponding to the link between the host and the gateway where the IPSec tunnel mode is required for guaranteeing security in this link.

6.4 Design Secure Industrial Networks Leveraging Service Function Chaining

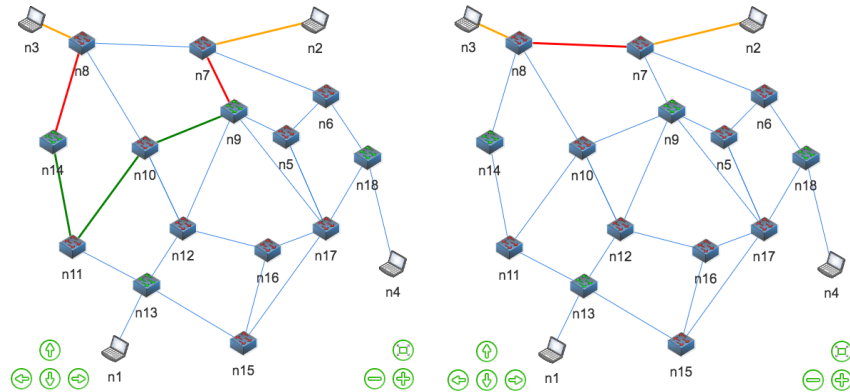
One of the main objectives in industrial networks is faster service provisioning. The time to provision the service is foreseen to be reduced from several days to several minutes. The concept of SFC has already shown promising results in enabling the faster time-to-market for the new services in the domain of telecom operators. This implies the potential to reduce CAPEX and OPEX, especially for short lived service. Depending on the focused aspect relevant for each deployment of the proposed reactive security leveraging a service function chaining, a mainly security-focused use case is identified.

This use case includes an approach to achieve reactive security for SDN/NFV-enabled industrial networks, based on the use of SFC to dynamically chain various security functions, classify traffic and steer traffic accordingly. The proof-of-concept application of this approach led to the development of the reactive security modeled on (and deployable to) an actual, operating wind park. This allows the continuous monitoring of the industrial network and detailed analysis of potential attacks. Thus isolating attackers can enable the assessment of their level of sophistication (e.g. from script kiddies to state actors). The main scope of this use case is to provide a security SFC-based enhancement, for both intra- and inter-domain deployments, with the ability to forward traffic, based on its security classification (e.g. legitimate/SCADA-traffic/malicious/unknown), following the classification of Service Function Paths for each traffic type. Moreover, this type of classification opens up various possibilities for the integration of advanced malicious traffic detection techniques (e.g. exploiting machine learning).

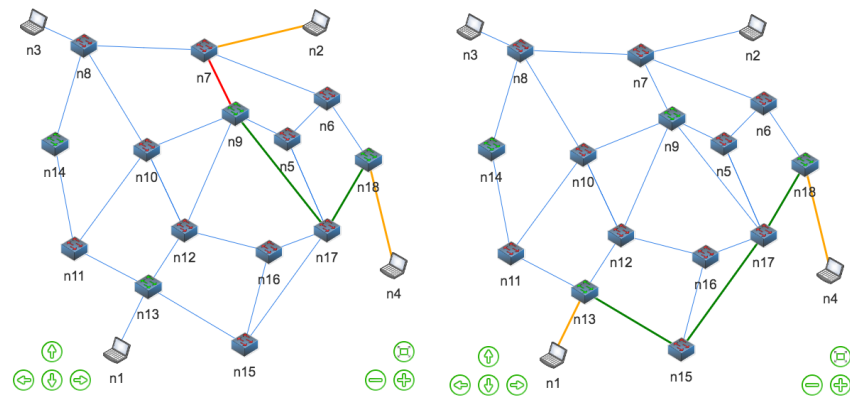
The role of the pattern framework in this use case is: i) to instantiate required security functions for the different SFC requests based on the *VNF Instantiation Pattern* and ii) to instantiate service function paths to forward the different type of traffics based on the *SFC Path Finding Pattern*. In the next subsections, the SFC reactive security is evaluated in a wind park use case to express traffic classification, as depicted in Figure 6.18.



(a) Partially Secure Traffic from n1 to n2 (b) Partially Secure Traffic from n1 to n3



(c) Multi-hop Partially Secure Traffic from n2 to n3 (d) Insecure Traffic from n2 to n3



(e) Partially Secure Traffic from n2 to n4 (f) Fully Secure Traffic from n1 to n4

Fig. 6.17 Evaluation of the End-to-End Security Pattern in the Network Simulator (green: secure, red: insecure, orange: undefined)

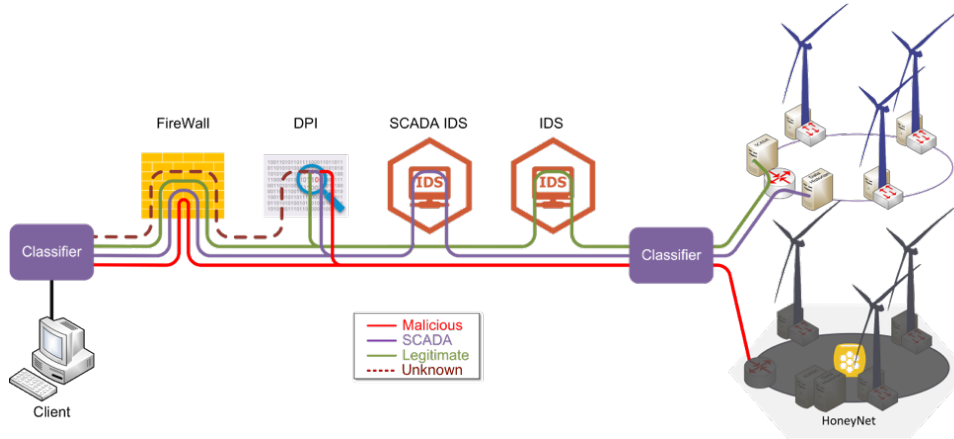


Fig. 6.18 Reactive Security - Implemented Service Chains

6.4.1 Virtual Functions Instantiation based on SFC Requests

The first aim of this use case is to instantiate the required chains and the respective VNFs for providing packet classification and secure traffic forwarding under a proposed industrial scenario. Based on the classification of each packet, the traffic can be directed to one of three different SFCs (legitimate, unknown suspicious or malicious) as depicted in Figure 6.19 (a). The aim for this process is to route unknown/suspicious traffic via the IDS and DPI Service Functions, in order to classify it (as either legitimate or malicious), thus allowing us to forward it to the windpark or the honeypot, accordingly. Thus, malicious traffic can be isolated in the honeypot, allowing us to track the attacker, identify its purpose and keep him occupied. For the inter-domain use case, (Figure 6.19 (b)), the procedure is similar to the intra-domain scenario. However, a more sophisticated honeypot deployment, such as a HoneyNet, can be used as an emulated wind park, having similar services and functions as the original wind park. Moreover, in this case, after having acquired the needed tag (as malicious or legitimate) in other parts of the larger wind park deployment, the traffic can avoid going through the same procedures (i.e. Service Functions) again. This highlights the benefits of SFC in terms of potential performance gains.

The per-traffic type classification is adopted for the reactive security, integrating all the developed security service functions detailed in the Chapter 5, via the following implemented SFCs. This type of classification forms the basis of the reactive security presented herein. Based on the S&D property guarantees by each service function, the following classification of type of traffic was made:

- The **legitimate traffic** (non-SCADA) is routed through the *Firewall* and then through the generic *IDS*, before being forwarded to the intended destination (in this case, the Data Historian).

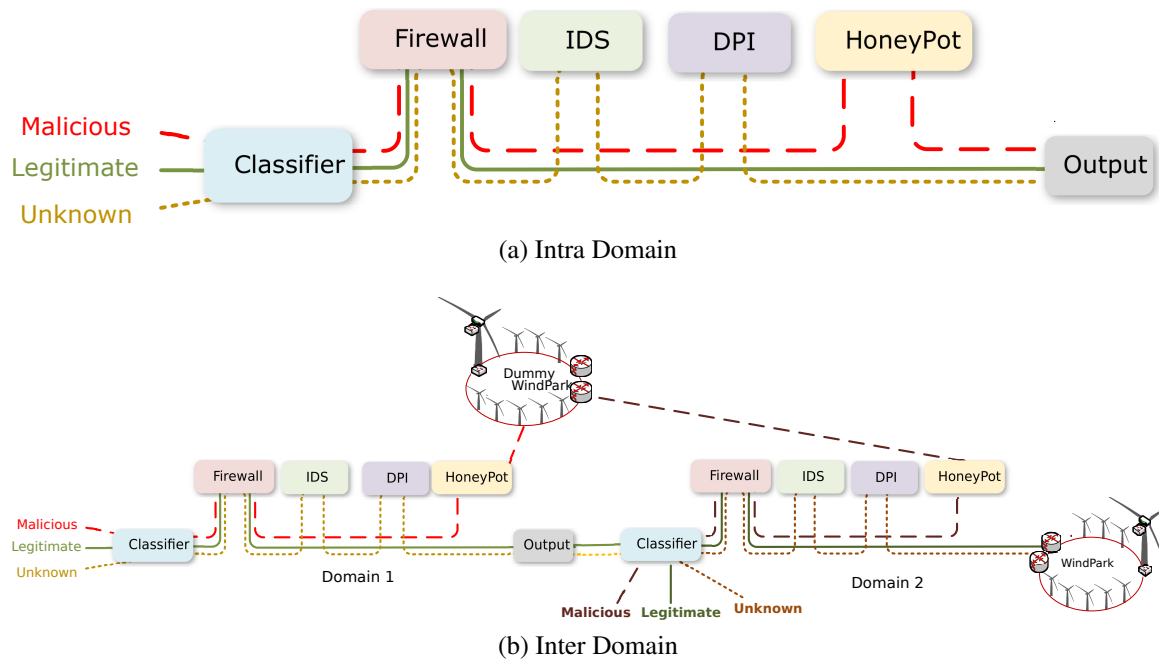


Fig. 6.19 SFC - Per Traffic Type Classification Example

Chain 1 - Legitimate (known) traffic: Firewall -> IDS -> Output (Data Historian)

- The **SCADA traffic** is routed through the *Firewall* and then through the *SCADA-IDS*, which features only SCADA rules, to minimise the performance impact, before being forwarded to its intended destination.

Chain 2 - SCADA traffic: Firewall -> IDS -> SCADA-IDS -> Output (SCADA)

- Traffic tagged as **malicious** (e.g. nmap port scanning), either by the Classifier or by the DPI functionality, is routed through the Firewall and then to the appropriate part of the honeynet; the latter can either be the SCADA Honeypot (if its original target was a SCADA system), the generic Honeypot (if its original target is an SDN device or a production system such as the Data Historian) or the passive EWIS honeypot (if the original target is some unused address, indicating malicious probing/footprinting of the network).

Chain 3 - Malicious traffic: Firewall -> Honeypot/Honeynet

- Finally, Traffic that is of **unknown type** (i.e. cannot be classified based on simple ACL rules that the Classifier has), is routed to the *Firewall* and then to the *DPI* (nDPI), where it is analysed, classified and its headers are updated appropriately and then is being assigned to the appropriate Chain (Chains 1 to 3).

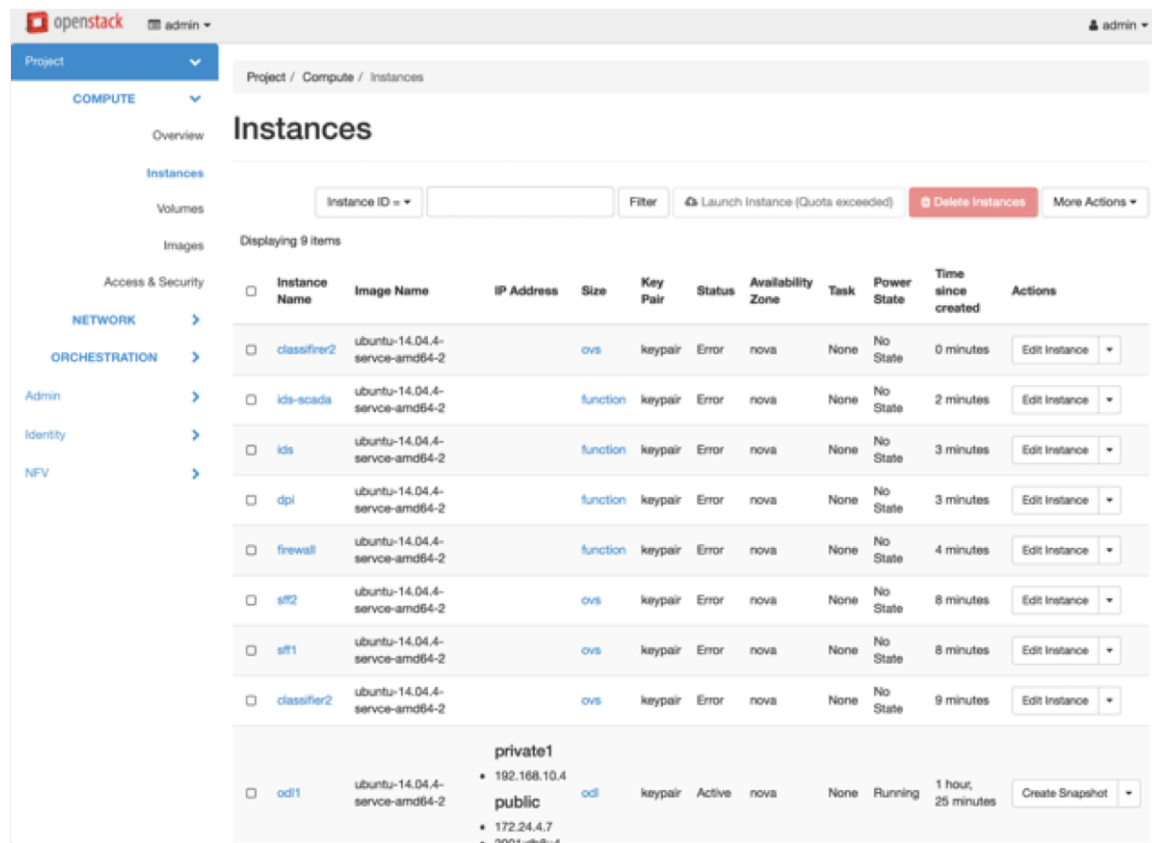
Chain 4 - Unknown traffic: Firewall -> DPI -> Chain 1 or Chain 2 or Chain 3

The procedure is followed by the framework when a data packet enters the intra-domain wind park deployment and a classification is made to identify in which chain the traffic should be forwarded. The classifier is responsible for classifying and forwarding packets based on predefined ACL rules, exploiting pattern matching and tags found on the packet headers. The (attached to the SFF) classifier forwards the packets through one of the predefined function chains. For every chain, an access control list is required to be inserted in the classifier as described below:

- **Service Functions:** $F = \{f_{fw}, f_{ids}, f_{scada-ids}, f_{dpi}, f_{hp}\}$
- **Service Function Chains:** $s_1 = (f_{fw}, f_{ids}), s_2 = (f_{fw}, f_{scada-ids}), s_3 = (f_{fw}, f_{hp}), s_4 = (f_{dpi}, \{s_1, s_2, s_3\})$
- **SFC:** $S = \{s_1, s_2, s_3, s_4\}$
- **Access Control Lists:** F is the set of ACL rules f_i the service functions chains.
- **Classifiers:** $\{(s_1, f_1), (s_2, f_2), (s_3, f_3), (s_4, F/\{f_1, f_2, f_3\})\}$

After the definition of the SFCs and the corresponding rules per chain, the next step includes the instantiation of the SFCs corresponding to the available VNFs or the instantiated ones. Therefore, the VNF Instantiation Pattern is responsible to request the specific identified VNFs from the MANO which has all the required information regarding the existence of the specific VNFs on the different service nodes. If there is not any available VNF, then the pattern should request to MANO to instantiate one from the closest to the source service node that has available resources. After the SFC pattern recursion, the instantiation of the SFC request will be completed.

The SFC instantiation includes two phases: the VNF instantiation and the insertion of the configuration SFC files in SDN controller configuration. More specifically, the instantiation of the VNFs can follow either the described MANO-approach or without MANO. For the MANO-based approach, Openstack is used as depicted in Figure 6.20. Furthermore, the second used approach includes the VNF instantiation without the use of MANO in the Proxmox hypervisor as depicted in the Figure 6.21. The second phase includes configuration of SFC requests in the ODL controller. When all the VNFs are instantiated based on SFC requests and included in the SFCs, the corresponding output files in Json formats Appendix B are inserted in the ODL controller. These templates include all the required information for SFC requests. The screenshots of the corresponding ODL SFC includes Service Functions 6.22, Service Node 6.23 and the Service Function Chains 6.24.



Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
classifier2	ubuntu-14.04.4-service-amd64-2		ovs	keypair	Error	nova	None	No State	0 minutes	Edit Instance
ids-scada	ubuntu-14.04.4-service-amd64-2		function	keypair	Error	nova	None	No State	2 minutes	Edit Instance
ids	ubuntu-14.04.4-service-amd64-2		function	keypair	Error	nova	None	No State	3 minutes	Edit Instance
dpi	ubuntu-14.04.4-service-amd64-2		function	keypair	Error	nova	None	No State	3 minutes	Edit Instance
firewall	ubuntu-14.04.4-service-amd64-2		function	keypair	Error	nova	None	No State	4 minutes	Edit Instance
sff2	ubuntu-14.04.4-service-amd64-2		ovs	keypair	Error	nova	None	No State	8 minutes	Edit Instance
sff1	ubuntu-14.04.4-service-amd64-2		ovs	keypair	Error	nova	None	No State	8 minutes	Edit Instance
classifier2	ubuntu-14.04.4-service-amd64-2		ovs	keypair	Error	nova	None	No State	9 minutes	Edit Instance
odi1	ubuntu-14.04.4-service-amd64-2	private1 • 192.168.10.4 public • 172.24.4.7 • 2001:db8::4	odi	keypair	Active	nova	None	Running	1 hour, 25 minutes	Create Snapshot

Fig. 6.20 VNF Instantiation in OpenStack

6.4.2 SFC Path Finding and Traffic Classification

The next step of the reactive security includes the path identification to forward the incoming traffic to the respective chains. The procedure involves the association of the instantiated VNFs to the classifiers and forwarders linked to the existing OVS switches. The inserted configurations of the Forwarders and the ACLs are imported in the ODL controller as separate Json files as depicted in Figures 6.25 and 6.26. In addition, the final step includes the SFC path finding in order to insert the respective rules in the classifiers and forwarders. The results of the OpenFlow rules inside the ovs switches are presented in Figure 6.27.

To evaluate the instantiated paths and the data flows through the different SFCs, suitable shell scripts are developed in order to evaluate the deployed SFC chains for different traffic types (Figure 6.28). In addition, changes in path for different traffic types are depicted on the GUI, with different colors for the various active chains at each instance in time; the various options that can be active (depicting real-time traffic flows and their associated chains) appear in Figure 6.30.

Proxmox Virtual Environment
Version: 4.1-1/2f9650d4

You are logged in as 'npetro@pam' Logout Create VM Create CT

Server View Node 'spica'

Search Summary Services Network DNS Time Syslog Task History Firewall Updates Console Ceph Subscription

Search:

Type	Description	Disk usage	Memory usage	CPU usage	Uptime
qemu	100 (win10)	0.0%	26.4%	1.0% of 2CPUs	33 days 07:31:31
qemu	101 (Linux-f001)	0.0%			-
qemu	102 (IDS)	0.0%	87.4%	0.2% of 2CPUs	21 days 21:09:00
qemu	103 (IDStest)	0.0%			-
qemu	602 (shamanwin10)	0.0%	35.0%	0.6% of 8CPUs	29 days 02:06:04
qemu	900 (Mininet)	0.0%	71.0%	0.2% of 4CPUs	45 days 03:43:33
qemu	901 (SFC-odf)	0.0%	88.7%	12.7% of 4CPUs	45 days 05:49:40
qemu	909 (Firewall)	0.0%	42.4%	41.6% of 4CPUs	14 days 15:12:08
qemu	910 (DPI)	0.0%	21.7%	42.1% of 4CPUs	14 days 15:24:38
qemu	911 (IDS)	0.0%	73.8%	0.9% of 4CPUs	28 days 21:47:53
qemu	912 (IDS-SCADA)	0.0%	51.5%	0.9% of 4CPUs	28 days 21:47:50
qemu	914 (HoneyNet)	0.0%	68.8%	0.0% of 4CPUs	34 days 01:46:50
qemu	915 (Classifier1)	0.0%	66.3%	0.2% of 4CPUs	35 days 21:29:30
qemu	920 (Classifier2)	0.0%	67.5%	0.3% of 4CPUs	35 days 07:31:36
qemu	923 (Sff1)	0.0%	66.5%	0.2% of 4CPUs	35 days 03:34:57
qemu	924 (Sff2)	0.0%	66.6%	0.2% of 4CPUs	35 days 04:03:45
qemu	925 (Sff3)	0.0%	62.0%	0.2% of 4CPUs	35 days 04:03:42
qemu	930 (Client)	0.0%	56.1%	0.1% of 4CPUs	34 days 03:29:35
qemu	932 (WindPark-Server)	0.0%	65.1%	0.1% of 4CPUs	34 days 03:29:32
qemu	933 (WindPark-SCADA)	0.0%	67.7%	0.2% of 2CPUs	21 days 06:57:48
qemu	938 (EWIS-HoneyPot)	0.0%	66.7%	25.3% of 4CPUs	30 days 06:37:51
qemu	940 (SCADA-HoneyPot)	0.0%	68.3%	0.2% of 2CPUs	29 days 04:15:26
qemu	950 (mininet-ovs)	0.0%	85.5%	0.1% of 4CPUs	5 days 04:30:22
qemu	1000 (Openstack)	0.0%	97.0%	6.3% of 16CPUs	1 day 06:09:32
storage	local (spica)	23.0%			-
storage	sfc (spica)	38.1%			-

Fig. 6.21 VNF Instantiation in Proxmox

Service Nodes Service Function Forwarders Service Functions Service Function Chains

Service Function Paths Access Lists/Classifiers NSH Metadata IPFIX APPID System Info Config

Add Service Function Clear sorting

Service Functions Inventory Service Function Types

Service Function name	Service Function type	IP Management Address	REST URI	NSH aware	Data plane locator	Actions
Search by name		Search by ip-mgmt-	Search by rest-uri		Search t	
dpi-1	dpi	192.168.10.20	http://192.168.10.20:5000	true	dpi-1-dpl	
firewall-1	firewall	192.168.10.10	http://192.168.10.10:5000	true	firewal-1-dpl	
ids-1	ids	192.168.10.30	http://192.168.10.30:5000	true	ids-1-dpl	
scada-ids-1	ids	192.168.10.40	http://192.168.10.40:5000	true	scada-ids-1-dpl	

Fig. 6.22 Service Functions Imported in the ODL

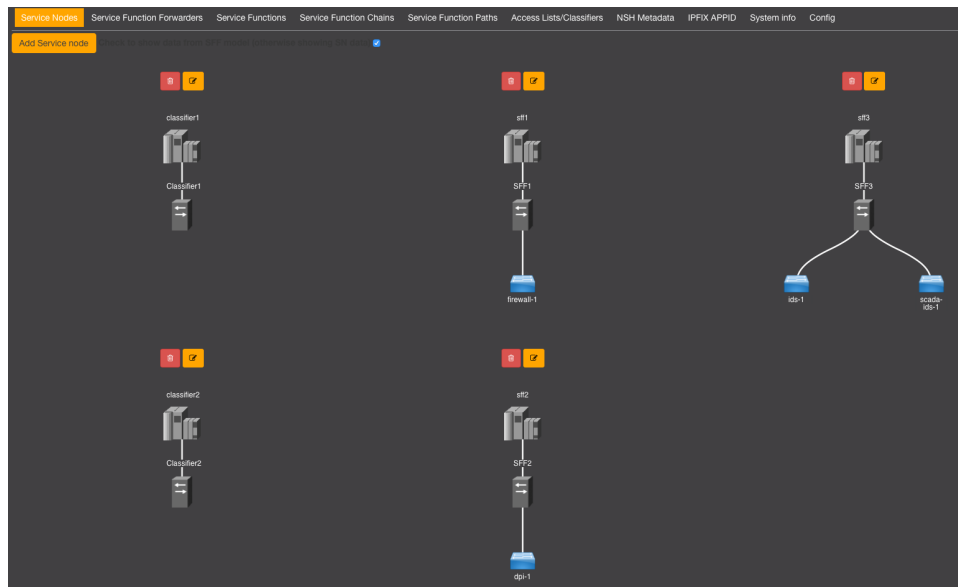


Fig. 6.23 Service Node Imported in the ODL

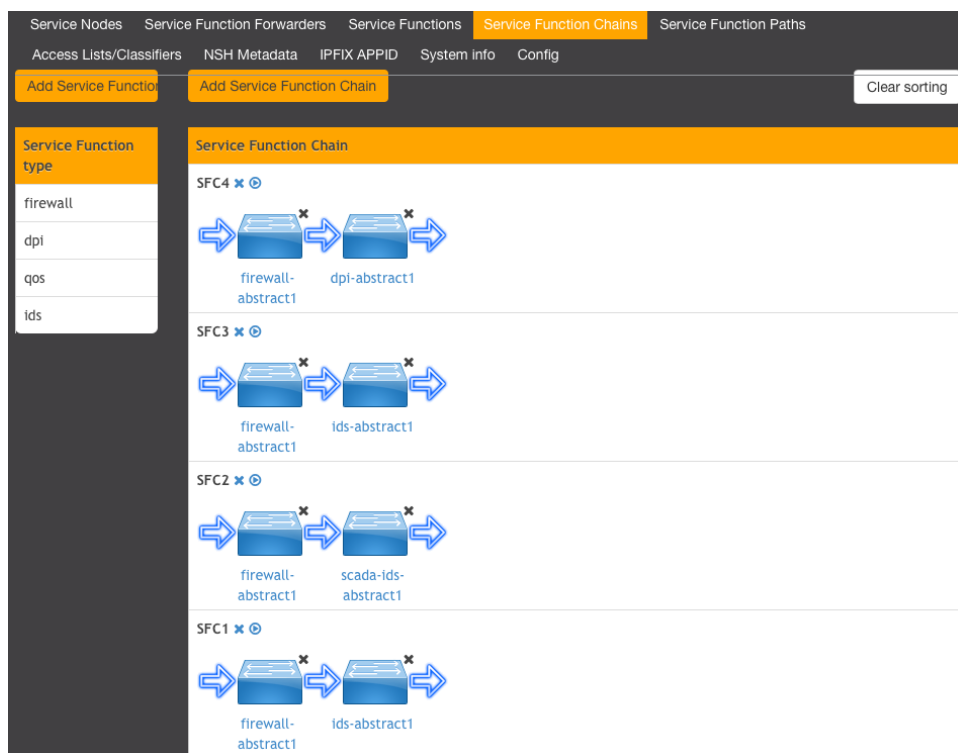


Fig. 6.24 Service Function Chains Imported in the ODL

In Table 6.9, the results of the conducted experiments are presented. Apart from the delay between end-hosts, following the respective service function chain, the delays between end-hosts i) when there is no function in the middle and ii) when all the security functions

Service Nodes

Service Function Forwarders

Service Functions

Service Function Chains

Service Function Paths

Access Lists/Classifiers

NSH Metadata

IPFIX APPID

System Info

Config

Add Service Function Forwarder

Create new Open vSwitch bridge

Clear sorting

SFF Inventory

Service Function Forwarder name	IP management address	REST URI	Service Node	Data plane locator	Service Function dictionary	Actions
<div>Search by name</div>	<div>Search by ip-mgmt-a</div>	<div>Search by rest-uri</div>	<div>Search by servi</div>	<div>Search by sff-d</div>	<div>Search by service-func</div>	
Classifier1	192.168.10.50	http://192.168.10.50:5000	classifier1	sff0-dpl		<div><div>✕</div><div></div></div>
Classifier2	192.168.10.60	http://192.168.10.60:5000	classifier2	sff4-dpl		<div><div>✕</div><div></div></div>
SFF1	192.168.10.70	http://192.168.10.70:5000	sff1	sff1-dpl	firewall-1	<div><div>✕</div><div></div></div>
SFF2	192.168.10.80	http://192.168.10.80:5000	sff2	sff2-dpl	dpi-1	<div><div>✕</div><div></div></div>
SFF3	192.168.10.90	http://192.168.10.90:5000	sff3	sff3-dpl	ids-1, scada-ids-1	<div><div>✕</div><div></div></div>

Fig. 6.25 Service Function Forwarders Imported in the ODL

Service Nodes

Service Function Forwarders

Service Functions

Service Function Chains

Service Function Paths

Access Lists/Classifiers

NSH Metadata

IPFIX APPID

System Info

Config

Add Access List

Clear sorting

ACL	ACE	Target RSP	Src IP/MAC(mask)	Dst IP/MAC(mask)	Flow	Src port	Dst port	IP protocol	DSCP	L7	Actions
<div>Search</div>	<div>Search</div>	<div>Search</div>	<div>Search by source</div>	<div>Search by dest</div>	<div>Search</div>	<div>Search</div>	<div>Search</div>	<div>Search</div>	<div>Search</div>	<div>Search</div>	
ACL1	ACE1	RSP1	192.168.2.0/24	192.168.2.0/24		0-	80-	6			<div><div></div><div></div></div>
ACL2	ACE2	RSP1-Reverse	192.168.2.0/24	192.168.2.0/24		80-	0-	6			<div><div></div><div></div></div>
ACL3	ACE3	RSP2	192.168.2.0/24	192.168.2.0/24		0-	81-	6			<div><div></div><div></div></div>
ACL4	ACE4	RSP2-Reverse	192.168.2.0/24	192.168.2.0/24		81-	0-	6			<div><div></div><div></div></div>
ACL5	ACE5	RSP3	192.168.2.0/24	192.168.2.0/24		0-	82-	6			<div><div></div><div></div></div>
ACL6	ACE6	RSP3-Reverse	192.168.2.0/24	192.168.2.0/24		82-	0-	6			<div><div></div><div></div></div>
ACL7	ACE7	RSP4	192.168.2.0/24	192.168.2.0/24		0-	0-	1			<div><div></div><div></div></div>
ACL11	ACE11	RSP4	192.168.2.0/24	192.168.2.0/24		0-	0-	17			<div><div></div><div></div></div>
ACL9	ACE7	RSP4	192.168.2.0/24	192.168.2.0/24		0-	100-550	6			<div><div></div><div></div></div>
ACL12	ACE12	RSP4-Reverse	192.168.2.0/24	192.168.2.0/24		0-	0-	17			<div><div></div><div></div></div>

Fig. 6.26 Access Control Lists Imported in the ODL

are used, are also presented. Although, the results of the experiments are related to the location and the distance of the VMs (in this case, they are all located on the same server), the

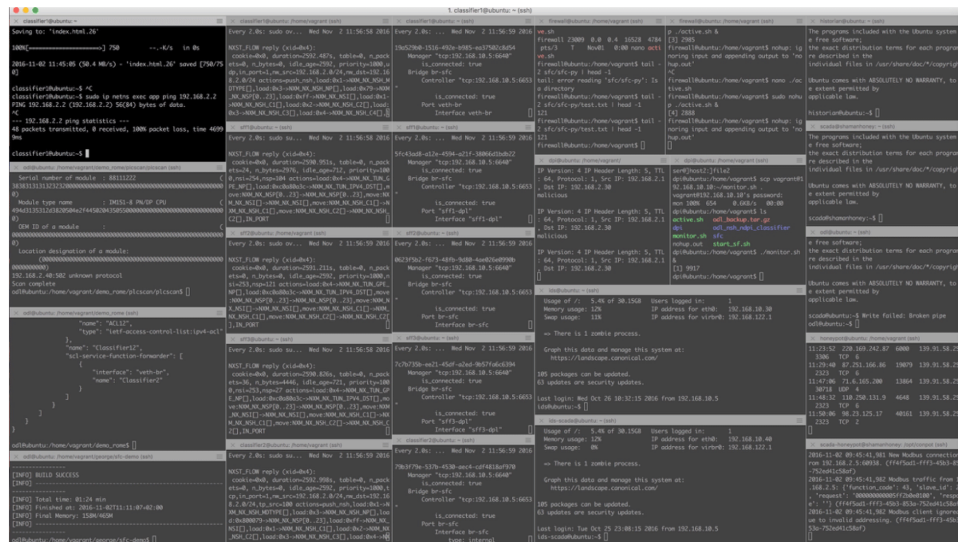


Fig. 6.27 OpenFlow Rules Inside the Forwarders and the Classifiers

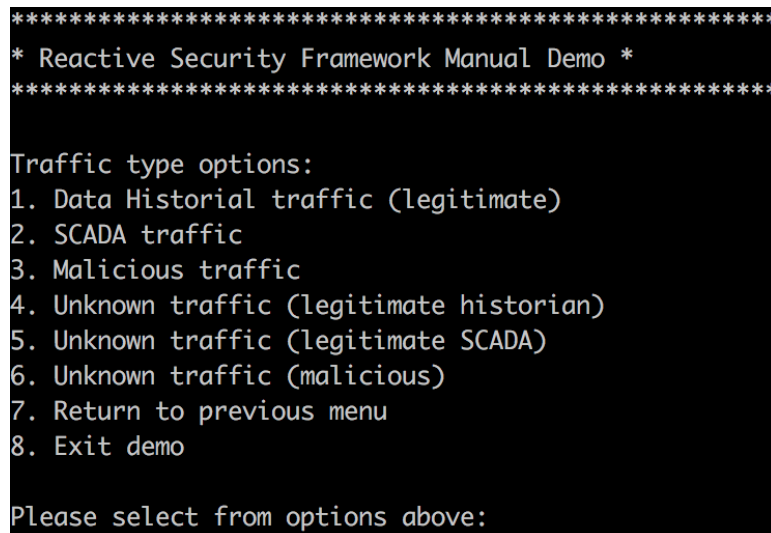


Fig. 6.28 Demo Shell Script

correlation between the number of functions and the delays is useful to evaluate the results of the real traces as presented below.

Considering the performance of the service function chaining, network traces captured from an operational wind park (in Brande, Denmark) are used to highlight the specificities of industrial traffic in the context of this application domain [189] and assess the presented performance evaluation in this context. The subject wind park consists of four wind turbines connected in a redundant star topology. Flows have an end-to-end latency requirement of 500 ms. The data to and from the SCADA server was gathered during approximately 1000

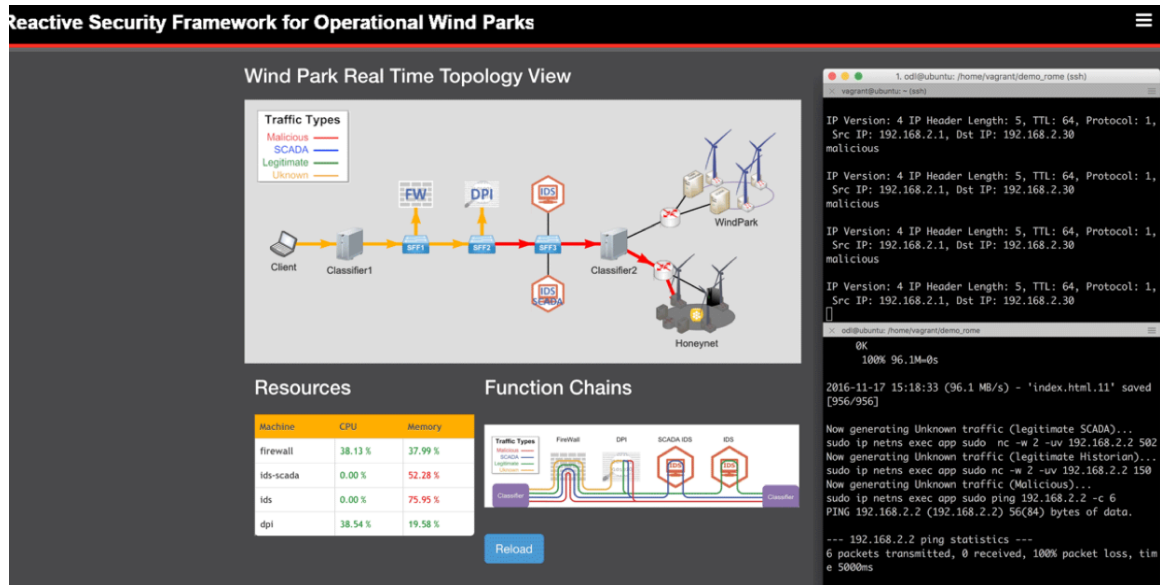


Fig. 6.29 Reactive Security - Demo

Table 6.9 Experimental Results of the SFC Reactive Security

Type	DPI	FW	IDS	SCADA IDS	End-Hosts	No Func	Delay
No Function					Anywhere	0	0,45 ms
Legitimate		O	O		Historian	2	66,57 ms
SCADA		O		O	SCADA	2	45,83 ms
Malicious		O			Honepot	1	13,53 ms
Unknown -> Legitimate	O	O	O		Historian	3	169,74 ms
Unknown -> SCADA	O	O		O	SCADA	3	138,92 ms
Unknown -> Malicious	O	O			Honepot	2	116,72 ms
All Functions	O	O	O	O	Anywhere	4	191,24 ms

seconds. Though critical, these flows have only end-to-end requirements of 100 ms, 250 ms or 500 ms depending on the specific service. The analysis of the traces has shown that industrial networks have very low data rate requirements. While a lot of short bursty flows exist, only a few of them have QoS requirements. Though these requirements can reach hundreds of milliseconds, a couple of critical flows have stringent low latency requirements of the order of tens of milliseconds.

Based on results in Table 6.9, an important observation of the service function chaining is applicable to operational wind parks, as the additional delays are affordable for most critical services (which all have latency requirements of *100ms*, *250ms* or *500ms*). Furthermore, although the emulated experiments of service SFC provisioning are conducted on VMs on the same server, thus minimising network delays, the multiple gigabit interconnections present in a wind park all feature low latencies (through over-provisioning), and are, therefore, expected

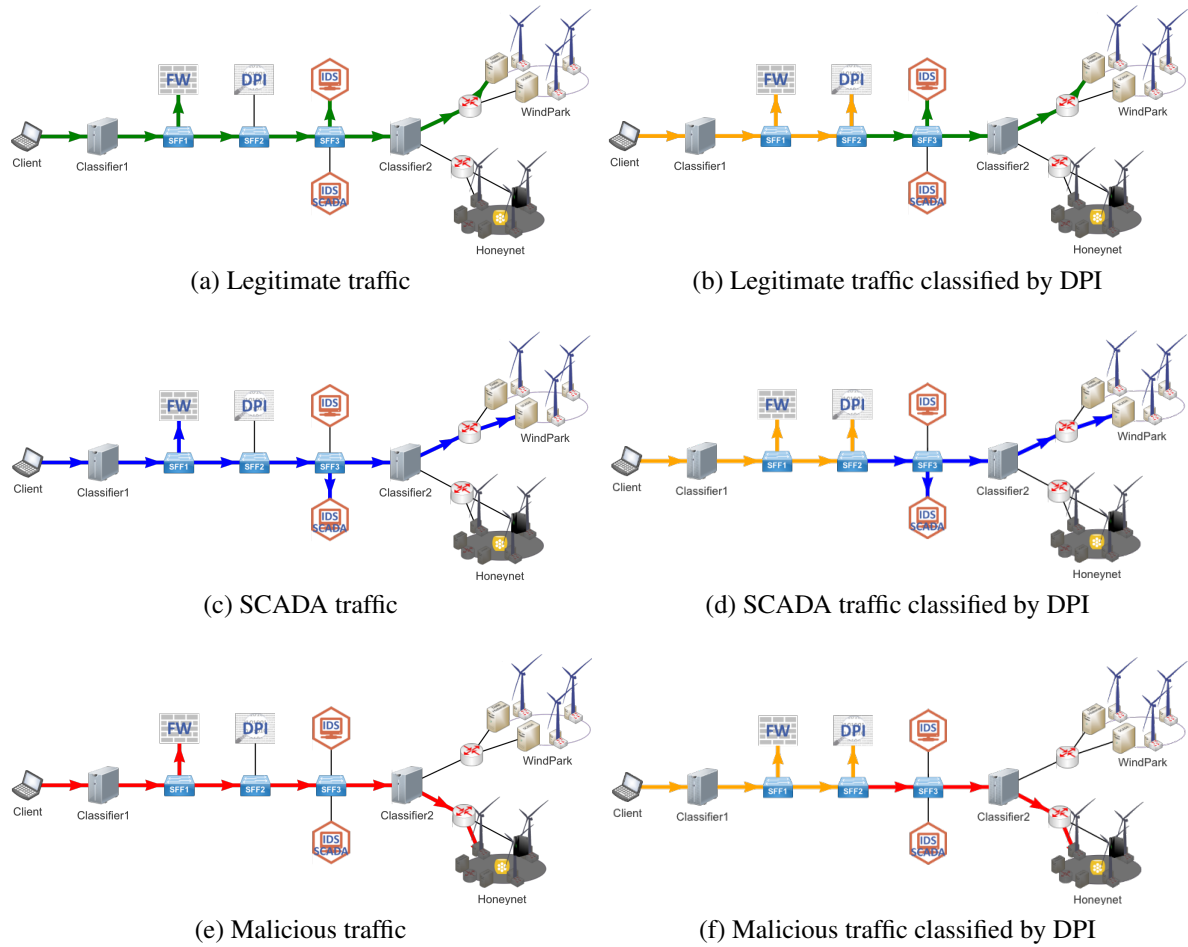


Fig. 6.30 Real-time Traffic Flows on the Controller's GUI

to introduce minimal delays. Finally, the evaluation of the proposed solution in an actual wind park, validates the feasibility of the approach, as it provides the necessary sophisticated, dynamic and continuous security monitoring required in industrial networks nowadays.

6.5 Summary

In this chapter, the developed design patterns were evaluated in the pattern framework to design network topologies, to validate reliability of the compositions and to guarantee system security and dependability and to apply service function chaining for secure traffic forwarding. More specifically, three different topology patterns were enforced and evaluated to design a number of network topologies with respect to connectivity with the line pattern, coverage with the mesh pattern and scalability with the tree pattern. The evaluation of design patterns

to satisfy S&D properties can be applied to design reliable networks with the reliability patterns, availability with the fault tolerance patterns and security with the encryption patterns. In addition, the design of secure industrial network can be benefited by the enforcement of service function chaining patterns. To prove the applicability of our pattern-based approach, all the developed patterns were inserted in the framework to support the design and verification of SDN/NFV-enabled network infrastructures and evaluated in certain initial experiments involving design and verification guaranteeing security and dependability. Leveraging SFC not only enhances the industrial network's security, but also decreases the performance impact of the security functions. Finally, the performance evaluation of the framework's implementation validates the feasibility of the approach, considering the current performance and requirements of industrial environment, as aggregated from an actual, operating wind park.

Chapter 7

Conclusion

7.1 Overview

In the final chapter of this thesis, summary and discussion of the research work are given. Furthermore, contributions in relation to the objectives are analysed. Moreover, evaluation and approach limitations compared to the achievements of the objectives are presented. Finally, short and long-term plans for future work and an extension of this research are also provided.

7.2 Summary of the Research

This work presented the definition of design patterns able to design secure and dependable SDN/NFV-enabled networks. S&D patterns express conditions and properties that need to be satisfied by abstract networks of different topologies in order to preserve particular end-to-end S&D properties. This research included the development of the S&D patterns used to minimise the effects of passive and active attacks on physical layer. Moreover, a pattern language, easily understandable and open to modifications was defined, based on the requirements of administrators and users. In addition, a pattern framework was proposed to enable the insertion of patterns to SDN/NFV-enabled network architectures, able to support network designs and to guarantee properties such as connectivity, scalability and coverage as well as the defined security and dependability properties. Flexibility of the framework to insert patterns expressed as rules, has rendered it able to guarantee properties and handle incidents. In addition, the pattern framework was extended in order to support reactive security leveraging SFC for SDN/NFV-enabled industrial networks. SFC can classify and steer traffic according to the instantiated service chains including a variety of security network

functions. To prove the applicability of the developed S&D patterns approach, a prototype network simulator was developed supporting the design and verification of SDN/NFV-enabled network infrastructures. The simulator was evaluated in initial experiments involving design and verification scenarios with security properties of confidentiality and availability.

7.3 Contributions

The contributions of our research can be summarised as follows:

- **Development of a pattern language for expressing design patterns**

One of the main contributions of this thesis is the definition of a pattern language for designing secure and dependable SDN/NFV-enabled networks. This language is based on formal verification and reasoning driven by production rules enabled by the use of the pattern engine. Pattern language can describe a variety of design patterns for designing large scale networks with functional and non-functional requirements. The semantics of the language defining pattern components and relationships are expressed as Java classes and are used by pattern rules in a Drools production rule format. This pattern-based language can be used to cover not only horizontally layered designs but also vertical ones for SDN/NFV-enabled networks and systems that preserve required S&D properties. Finally, the proposed pattern language can be used by experts and non-expert programmers as it offers an interface for fast interaction with different layers and components such as an SDN controller.

- **Development of various pattern instances**

Based on the developed language, a number of different pattern instances were developed to design network topologies with respect to S&D attributes. More specifically, the main properties guaranteed by the developed topology patterns are connectivity, scalability and coverage. In addition, the developed reliability, fault tolerance patterns can guarantee dependable networks. Furthermore, the developed security patterns can guarantee confidentiality through the deployed encryption patterns. Thus, SFC patterns were proposed to support service function instantiation/identification based on the respective instantiated service function chains. Finally, the path discovery pattern applicable to all patterns, enabled the capability for path decomposition in deployed network topologies.

- **Development of a pattern framework**

To provide proactive and reactive mechanism able to handle security incidents, failures,

faults and attacks, a pattern framework was developed. This pattern framework was able to host and enable an open and flexible way of designing networks, based on the enforcement of the patterns as expressed by the pattern rules. The framework enables multi-layer network support for the design of SDN/NFV-enabled networks or the interaction with the involved network elements. The ability of the pattern framework to interact with multiple distributed pattern engines on different layers, including pattern enforcement inside SDN controller or from the application layer, indicate its importance. Finally, the open architecture of the pattern framework and due to its proven capability to insert additional S&D patterns at runtime, presents great potentiality for further developments.

- **Deployment of reactive security leveraging SFC in the pattern framework**

One additional contribution of this research was the enablement of service function chaining to support reactive security, modelled (and deployable) to allow continuous monitoring of the industrial network and detailed analysis of the potential attacks. Its combination with the pattern framework ensured security functions to be proactively deployed through the respective patterns, whether the underlying network and service functions were virtualised or not. Moreover, the instantiation of the service functions chains through the pattern framework not only enhanced the industrial network's security, but also decreased the performance impact of the security functions. Finally, the potentiality of the pattern framework to interact with hypervisor for instantiating VNFs following SFC concept through the developed pattern instances underlined the importance of the framework.

- **Development of a network simulator for monitoring and management of network topologies**

In order to monitor and manage of deployed network topologies, a network simulator was developed. This simulator is able to interact with pattern engines and the output of the respective patterns to preview the data plane topologies of different SDN/NFV topologies. The current and the updated network topologies can be inserted inside the simulator in order to express the different interactions between forwarding devices such as switches, routers, gateways and end-hosts such clients, servers and IoT devices or service functions such as firewall, DPI and IDS. The simulator can support either SDN/NFV components or legacy ones. Moreover, the capability of the simulator to depict dynamically traffic forwarding through different chains and paths at runtime made the potentiality of the developed tool even more crucial. Finally, the open and

extensible architecture of the simulator can be used for a variety of different potential use cases.

- **Evaluation of the pattern framework**

The last contribution of this research included the evaluation of the developed framework and the enforcement of the different developed pattern instances with a number of different use cases and scenarios.

- The first use case included the design of network topologies based on the different developed network topologies patterns to support the deployment of the required network devices, in order to guarantee end-to-end connectivity, scalability and coverage. A number of different experiments were conducted measuring the required number of relay nodes for different distances and coverage requirements. The results of the related patterns enforcement in the pattern framework were presented and detailed. Moreover, the various network designs were deployed in the network simulator.
- The second use case was related to the design and verification of S&D SDN/NFV-enabled network topologies. Different scenarios were deployed to support reliability, fault tolerance and security through a number of conducted experiments. Apart from the recursive guarantee of reliability in network topologies, satisfied and evaluated by the respective reliability pattern, the evaluation of the fault tolerance in SDN infrastructure and the provision of security through the respective pattern proved the multidimensional applicability of the proposed framework.
- The last part of this evaluation included the provision of reactive security leveraging SFC in the pattern framework. An industrial use case was selected to validate the deployment of different service chains for the different types of traffics. The evaluation of this use case for the function virtualisation, the path finding and the traffic classification provided a complete solution for reactive security by predefining the respective chains at design and by forwarding the different kind of traffic through the respective service chains at runtime.

7.4 Limitations

The developed pattern framework and the pattern instances for the design of SDN/NFV-enabled networks present some limitations. These limitations concerning the objectives and the evaluation of the results are summarised below:

- **Design network limitations**

The design of networks with model-based approaches is feasible and may present very efficient and accurate results. However, these results may be affected by external factors in real world. For instance, the existence of buildings or the dissimilarity of terrain may interfere in optimum design networks. Moreover, different economic or energy constraints can affect the desirable results. For example, the installation of a number of intermediate nodes necessary to reach the required reliability percentage may be theoretically correct, however in real world this is hardly feasible due to the exponential increment of CAPEX and OPEX cost. Finally, the deployment of new network designs and their co-existence with older ones may also modify the properties guaranteed by new designs.

- **Pattern schema limitations**

The proposed pattern schema describes an approach on modelling network designs with respect to specific functional and non-functional properties. However, additional patterns and pattern parameters can be included in the proposed pattern schema. These may include the modelling of all parameters and factors as additional attributes of components. Therefore, the development of SDN and NFV network technologies can enable the exposure of related interfaces and system parameters in a standardised format able to be used by the design patterns.

- **Design pattern limitations**

Design patterns are mainly used for software integration and service composition. Therefore, one of the most important challenges faced in this thesis was the transition from software engineering to network engineering compositions. For instance, the call or the replacement of web service is different that calling or replacing of a network switch. Moreover, the interaction between two components requires the existence of a network link where in the service composition the network is transparent. These boundaries have affected the development of the respective network design patterns.

- **Service provision imitations**

As far as the application of design patterns is concerned , limitations include available resources for the distribution of physical nodes to support the required virtual ones. In addition, the instantiation of the respective service chains hypothesis that all the intermediate switches are Openflow and NSH enabled ones. Otherwise the instantiation of the respective should be done through VLAN to applying service chaining requiring additional intermediate configurations.

7.5 Future Work

This research proposed a pattern-based framework that is able to design SDN/NFV-enabled networks infrastructures satisfying specific security and dependability properties based on predefined pattern instances. In the future, this work can be expanded in order to support additional functionalities, properties and network designs. In this section, some directions for future research are listed:

- **Pattern language extension**

The pattern specification language could be extended to support additional patterns suitable to guarantee a variety of network and application properties and attributes. In order to apply a pattern language extension, further tools, such as new classes development and exposed interface, will be necessary to support horizontal and vertical layer interaction and semantic interoperability.

- **Additional pattern instances**

Additional patterns can be developed to support a variety of functional and non-functional properties. In the current work, patterns were able to guarantee security and dependability by design at the SDN/NFV data plane. A future work, may concern the development of new pattern instances able to guarantee security and dependability by design of the control plane. For instance, this may be realized by protecting the control plane from security attacks with installation of malicious controllers, man in the middle and Byzantine fault tolerance as appeared to be a very crucial part on SDN/NFV-enabled infrastructures.

- **Development of a global pattern framework**

Pattern framework can be enriched with new functional capabilities to cover not only horizontally layered designs but also vertical layers of SDN/NFV-enabled architectures. This can be realised by deploying multi-layer pattern engines handled centrally by a pattern orchestrator who has a global view of the current conditions of the network and additional components. The important contribution of our work is highlighted by the advantage of this framework to be attached to different systems and architectures.

- **Fully integration with NFV MANO**

The framework can be enhanced via the fully automated use of the NFV MANO software stack and the definition of the service templates at the MANO to instantiate the required templates responsible for the boot-up of the necessary VMs using a VIM software. In turn, the MANO can be used to program the ODL Controller accordingly,

passing the necessary information to the SFC Manager. This will also ensure a more accurate monitoring of the Service Functions' resources (e.g. allowing the instantiation of additional VMs when one of the existing functions is overloaded).

- **Security service function improvements**

Security service functions can be Improved in order to support further integration with the pattern instances not only at design time but also at runtime. This may include the extension of the packet inspection, enforced by patterns, to enable DPI functionalities (essential for traffic-type classification), in order to minimise the impact of the framework on the network's performance and to enable its use in more time-critical industrial applications.

- **Network simulator upgrade**

The developed network simulator was proposed to depict and monitor the design and the condition of network topologies. Its consequent capability to integrate them with the pattern framework, creates immense opportunities for further development to support additional potentialities. These may include the direct interaction with physical and virtual component and topology deployment together with the enforcement of pattern instances used to guarantee different functional and non-functional properties of SDN/NFV-enabled infrastructures.

References

- [1] NGMN Alliance. 5g white paper. *Next generation mobile networks, white paper*, 2015.
- [2] NGMN Alliance. 5g network and service management including orchestration. *Next generation mobile networks*, 2019.
- [3] Patrick Agyapong, Mikio Iwamura, Dirk Staehle, Wolfgang Kiess, and Anass Benjebbour. Design considerations for a 5g network architecture. volume 52, pages 65–75. IEEE, 2014.
- [4] Mario Hermann, Tobias Pentek, and Boris Otto. Design principles for industrie 4.0 scenarios. In *System Sciences (HICSS), 2016 49th Hawaii International Conference on*, pages 3928–3937. IEEE, 2016.
- [5] Mauricio Jose da Silva Theo Lins and Ricardo Augusto Rabelo Oliveira. Software-defined networking for industry 4.0. In *20th Advanced International Conference on Telecommunications*, 2016.
- [6] Nikolaos Petroulakis, Toktam Mahmoodi, Vivek Kulkarni, Andreas Roos, Petra Vizarreta, Khawar Abbasik, Xavier Vilajosana, Spiros Spirou, Anton Masiuk, Ermin Sakic, et al. Virtuwind: Virtual and programmable industrial network prototype deployed in operational wind park. 2016.
- [7] Eric D Knapp and Joel Thomas Langill. Industrial network security: Securing critical infrastructure networks for smart grid, scada, and other industrial control systems. 2014.
- [8] Teemu Koponen, Martin Casado, Natasha Gude, and Jeremy Stribling. Distributed control platform for large-scale production networks, September 9 2014. US Patent 8,830,823.
- [9] Nikos Bizanis and Fernando A Kuipers. Sdn and virtualization solutions for the internet of things: A survey. *IEEE Access*, 4:5591–5606, 2016.
- [10] Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo. Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In *IEEE Computer Communications (INFOCOM)*, pages 513–521. IEEE, 2015.
- [11] Open Networking Foundation (ONF). www.opennetworking.org/.

- [12] Opendaylight: Open source sdn platform. <https://www.opendaylight.org>.
- [13] Onos: Open network operating system. <https://www.onosproject.org>.
- [14] Sharone Zitzman. What is sdn? sdn controllers wiki and roundup -.opendaylight, openflow, network automation and more.
- [15] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.
- [16] Yakov Rekhter, Tony Li, and Susan Hares. A border gateway protocol 4 (bgp-4). Technical report, 2005.
- [17] JL Le Roux. Path computation element (pce) communication protocol (pcep). 2009.
- [18] Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. Network configuration protocol (netconf). *Network*, 2011.
- [19] IETF. Alto: Application layer traffic optimization for distributed topologies.
- [20] Hyperflow: A distributed control plane for openflow, author=Tootoonchian, Amin and Ganjali, Yashar, booktitle = Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, year=2010.
- [21] ISGNFV ETSI. Network functions virtualisation (nfv); ecosystem; report on sdn usage in nfv architectural framework. *ETSI GS NFV-EVE*, 5:V1, 2015.
- [22] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60. ACM, 2013.
- [23] A Belmonte Martin, L Marinos, E Rekleitis, G Spanoudakis, and NE Petroulakis. Threat landscape and good practice guide for software defined networks/5g. 2015.
- [24] Cyber-Attack Against Ukrainian Critical Infrastructure. *ICS-CERT, Alert (IR-ALERT-H-16-056-01)*, 2015.
- [25] NERC Standard CIP. 007-6-Cyber Security-Systems Security Management. 2013.
- [26] Paul Quinn and Tom Nadeau. Problem Statement for Service Function Chaining, apr 2015.
- [27] Paul Browne. *JBoss Drools Business Rules*. Packt Publishing Ltd, 2009.
- [28] J.P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann Publishers Inc., 2010.
- [29] J. Laprie. Dependable computing and fault-tolerance. *Digest of Papers FTCS-15*, pages 2–11, 1985.

- [30] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, and F. Springsteel. *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. 1991.
- [31] P. Park, P. Di Marco, C. Fischione, and K. Johansson. Modeling and optimization of the ieee 802.15. 4 protocol for reliable and timely communications. *Parallel and Distributed Systems*, 24(3), 2013.
- [32] Jue Chen, Jinbang Chen, Fei Xu, Min Yin, and Wei Zhang. *When Software Defined Networks Meet Fault Tolerance: A Survey*, pages 351–368. Springer International Publishing, Cham, 2015.
- [33] Dhiren R Patel. *Information security: theory and practice*. PHI Learning Pvt. Ltd., 2008.
- [34] Na Li, Nan Zhang, Sajal K. Das, and Bhavani Thuraisingham. Privacy preservation in wireless sensor networks: A state-of-the-art survey. *Ad Hoc Networks*, 7(8):1501–1514, November 2009.
- [35] Celal Ozturk, Yanyong Zhang, and Wade Trappe. Source-location privacy in energy-constrained sensor network routing. *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks - SASN '04*, page 88, 2004.
- [36] Georgios Mantas, Nikos Komninos, J Rodriuez, Evariste Logota, and Hugo Marques. Security for 5g communications. 2015.
- [37] Panagiotis Demestichas, Andreas Georgakopoulos, Dimitrios Karvounas, Kostas Tsagkaris, Vera Stavroulaki, Jianmin Lu, Chunshan Xiong, and Jing Yao. 5g on the horizon: key challenges for the radio-access network. *IEEE Vehicular Technology Magazine*, 8(3):47–53, 2013.
- [38] Alexandros G Fragkiadakis, Elias Z Tragos, and Ioannis G Askoxylakis. A survey on security threats and detection techniques in cognitive radio networks. *IEEE Communications Surveys & Tutorials*, 15(1):428–445, 2013.
- [39] Ruiliang Chen, Jung-Min Park, Y Thomas Hou, and Jeffrey H Reed. Toward secure distributed spectrum sensing in cognitive radio networks. *IEEE Communications Magazine*, 46(4), 2008.
- [40] A. Fragkiadakis, V. Siris, and N. Petroulakis. Anomaly-based intrusion detection algorithms for wireless networks. In *the 8th WWIC 2010*, June 2010.
- [41] Nikolaos E Petroulakis, Elias Z Tragos, and Ioannis G Askoxylakis. An experimental investigation on energy consumption for secure life-logging in smart environments. In *IEEE CAMAD*, pages 292–296. IEEE, 2012.
- [42] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, 17(2):20–27, 2013.
- [43] Michael Pearce, Sherali Zeadally, and Ray Hunt. Virtualization: Issues, security threats, and solutions. *ACM Computing Surveys (CSUR)*, 45(2):17, 2013.

- [44] ETSI NFV ISG. Network functions virtualisation (nfv)-nfv security: Problem statement. *White paper*, 2014.
- [45] ETSI NFV ISG. Network functions virtualisation (nfv); nfv security; security and trust guidance. *White paper*, 2016.
- [46] R Guerzoni et al. Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In *SDN and OpenFlow World Congress*, volume 1, pages 5–7, 2012.
- [47] D. E. Britton. Formal verification of a secure network with end-to-end encryption. In *1984 IEEE Symposium on Security and Privacy*, pages 154–154, April 1984.
- [48] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *3rd IEEE European Symposium on Security and Privacy (EuroS&P 2018)*, 2018.
- [49] AbdelNasir Alshamsi and Takamichi Saito. A technical comparison of ipsec and ssl. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, volume 2, pages 395–398. IEEE, 2005.
- [50] Mark Needleman. The internet engineering task force. *Serials Review*, 26(1):69–72, 2000.
- [51] R Lopez and G Lopez-Millan. Software-defined networking (sdn)-based ipsec flow protection. In *Internet Engineering Task Force, Internet-Draftdraft-ietf-i2nsf-sdn-ipsec-flow-protection-00*, 2017.
- [52] Robert W Shirey. Internet security glossary, version 2. *IETF*, 2007.
- [53] Cisco. Security for VPNs with IPsec Configuration Guide, Cisco IOS XE Release 3S. 2017.
- [54] CISCO. Chapter 8: Implementing virtual private networks: Ccna security. 2008.
- [55] Hazem Hamed, Ehab Al-Shaer, and Will Marrero. Modeling and verification of ipsec and vpn security policies. In *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, pages 10–pp. IEEE, 2005.
- [56] Riaz Ahmed Shaikh, Hassan Jameel, Brian J D’Auriol, Heejo Lee, Sungyoung Lee, and Young-Jae Song. Achieving network level privacy in Wireless Sensor Networks. *Sensors (Basel, Switzerland)*, 10(3):1447–72, January 2010.
- [57] Veeranna, M.Venkata Reddy Krishna, and D. Jamuna. Enhancement of Privacy Level in Wireless Sensor Network. *ijecse.com*, 1(September):1024–1029, 2012.
- [58] Yih-Chun Hu and Helen J. Wang. A framework for location privacy in wireless networks. *ACM SIGCOMM Asia Workshop*, 2005.
- [59] Xi Luo, Xu Ji, and Myong-Soon Park. Location Privacy against Traffic Analysis Attacks in Wireless Sensor Networks. *2010 International Conference on Information Science and Applications*, pages 1–6, 2010.

- [60] Marco Gruteser, Graham Schelle, and Ashish Jain. Privacy-aware location sensor networks. *Proceedings of the 9th . . .*, 2003.
- [61] Pandurang Kamat, Yanyong Zhang, Wade Trappe, and Celal Ozturk. Enhancing Source-Location Privacy in Sensor Network Routing. *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 599–608, 2005.
- [62] Kanika Grover, Alvin Lim, and Qing Yang. Jamming and anti-jamming techniques in wireless networks: a survey. *International Journal of Ad Hoc and Ubiquitous Computing*, 17(4):197–215, 2014.
- [63] Vishnu Navda, Aniruddha Bohra, Samrat Ganguly, and Dan Rubenstein. Using channel hopping to increase 802.11 resilience to jamming attacks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 2526–2530. IEEE, 2007.
- [64] VA Siris and M Delakis. Interference-aware channel assignment in a metropolitan multi-radio wireless mesh network with directional antennas. *Computer Communications*, 2011. Query date: 2012-10-13.
- [65] Haowen Chan and Andrian Perrig. Security In Networks. *IEEE Computer*, 36(10)(October):103–105, 2003.
- [66] Kai Xing, Shyaam Sundhar, Rajamadam Srinivasan, and Manny Rivera. Attacks and Countermeasures in Sensor Networks : A Survey A wireless sensor network (WSN) is comprised of a large number of sensors that. pages 1–28, 2005.
- [67] E Bayraktaroglu, C King, and X Liu. On the Performance of IEEE 802.11 under Jamming. *Infocom/08*, 2008.
- [68] Alexandros G Fragkiadakis, Elias Z Tragos, Theo Tryfonas, and Ioannis G Askoxylakis. Design and performance evaluation of a lightweight wireless early warning intrusion detection prototype. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):73, 2012.
- [69] Incheol Shin, Yilin Shen, Ying Xuan, MT Thai, and T Znati. Reactive jamming attacks in multi-radio wireless sensor networks: an efficient mitigating measure by identifying trigger nodes. *FOWANC'09*, pages 87–96, 2009.
- [70] V. Bhuse, a. Gupta, and a. Al-Fuqaha. Detection of Masquerade Attacks on Wireless Sensor Networks. *2007 IEEE International Conference on Communications*, pages 1142–1147, June 2007.
- [71] Jaydip Sen. A Survey on Wireless Sensor Network Security. *International Journal of Communication Networks and Information Security (IJCNIS)*, Vol. 1, No. 2(2):55–78, 2009.
- [72] Siwar Ben Hadj Said, Bernard Cousin, and Samer Lahoud. Software defined networking (sdn) for reliable user connectivity in 5g networks. In *Network Softwarization (NetSoft), 2017 IEEE Conference on*, pages 1–5. IEEE, 2017.

- [73] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 121–126. ACM, 2012.
- [74] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, pages 37–44. ACM, 2010.
- [75] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, Jennifer Rexford, et al. A nice way to test openflow applications. In *NSDI*, volume 12, pages 127–140, 2012.
- [76] Seuk Son, Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Model checking invariant security properties in openflow. In *Communications (ICC), 2013 IEEE International Conference on*, pages 1974–1979. IEEE, 2013.
- [77] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.
- [78] Adnan Akhunzada, Ejaz Ahmed, Abdullah Gani, Muhammad Khan, Muhammad Imran, and Sghaier Guizani. Securing software defined networks: taxonomy, requirements, and open issues. *Communications Magazine, IEEE*, 53(4):36–44, 2015.
- [79] Open Networking Foundation. Openflow specification 1.5. 2010.
- [80] M. Wasserman, S. Hartman, and D. Zhang. Security analysis of the open networking foundation (onf) openflow switch specification. *IETF*, 2012.
- [81] What is the difference between tls 1.3 and tls 1.2? 2019.
- [82] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, Anja Feldmann, et al. Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks. In *USENIX ATC*, 2014.
- [83] Jon Matias, Jokim Garay, Alaitz Mendiola, Nerea Toledo, and Eduardo Jacob. Flownac: Flow-based network access control. In *Software Defined Networks (EWSN), 2014 Third European Workshop on*, pages 79–84. IEEE, 2014.
- [84] Seungwon Shin, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 78–89. ACM, 2014.
- [85] Xitao Wen, Yan Chen, Chengchen Hu, Chao Shi, and Yi Wang. Towards a secure controller platform for openflow applications. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 171–172. ACM, 2013.

- [86] Seungwon Shin, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.
- [87] Ryan Wallner and Robert Cannistra. An sdn approach: quality of service using big switch’s floodlight open-source controller. *Proceedings of the Asia-Pacific Advanced Network*, 35:14–19, 2013.
- [88] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424. ACM, 2013.
- [89] The Challenge. Defenseflow – sdn based network ddos , application dos and apt protection. 2014.
- [90] HP. Realizing the power of SDN with HP Virtual Application Networks. *Technical White Paper*, 2012.
- [91] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.
- [92] Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster. Fattire: Declarative fault tolerance for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 109–114. ACM, 2013.
- [93] Hyojoon Kim, Mike Schlansker, Jose Renato Santos, Jean Tourrilhes, Yoshio Turner, and Nick Feamster. Coronet: Fault tolerance for software defined networks. In *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pages 1–2. IEEE, 2012.
- [94] Naga Katta, Haoyu Zhang, Michael Freedman, and Jennifer Rexford. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 4. ACM, 2015.
- [95] Fábio Botelho, Alysson Bessani, Fernando MV Ramos, and Paulo Ferreira. On the design of practical fault-tolerant sdn controllers. pages 73–78, 2014.
- [96] Balakrishnan Chandrasekaran and Theophilus Benson. Tolerating sdn application failures with legosdn. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 22. ACM, 2014.
- [97] Teemu Koponen, Martín Casado, Natasha Gude, Jeremy Stribling, Leonid B. Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *OSDI*, 2010.

- [98] Maciej Kuźniar, Peter Perešini, Nedeljko Vasić, Marco Canini, and Dejan Kostić. Automatic failure recovery for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 159–160. ACM, 2013.
- [99] Karim ElDefrawy and Tyler Kaczmarek. Byzantine fault tolerant software-defined networking (sdn) controllers. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 2, pages 208–213. IEEE, 2016.
- [100] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. State machine replication for the masses with bft-smart. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 355–362. IEEE, 2014.
- [101] He Li, Peng Li, Song Guo, and Amiya Nayak. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. *IEEE Transactions on Cloud Computing*, 2(4):436–447, 2014.
- [102] Sandra Scott-Hayward, Gemma O’Callaghan, and Sakir Sezer. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pages 1–7. IEEE, 2013.
- [103] S. Kumar, M. Tufail, S. Majee, C Captari, and S. Homma. Service Function Chaining Use Cases in Data Centers. 2016.
- [104] W. Haeffner, J. Napper, M. Stiernerling, D. Lopez, and J. Uttaro. Service Function Chaining Use Cases in Mobile Networks. 2015.
- [105] Wolfgang John, Konstantinos Pentikousis, George Agapiou, Eduardo Jacob, Mario Kind, Antonio Manzalini, Fulvio Risso, Dimitri Staessens, Rebecca Steinert, and Catalin Meirosu. Research Directions in Network Service Chaining. In *IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7. IEEE, nov 2013.
- [106] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks, no. rfc 7348. Technical report, 2014.
- [107] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.
- [108] Liberios Vokorokos, Michal Ennert, Marek Cajkovský, and Ján Radušovský. A Survey of parallel intrusion detection on graphical processors. *Open Computer Science*, 4(4), jan 2014.
- [109] A Bremler-Barr, Y Harchol, D Hay, and Y Koral. Deep Packet inspection as a service. In *10th ACM International Conference on Emerging Networking Experiments and Technologies, CoNEXT 2014*, pages 271–282, 2014.
- [110] J Halpern and C Pignataro. Service function chaining (sfc) architecture. Technical report, 2015.

- [111] L4-L7 Service Function Chaining Solution Architecture. *Open Networking Foundation*, pages 1–36, 2015.
- [112] Service function chaining (sfc) working group. <https://datatracker.ietf.org/wg/sfc/charter/>.
- [113] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138 – 155, 2016.
- [114] Paul Quinn and Jim Guichard. Service function chaining: Creating a service plane via network service headers. *Computer*, 47(11):38–44, 2014.
- [115] P. Quinn and U. Elzur. Network Service Header. *Network Working Group, IETF Draft*, 2016.
- [116] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Geoffrey Lefebvre, Ravi Manghir-malani, Ramesh Mishra, Ritun Patneyt, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, and Mallik Tatipamula. StEERING: A software-defined network-ing for inline service chaining. In *Proceedings - International Conference on Network Protocols, ICNP*, 2013.
- [117] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying middlebox policy enforcement using SDN. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13*, page 27, New York, New York, USA, 2013. ACM Press.
- [118] GS NFV-SEC 013 ETSI. Network functions virtualisation (nfv) release 3; security; security management and monitoring specification. 2017.
- [119] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *2014 IEEE 3rd International Conference on Cloud Networking, CloudNet 2014*, pages 7–13, 2014.
- [120] Jeremias Blendin, Julius Ruckert, Nicolai Leymann, Georg Schyguda, and David Hausheer. Position paper: Software-defined network service chaining. In *Proceedings - 2014 3rd European Workshop on Software-Defined Networks, EWSDN 2014*, pages 109–114, 2014.
- [121] Microservices a definition of this new architectural term. <http://martinfowler.com/articles/microservices.html>.
- [122] Johannes Thönes. Microservices. *IEEE Software*, 32(1), 2015.
- [123] Microservices five architectural constraints. <http://www.nirmata.com/2015/02/microservices-five-architectural\protect\discretionary{\char\hyphenchar\font}{ }{ }constraints/>.
- [124] DC. Schmidt. Model-driven engineering. *Computer Society*, pages 286–298, 2006.
- [125] Krishna Doddapaneni, Enver Ever, Orhan Gemikonakli, Ivano Malavolta, Leonardo Mostarda, and Henry Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *SESENA*, 2012.

- [126] Felipe A Lopes, Marcelo Santos, Robson Fidalgo, and Stenio Fernandes. Model-driven networking: A novel approach for sdn applications development. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 770–773. IEEE, 2015.
- [127] Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner. Applying patterns to model-driven development of automation systems: an industrial case study. In *Proceedings of the 17th European Conference on Pattern Languages of Programs*, page 5. ACM, 2012.
- [128] Brahim Hamid, Christian Percebois, and Damien Gouteux. A methodology for integration of patterns with validation purpose. In *Proceedings of the 17th European Conference on Pattern Languages of Programs*, page 8. ACM, 2012.
- [129] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.
- [130] G. Spanoudakis and S. Kokolakis. *Security and Dependability for Ambient Intelligence*. Springer Science & Business Media, 2009.
- [131] Haralambos Mouratidis. *Software Engineering for Secure Systems: Industrial and Research Perspectives: Industrial and Research Perspectives*. IGI Global, 2010.
- [132] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013.
- [133] Eduardo Fernandez-Buglioni. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.
- [134] Brahim Hamid, Jacob Geisel, Adel Ziani, Jean-Michel Bruel, and Jon Perez. Model-driven engineering for trusted embedded systems based on security and dependability patterns. In *SDL 2013: Model-Driven Dependability Engineering*, pages 72–90. Springer, 2013.
- [135] H. Petritsch. Service-oriented architecture (soa) vs. component based architecture. *Vienna University of Technology white paper*, available at <http://whitepapers.techre-public.com.com/abstract.aspx>, 2006.
- [136] G. Gössler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 2005.
- [137] W. Van Der Aalst, A. Ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and parallel databases*, 14(1), 2003.
- [138] Michael C Jaeger, Gregor Rojec-goldmann, and M Gero. QoS Aggregation for Web Service Composition using Workflow Patterns. Number Edoc, 2004.
- [139] L. Pino, K. Mahbub, and G. Spanoudakis. Designing Secure Service Workflows in BPEL. In *12th International Conference, ICSOC 2014, Paris, France, November 3-6, 2014.*, 2014.

- [140] L. Pino, G. Spanoudakis, A. Fuchs, and S. Gürgens. Discovering secure service compositions. In *4th International Conference on Cloud Computing and Services Sciences, Barcelona, Spain*, 2014.
- [141] Dietmar Kühl. Design patterns for the implementation of graph algorithms. In *MASTER'S THESIS, TECHNISCHE UNIVERSITÄT*. Citeseer, 1996.
- [142] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. Association rules with graph patterns. *Proceedings of the VLDB Endowment*, 8(12):1502–1513, 2015.
- [143] Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. volume 71, pages 1–30. Elsevier, 2014.
- [144] Wei Zhou, Li Li, Min Luo, and Wu Chou. Rest api design patterns for sdn northbound api. In *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*, pages 358–365. IEEE, 2014.
- [145] Reuven Cohen, Katherine Barabash, Benny Rochwerger, Liran Schour, Daniel Crisan, Robert Birke, Cyriel Minkenberg, Mitch Gusat, Renato Recio, and Vinesh Jain. An intent-based approach for network virtualization. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 42–50. IEEE, 2013.
- [146] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.
- [147] Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2001.
- [148] Leon Sterling. Patterns for prolog programming. In *Computational logic: Logic programming and beyond*, pages 374–401. Springer, 2002.
- [149] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(1):17–37, 1982.
- [150] Matthias Tichy, Daniela Schilling, and Holger Giese. Design of self-managing dependable systems with uml and fault tolerance patterns. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 105–109. ACM, 2004.
- [151] Titos Saridakis. A system of patterns for fault tolerance. In *EuroPLoP*, pages 535–582, 2002.
- [152] Robert Hanmer. *Patterns for fault tolerant software*. John Wiley & Sons, 2013.
- [153] Ferran Adelantado and Christos Verikoukis. Detection of malicious users in cognitive radio ad hoc networks: A non-parametric statistical approach. *Ad Hoc Networks*, 11(8):2367–2380, 2013.
- [154] Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. Modular sdn programming with pyretic. 2013.
- [155] Alireza Shameli Sendi, Yosr Jarraya, Makan Pourzandi, and Mohamed Cheriet. Efficient provisioning of security service function chaining using network security defense patterns. *IEEE Transactions on Services Computing*, 2016.

- [156] David Dolson, Michael Marchetti, and Kyle Larose. Efficient Patterns for Service Function Chaining within Network Function Virtualization Infrastructure . Internet-Draft draft-dolson-sfc-nfv-patterns-00, Internet Engineering Task Force, March 2016. Work in Progress.
- [157] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25(27):79–80, 1995.
- [158] Avelino J Gonzalez and Douglas D Dankel. *The engineering of knowledge-based systems: theory and practice*. Prentice-Hall, Inc., 1993.
- [159] Mélanie König, Michel Leclerc, Marie-Laure Mugnier, and Michaël Thomazo. A sound and complete backward chaining algorithm for existential rules. In *International Conference on Web Reasoning and Rule Systems*, pages 122–138. Springer, 2012.
- [160] Robert Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114–121. IEEE, 1971.
- [161] Ajlan Al-Ajlan. The comparison between forward and backward chaining. *International Journal of Machine Learning and Computing*, 5(2):106, 2015.
- [162] A Knopfmacher and ME Mays. Graph compositions i: Basic enumeration. *Integers: Electronic Journal of Combinatorial Number Theory*, 1:A04, 2001.
- [163] S. Hsieh and C. Chen. Component-composition graphs: (t,k)-diagnosability and its application. *IEEE Transactions on Computers*, 62:1097–1110, 06 2013.
- [164] Sfc environment security requirements. [https://tools.ietf.org/html/draft-mglt-sfc/securityenvironment-req-01](https://tools.ietf.org/html/draft-mglt-sfc-securityenvironment-req-01).
- [165] H. Friis. A note on a simple transmission formula. *IRE*, 34(5), 1946.
- [166] Wil MP van der Aalst, Alexander Hirschnall, and HMW Verbeek. An alternative way to analyze workflow graphs. In *International Conference on Advanced Information Systems Engineering*, pages 535–552. Springer, 2002.
- [167] I. Buckley, E. Fernandez, G. Rossi, and S. Sadjadi. Web services reliability patterns. In *SEKE*, pages 4–9, 2009.
- [168] Ming Wang and Qiao Li. Conditional edge connectivity properties, reliability comparisons and transitivity of graphs. *Discrete Mathematics*, 258(1-3):205–214, 2002.
- [169] D. Coit and A. Smith. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability*, 45(2), 1996.
- [170] J. Lin, S. Sedigh, and A. Miller. Modeling cyber-physical systems with semantic agents. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*. IEEE, 2010.
- [171] Hugo Krawczyk. Perfect forward secrecy. *Encyclopedia of Cryptography and Security*, pages 921–922, 2011.

- [172] Christoph G Günther. An identity-based key-exchange protocol. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 29–37. Springer, 1989.
- [173] S. Lee, S. Pack, M-K. Shin, and E. Paik. SFC dynamic instantiation. Internet-Draft draft-lee-sfc-dynamic-instantiation-01, Internet Engineering Task Force, October 2014. Work in Progress.
- [174] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 98–106. IEEE, 2015.
- [175] Milad Ghaznavi, Nashid Shahriar, Reaz Ahmed, and Raouf Boutaba. Service function chaining simplified. *arXiv preprint arXiv:1601.00751*, 2016.
- [176] Alireza Shameli Sendi, Yosr Jarraya, Makan Pourzandi, and Mohamed Cheriet. Efficient Provisioning of Security Service Function Chaining Using Network Security Defense Patterns. *IEEE Transactions on Services Computing*, (November):1–1, 2016.
- [177] Snort. <http://blog.snort.org/2012/01/snort-292-scada-preprocessors.html>.
- [178] Snort 2.9.2: Scada preprocessors. <http://www.snort.org>.
- [179] Honeyd. <https://github.com/sk4ld/gridpot>.
- [180] Scada honeynet project. <http://scadahoneynet.sourceforge.net>.
- [181] Panos Chatziadam, Ioannis G. Askoxylakis, and Alexandros Fragkiadakis. A network telescope for early warning intrusion detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8533 LNCS, pages 11–22, 2014.
- [182] Open source security. <https://pfsense.org/>.
- [183] Vm-series: Next-generation security for private and public clouds. <https://www.paloaltonetworks.com/>.
- [184] ndpi: Open and extensible lgplv3 deep packet inspection library. <http://www.ntop.org/products/deep-packet-inspection/ndpi/>.
- [185] Nodejs library. <http://www.nodejs.org>.
- [186] ETSI Group Specification NFV 002. Network functions virtualisation (nfv); architectural framework.
- [187] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015.
- [188] Open virtual switch. <http://www.openvswitch.org>.
- [189] Project VirtuWind. Deliverable D3.2: Detailed Intra-Domain SDN & NFV Architecture, 2017.

Appendix A

Developed Java Classes

Implemented Java Classes for the Pattern Framework

Graph.java

```
1 import java.util.HashSet;
2 import java.util.Objects;
3 public class Graph {
4     public Node node;
5     public Link link;
6     public HashSet<Node> nodes = new HashSet<Node>();
7     public HashSet<Link> links = new HashSet<Link>();
8     public boolean directed = true;
9
10    public Graph() {}
11    public Graph(String type) {
12        if (type=="undirected") {
13            this.directed = false;
14        }
15    }
16    public Graph(HashSet<Node> nodes, HashSet<Link> links) {
17        this.nodes = nodes;
18        this.links = links;
19    }
20    public HashSet<Node> getNodes() {
21        return nodes;
```

```
22     }
23     public HashSet<Link> getLinks() {
24         return links;
25     }
26     public HashSet<Node> addNode(Node node) {
27         getNodes().add(node);
28         return nodes;
29     }
30     public HashSet<Node> addNodes(HashSet<Node> nodes) {
31         for (Node node : nodes) {
32             addNode(node);
33         }
34         return nodes;
35     }
36     public HashSet<Link> addLink(Link link) {
37         if (getDirected() == false) {
38             getLinks().add(new Link(link.dst, link.src));
39         }
40         getLinks().add(link);
41         return links;
42     }
43     private boolean getDirected() {
44         return directed;
45     }
46     public HashSet<Link> addLinks(HashSet<Link> links) {
47         for (Link link : links) {
48             addLink(link);
49         }
50         return links;
51     }
52     public static class SubGraph extends Graph{
53         public SubGraph() {}
54     }
55 }
```


Node.java

```
1 import java.util.ArrayList;
2 import java.util.List;
3 public class Node {
4     public Integer id;
5     public String idS;
6     public String name;
7     public String type, group, label;
8     public String ip, address;
9     public String port;
10    public Point point;
11    public double range;
12    public List<String> ports = new ArrayList<String>();
13    public Resource resource;
14    public Node src, dst;
15    public Node() {}
16    public Node(Integer id, String group, Point point) {
17        this.id = id;
18        this.label = "n"+id.toString();
19        this.group = group;
20        this.point = point;
21    }
22    public Node(Integer id, String label, String group, Point
        point) {
23        this.id = id;
24        this.label = label;
25        this.group = group;
26        this.point = point;
27    }
28    public Node(Integer id, String label, String group, Point
        point, Resource resource) {
29        this.id = id;
30        this.label = label;
31        this.group = group;
32        this.point = point;
33        this.resource = resource;
```

```
34     }
35     public Node(Integer id, String label, String group,
36         String address, List<String> ports, Point point,
37         Resource resource) {
38         this.id = id;
39         this.label = label;
40         this.group = group;
41         this.address = address;
42         this.ports = ports;
43         this.point = point;
44         this.resource = resource;
45     }
46     public Integer getId() {
47         return id;
48     }
49     public String getName() {
50         return name;
51     }
52     public String getGroup() {
53         return group;
54     }
55     public void setGroup(String group) {
56         this.group = group;
57     }
58     public double getRange() {
59         return range;
60     }
61     public String getAddress() {
62         return address;
63     }
64     public List<String> getPorts(String port) {
65         return ports;
66     }
67     public Node getSrc() {
68         return src;
```

```
68     }
69     public Node getDst() {
70         return dst;
71     }
72     public static class Point {
73         public Node node1, node;
74         public Double distance;
75         public double x, y;
76         public double range;
77         public Point point;
78         public Point() {
79             }
80         public Point(double x, double y) {
81             this.x = x;
82             this.y = y;
83         }
84         public Point(Node node1, Node node2, Double distance) {
85             double a = (node1.point.getY() - node2.point.getY())/
86                 (node1.point.getX() - node2.point.getX());
87             this.x = node1.point.getX() + distance/
88                 (Math.sqrt(1+Math.pow(a,2)));
89             this.y = a*(distance/(Math.sqrt(1+Math.pow(a,2))))+
90                 node1.point.getY();
91         }
92         public Point(Node node1, Node node2) {
93             this.x = (node1.point.getX() + node2.point.getX())/2;
94             this.y = (node1.point.getY() + node2.point.getY())/2;
95         }
96         public Point(Node node1, double range) {
97             this.x = node1.point.getX() + range;
98             this.y = node1.point.getY();
99         }
100         public double getX() {
101             return x;
102         }
103         public double getY() {
```

```
103     return y;
104 }
105 public Point getPoint() {
106     return point;
107 }
108 public void setPoint(Point point) {
109     this.point = point;
110 }
111 }
112 public static class Resource{
113     public int cpu, mem, storage;
114     public Resource() {
115     }
116     public Resource(int cpu, int mem, int storage) {
117         this.cpu = cpu;
118         this.mem = mem;
119         this.storage = storage;
120     }
121     public int getCpu() {
122         return cpu;
123     }
124     public int getMem() {
125         return mem;
126     }
127     public int getStorage() {
128         return storage;
129     }
130 }
131 }
```

Link.java

```
1 import java.util.Objects;
2 import org.kie.api.definition.type.Position;
3 public class Link{
4     @Position(0)
```

```
5  public Node src;
6  @Position(1)
7  public Node dst;
8  public Integer id;
9  public int weight = 0;
10 public String srcId, dstId;
11 public String srcPort, dstPort;
12 public Double distance;
13 public Link(){
14 }
15 public Link(Node src, Node dst) {
16     this.src = src;
17     this.dst = dst;
18 }
19 public Link(Node src, Node dst, int weight) {
20     this.src = src;
21     this.dst = dst;
22     this.weight=weight;
23 }
24 public Link(String srcId, String dstId) {
25     this.srcId = srcId;
26     this.dstId = dstId;
27 }
28 public Link(String id,String srcId, String dstId) {
29     this.dstId = id;
30     this.srcId = srcId;
31     this.dstId = dstId;
32 }
33 public Link(String srcId, String srcPort, String dstId,
34     String dstPort) {
35     this.srcId = srcId;
36     this.srcPort = srcPort;
37     this.dstId = dstId;
38     this.dstPort = dstPort;
39 }
39 public Distance(Node node1, Node node2) {
```

```
40     this.node1 =node1;
41     this.node2 =node2;
42     this.distance = Math.sqrt(Math.pow(node1.point
43     .getX()-node2.point.getX(),2)
44     + Math.pow(node1.point.getY()-node2.point.getY(),2));
45 }
46 public Node getSrc() {
47     return src;
48 }
49 public Node getDst() {
50     return dst;
51 }
52 public int getWeight() {
53     return weight;
54 }
55 public Double getDistance() {
56     return distance;
57 }
58 public String getSrcId() {
59     return srcId;
60 }
61 public String getSrcPort() {
62     return srcPort;
63 }
64 public String getDstId() {
65     return dstId;
66 }
67 public void setWeight(int weight) {
68     this.weight = weight;
69 }
70 public String getDstPort() {
71     return dstPort;
72 }
73 }
```

Path.java

```
1 import java.util.ArrayList;
2 import java.util.LinkedList;
3 import org.kie.api.definition.type.Position;
4 public class Path {
5     @Position(0)
6     public Node src;
7     @Position(1)
8     public Node dst;
9     public Node node;
10    public Path path;
11    public int length;
12    public ArrayList<Node> nodes = new ArrayList<Node>();
13    public LinkedList<Link> links = new LinkedList<Link>();
14    public Path() {
15    }
16    public Path(ArrayList<Node> nodes) {
17        this.nodes = nodes;
18        this.length= nodes.size()-1;
19    }
20    public Path(LinkedList<Link> links) {
21        this.links = links;
22        this.length= links.size();
23    }
24    public void addNode(Node node) {
25        getNodes().add(node);
26    }
27    public void addLink(Link link) {
28        getLinks().add(link);
29    }
30    public void addPath(Path path) {
31        for(int i=0; i<=path.getLinks().size()-1; i++){
32            addLink(path.getLinks().get(i));
33        }
34    }
35    public void addLinks(LinkedList<Link> links) {
```

```
36     for(int i=0; i<=links.size()-1; i++){
37         addLink(links.get(i));
38     }
39 }
40 public ArrayList<Node> getNodes() {
41     for(int i=0; i<links.size()-1; i++){
42         nodes.add(links.get(i).src);
43     }
44     nodes.add(links.getLast().src);
45     return nodes;
46 }
47 public LinkedList<Link> getLinks() {
48     return links;
49 }
50 public Node getSrc() {
51     if (getLinks().size()>0) {
52         src= getLinks().get(0).src;
53     }
54     else {
55         src=null;
56     }
57     return src;
58 }
59 public Node getNode() {
60     return node;
61 }
62 public Node getDst() {
63     if (getLinks().size()>0) {
64         dst = getLinks().get(links.size()-1).dst;
65     }
66     else {
67         dst=null;
68     }
69     return dst;
70 }
71 public double getWeight() {
```



```
72     double weight = 0;
73     for(int i=0; i<links.size()-1; i++){
74         weight = weight + getLinks().get(i).getWeight();
75     }
76     return weight;
77 }
78 public Double getDistance() {
79     double distance = 0;
80     for(int i=0; i<links.size()-1; i++){
81         distance = distance + getLinks().get(i).getDistance();
82     }
83     return distance;
84 }
85 public int getLength() {
86     length = links.size();
87     return length;
88 }
89 }
```

Topo.java

```
1 import java.lang.Math;
2 import java.util.HashSet;
3
4 import org.kie.api.definition.type.Position;
5 import com.sample.Node;
6
7 public class Topo {
8     @Position(0)
9     public Node node1;
10    @Position(1)
11    public Node node2;
12    @Position(3)
13    public Node node3=null;
14    @Position(4)
15    public Node node4=null;
```

```
16 public Node src, dst;
17 public Double distance, diagonal;
18 public Constraint constraint;
19 public Topo topo1, topo2, topo3, topo4;
20 public Topo() {}
21 public static class Line extends Topo {
22     public Line() {
23     };
24     public Line(Topo topo1, Topo topo2) {
25         this.topo1 =topo1;
26         this.topo2=topo2;
27     }
28     public Line(Node node1, Constraint constraint) {
29         this.node1 =node1;
30         this.constraint=constraint;
31     }
32     public Line(Node node1, Node node2, Constraint
33         constraint) {
34         this.node1 =node1;
35         this.constraint=constraint;
36         this.distance = Math.sqrt(Math.pow(node1.point
37             .getX()-node2.point.getX(),2)
38             + Math.pow(node1.point.getY()-node2.point.getY(),2));
39     }
40     public Line(Node node1, Node node2) {
41         this.node1 =node1;
42         this.node2 =node2;
43         this.distance = Math.sqrt(Math.pow(node1.point
44             .getX()-node2.point.getX(),2)
45             + Math.pow(node1.point.getY()-node2.point.getY(),2));
46     }
47     public void setLine(Node node1, Node node2) {
48         this.node1 = node1;
49         this.node2 = node2;
50         this.distance = Math.sqrt(Math.pow(node1.point
51             .getX()-node2.point.getX(),2)
```

```
51         + Math.pow(node1.point.getY()-node2.point.getY(),2));
52     }
53 }
54 public static class Tree extends Topo{
55     public HashSet<Node> nodes;
56     public HashSet<Node> leafs;
57     public Tree() {
58     }
59     public Tree(Node node1, HashSet<Node> nodes) {
60         this.node1 = node1;
61         this.nodes = nodes;
62     }
63     public Tree(Node node1) {
64         this.node1 = node1;
65     }
66     public Tree(Node node1, Constraint constraint) {
67         this.node1 = node1;
68         this.constraint = constraint;
69         //this.distance = null;
70     }
71     public Tree(Node node1, Node node2, Node node3) {
72         this.node1 = node1;
73         this.node2 = node2;
74         this.node3 = node3;
75         this.distance = Math.sqrt(Math.pow(node1.point
76             .getX()-node2.point.getX(),2)
77             + Math.pow(node1.point.getY()-node2.point.getY(),2));
78     }
79     public Tree(Node node1, Node node2, Node node3, Node
80         node4) {
81         this.node1 = node1;
82         this.node2 = node2;
83         this.node3 = node3;
84         this.node4 = node4;
85         this.distance = Math.sqrt(Math.pow(node1.point
            .getX()-node2.point.getX(),2)
```

```
86         + Math.pow(node1.point.getY()-node2.point.getY(),2));
87     }
88     public void setTree(Node node1, Node node2, Node node3)
89     {
90         this.node1 = node1;
91         this.node2 = node2;
92         this.node3 = node3;
93         this.distance = Math.sqrt(Math.pow(node1.point
94             .getX()-node2.point.getX(),2)
95             + Math.pow(node1.point.getY()-node2.point.getY(),2));
96     }
97     public void setTree(Node node1, Node node2, Node node3,
98         Node node4) {
99         this.node1 = node1;
100        this.node2 = node2;
101        this.node3 = node3;
102        this.node4 = node4;
103        this.distance = Math.sqrt(Math.pow(node1.point.
104            getX()-node2.point.getX(),2)
105            + Math.pow(node1.point.getY()-node2.point.getY(),2));
106    }
107    }
108    public static class Mesh extends Topo{
109        public Mesh() {
110        }
111        public Mesh(Node node1, Node node2, Node node3, Node
112            node4) {
113            this.node1 = node1;
114            this.node2 = node2;
115            this.node3 = node3;
116            this.node4 = node4;
117            this.distance = Math.sqrt(Math.pow(node1.point
118                .getX()-node2.point.getX(),2)
119                + Math.pow(node1.point.getY()-node2.point.getY(),2));
120            this.diagonal = Math.sqrt(Math.pow(node1.point
121                .getX()-node3.point.getX(),2)
```

```
119         + Math.pow(node1.point.getY()-node3.point.getY(),2));
120     }
121     public void setMesh(Node node1, Node node2, Node node3,
122         Node node4) {
123         this.node1 = node1;
124         this.node2 = node2;
125         this.node3 = node3;
126         this.node4 = node4;
127         this.distance = Math.sqrt(Math.pow(node1.point
128             .getX()-node2.point.getX(),2)
129             + Math.pow(node1.point.getY()-node2.point.getY(),2));
130         this.diagonal = Math.sqrt(Math.pow(node1.point
131             .getX()-node3.point.getX(),2)
132             + Math.pow(node1.point.getY()-node3.point.getY(),2));
133     }
134     public static class Sequence extends Topo{
135         public Sequence() {
136         }
137         public Sequence(Node node1, Node node2) {
138             this.node1 = node1;
139             this.node2 = node2;
140             this.distance = Math.sqrt(Math.pow(node1.point
141                 .getX()-node2.point.getX(),2)+
142                 Math.pow(node1.point.getY()-node2.point.getY(),2));
143         }
144         public Sequence(Topo topo1, Node node2) {
145             this.topo1 = topo1;
146             this.node2 = node2;
147         }
148         public Sequence(Node node1, Topo topo2) {
149             this.node1 = node1;
150             this.topo2 = topo2;
151             this.distance = Math.sqrt(Math.pow(node1.point
152                 .getX()-node2.point.getX(),2)
153                 + Math.pow(node1.point.getY()-node2.point.getY(),2));
```

```
153     }
154     public Sequence(Topo topo1, Topo topo2) {
155         this.topo1 = topo1;
156         this.topo2 = topo2;
157         this.distance = Math.sqrt(Math.pow(node1.point
158             .getX()-node2.point.getX(),2)
159             + Math.pow(node1.point.getY()-node2.point.getY(),2));
160     }
161     public void setMesh(Node node1, Node node2) {
162         this.node1 = node1;
163         this.node2 = node2;
164         this.distance = Math.sqrt(Math.pow(node1.point
165             .getX()-node2.point.getX(),2)
166             + Math.pow(node1.point.getY()-node2.point.getY(),2));
167     }
168 }
169 public void setNode1(Node node1) {
170     this.node1 = node1;
171 }
172 public void setNode2(Node node2) {
173     this.node2 = node2;
174 }
175 public void setNode3(Node node3) {
176     this.node3 = node3;
177 }
178 public void setNode4(Node node4) {
179     this.node4 = node4;
180 }
181 public Node getNode1() {
182     return node1;
183 }
184 public Node getNode2() {
185     return node2;
186 }
187 public Node getNode3() {
188     return node3;
```

```
189     }
190     public Node getNode4() {
191         return node4;
192     }
193     public Constraint getConstraint() {
194         return constraint;
195     }
196     public Double getDistance() {
197         return distance;
198     }
199
200     public void setDistance(Double distance) {
201         this.distance = distance;
202     }
203
204     public void setDiagonal(Double diagonal) {
205         this.diagonal = diagonal;
206     }
207
208     public Double getDiagonal() {
209         return diagonal;
210     }
211 }
```

Pro.java

```
1 public class Pro {
2     public String name;
3     public double value;
4     public Area area;
5     public boolean encrypted;
6     public boolean reached;
7     public Chain chain;
8     public Pro() {
9     }
10    // Functional Properties
```

```
11 public static class Connectivity extends Pro{
12     public Connectivity() {}
13     public Connectivity(Double value) {
14         this.name = "Connectivity";
15         this.value=value;
16     }
17 }
18 public static class Coverage extends Pro{
19     public Coverage() {}
20     public Coverage(double value) {
21         this.name = "Coverage";
22         this.value=value;
23     }
24 }
25 public static class Scalability extends Pro{
26     public Scalability() {}
27     public Scalability(double value) {
28         this.name = "Scalability";
29         this.value = value;
30     }
31 }
32 public static class Reachability extends Pro{
33     public Reachability() {}
34     public Reachability(boolean reached) {
35         this.name = "Reachability";
36         this.reached = reached;
37     }
38 }
39 // Non Functional Properties
40 public static class FunctionChain extends Pro{
41     public FunctionChain() {}
42     public FunctionChain(Chain chain) {
43         this.name = "Chaining";
44         this.chain = chain;
45     }
46     public FunctionChainPath(Chain chain) {
```



```
47     this.name = "Chain Path";
48     this.chain = chain;
49 }
50 }
51 public static class Confidentiality extends Pro{
52     public Confidentiality() {}
53     public Confidentiality(boolean encrypted) {
54         this.name = "Confidentiality";
55         this.encrypted = encrypted;
56     }
57 }
58 public static class Availability extends Pro{
59     public Availability() {}
60     public Availability(double value) {
61         this.name = "Availability";
62         this.value = value;
63     }
64 }
65 public static class Area{
66     public double length, width;
67     public Area(double length, double width) {
68         this.length = length;
69         this.width = width;
70     }
71 }
72 public String getName() {
73     return name;
74 }
75 public Area getArea() {
76     return area;
77 }
78 public double getValue() {
79     return value;
80 }
81 public Chain getChain() {
82     return chain;
```

```
83     }
84     public boolean isReached() {
85         return reached;
86     }
87     public boolean isEncrypted() {
88         return encrypted;
89     }
90     public void setValue(double value) {
91         this.value = value;
92     }
93 }
```

Req.java

```
1 import java.util.HashSet;
2 import java.util.Set;
3 import org.kie.api.definition.type.Position;
4 import com.sample.Node;
5 public class Req {
6     @Position(0)
7     public Node src;
8     @Position(1)
9     public Node dst;
10    public Node node1, node2;
11    public Topo topo;
12    public Path path;
13    public Pro property;
14    public Constraint constraint;
15    public boolean satisfied;
16    public Req req;
17    public Set<Req> inferredReqs = new HashSet<Req>();
18    public Req() {
19    }
20    public Req(Node src, Node dst, Pro property, boolean
        satisfied) {
21        this.src = src;
```

```
22     this.dst = dst;
23     this.property = property;
24     this.satisfied = satisfied;
25 }
26 public Req(Node src, Node dst, Pro property, Constraint
    constraint, boolean satisfied) {
27     this.src = src;
28     this.dst = dst;
29     this.property = property;
30     this.satisfied = satisfied;
31     this.constraint = constraint;
32 }
33 public Req(Topo topo, Pro property, boolean satisfied) {
34     this.topo = topo;
35     this.src = topo.node1;
36     this.dst = topo.node2;
37     this.property = property;
38     this.satisfied = satisfied;
39 }
40 public Req(Topo topo, Pro property, Constraint
    constraint, boolean satisfied) {
41     this.topo = topo;
42     this.src = topo.node1;
43     this.dst = topo.node2;
44     this.property = property;
45     this.constraint = constraint;
46     this.satisfied = satisfied;
47 }
48 public Req(Path path, Pro property, boolean satisfied) {
49     this.path = path;
50     this.property = property;
51     this.satisfied = satisfied;
52 }
53 public Node getSrc() {
54     return src;
55 }
```

```
56     public Node getDst() {
57         return dst;
58     }
59     public void setSrc(Node src) {
60         this.src = src;
61     }
62     public void setDst(Node dst) {
63         this.dst = dst;
64     }
65     public Req getReq(){
66         return req;
67     }
68     public Topo getTopo() {
69         return topo;
70     }
71     public Pro getProperty() {
72         return property;
73     }
74     public Constraint getConstraint() {
75         return constraint;
76     }
77     public boolean isSatisfied() {
78         return satisfied;
79     }
80     public Node getNode1() {
81         return node1;
82     }
83     public Node getNode2() {
84         return node2;
85     }
86     public Path getPath() {
87         return path;
88     }
89     public Set<Req> getInferredReqs() {
90         return inferredReqs;
91     }
```

```
92     public void setNode1(Node node1) {
93         this.node1 = node1;
94     }
95     public void setNode2(Node node2) {
96         this.node2 = node2;
97     }
98     public void setTopo(Topo topo) {
99         this.topo = topo;
100    }
101    public void setPath(Path path) {
102        this.path = path;
103    }
104    public void setProperty(Pro property) {
105        this.property = property;
106    }
107    public void setConstraint(Constraint constraint) {
108        this.constraint = constraint;
109    }
110    public void setSatisfied(boolean satisfied) {
111        this.satisfied = satisfied;
112    }
113    public void setReq(Req req) {
114        this.req = req;
115    }
116    public void setInferredReqs(Set<Req> inferredReqs) {
117        this.inferredReqs = inferredReqs;
118    }
119 }
```

Flow.java

```
1 import java.util.ArrayList;
2 import java.util.List;
3 public class Flow {
4     public int id;
5     public String switchId, flowId, inPort, output, tableId;
```

```
6     public List<String> outputs = new ArrayList<String>();
7     public int table, priority;
8     public Flow() {
9     }
10    public Flow(String switchId, String inPort, String
        output) {
11        this.switchId = switchId;
12        this.inPort = inPort;
13        this.output = output;
14    }
15    public Flow(int id, String switchId, String inPort,
        String output) {
16        this.id = id;
17        this.switchId = switchId;
18        this.inPort = inPort;
19        this.output = output;
20    }
21    public Flow(int id, String switchId, String flowId,
        String inPort, String output) {
22        this.id = id;
23        this.switchId = switchId;
24        this.flowId = flowId;
25        this.inPort = inPort;
26        this.output = output;
27    }
28    public Flow(int id, String switchId, String tableId,
        String flowId,
29        String inPort, List<String> outputs) {
30        this.id = id;
31        this.switchId = switchId;
32        this.tableId = tableId;
33        this.flowId = flowId;
34        this.inPort = inPort;
35        this.outputs = outputs;
36    }
37    public int getId() {
```

```
38     return id;
39 }
40 public String getSwitchId() {
41     return switchId;
42 }
43 public String getTableId() {
44     return tableId;
45 }
46 public String getFlowId() {
47     return flowId;
48 }
49 public String getInPort() {
50     return inPort;
51 }
52 public int getPriority() {
53     return priority;
54 }
55 public List<String> getOutputs() {
56     return outputs;
57 }
58 }
```

Chain.java

```
1 import java.util.ArrayList;
2 public class Chain {
3     public Node sf1;
4     public ArrayList<Node> functions = new
        ArrayList<Node>();
5     public Chain() {
6     }
7     public Chain(Node sf1) {
8         getFunctions().add(sf1);
9     }
10    public Chain(Node sf1, Node sf2) {
11        getFunctions().add(sf1);
```

```
12     getFunctions().add(sf2);
13 }
14 public Chain(Node sf1,Node sf2,Node sf3) {
15     getFunctions().add(sf1);
16     getFunctions().add(sf2);
17     getFunctions().add(sf3);
18 }
19 public Chain(ArrayList<Node> functions) {
20     this.functions = functions;
21 }
22 public ArrayList<Node> getFunctions() {
23     return functions;
24 }
25 public Node getFirst() {
26     return getFunctions().get(0);
27 }
28 public Node removeFirst() {
29     return getFunctions().remove(0);
30 }
31 }
```

SecAs.java

```
1 public class SecAs {
2     public Node src, dst;
3     public IPsec ipsec;
4     public SecAs(Node src, Node dst, IPsec ipsec) {
5         this.src = src;
6         this.dst=dst;
7         this.ipsec = ipsec;
8         this.key = key;
9     }
10    public static class IPsec{
11        public String key;
12        public Proto protocol;
13        public Algo algorithm;
```



```
14     public IPSec(Proto protocol, Algo algorithm, String
        key) {
15         this.protocol = protocol;
16         this.algorithm = algorithm;
17         this.key = key;
18     }
19 }
```

Encrypt.java

```
1 import java.util.Arrays;
2 import java.util.Base64;
3 import javax.crypto.Cipher;
4 import javax.crypto.spec.SecretKeySpec;
5 public class Encrypt {
6     public String packet;
7     public String secret
8     public Encrypt(String packet, String secret) {
9         setKey(secret);
10        Cipher cipher =
11            Cipher.getInstance("AES/ECB/PKCS5Padding");
12        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
13        return
14            Base64.getEncoder().encodeToString(cipher.
15                doFinal(packet.getBytes("UTF-8")));
16    }
17 }
```

Decrypt.java

```
1 import java.util.Arrays;
2 import java.util.Base64;
3 import javax.crypto.Cipher;
4 import javax.crypto.spec.SecretKeySpec;
5 public class Decrypt {
```

```
6      public String cipherPacket;  
7      public String secret  
8      public Decrypt (String cipher, String secret){  
9          setKey(secret);  
10         Cipher cipher =  
11             Cipher.getInstance("AES/ECB/PKCS5PADDING");  
12         cipher.init(Cipher.DECRYPT_MODE, secretKey);  
13         return new  
14             String(cipher.doFinal(Base64.getDecoder().  
                decode(cipherPacket)));  
15     }  
16 }
```

Appendix B

Service Function Chaining Files

Service Function Chaining Output

service-nodes.json

```
1 {
2   "service-nodes": {
3     "service-node": [
4       {
5         "name": "sf4",
6         "service-function": [
7           "scada-ids-1"
8         ],
9         "ip-mgmt-address": "192.168.10.40"
10      },
11      {
12        "name": "sf3",
13        "service-function": [
14          "ids-1"
15        ],
16        "ip-mgmt-address": "192.168.10.30"
17      },
18      {
19        "name": "sf2",
20        "service-function": [
21          "dpi-1"
```

```
22     ],
23     "ip-mgmt-address": "192.168.10.20"
24 },
25 {
26     "name": "sf1",
27     "service-function": [
28         "firewall-1"
29     ],
30     "ip-mgmt-address": "192.168.10.10"
31 },
32 {
33     "name": "sff3",
34     "ip-mgmt-address": "192.168.10.90"
35 },
36 {
37     "name": "sff2",
38     "ip-mgmt-address": "192.168.10.80"
39 },
40 {
41     "name": "sff1",
42     "ip-mgmt-address": "192.168.10.70"
43 },
44 {
45     "name": "classifier2",
46     "ip-mgmt-address": "192.168.10.60"
47 },
48 {
49     "name": "classifier1",
50     "ip-mgmt-address": "192.168.10.50"
51 }
52 ]
53 }
54 };
```

service-functions.json

```
1 {
2   "service-functions": {
3     "service-function": [
4       {
5         "name": "firewall-1",
6         "nsh-aware": true,
7         "ip-mgmt-address": "192.168.10.10",
8         "type": "firewall",
9         "sf-data-plane-locator": [
10          {
11            "name": "firewal-1-dpl",
12            "service-function-forwarder": "SFF1",
13            "transport": "service-locator:vxlan-gpe",
14            "ip": "192.168.10.10",
15            "port": 6633
16          }
17        ],
18        "rest-uri": "http://192.168.10.10:5000"
19      },
20      {
21        "name": "dpi-1",
22        "nsh-aware": true,
23        "ip-mgmt-address": "192.168.10.20",
24        "type": "dpi",
25        "sf-data-plane-locator": [
26          {
27            "name": "dpi-1-dpl",
28            "service-function-forwarder": "SFF2",
29            "transport": "service-locator:vxlan-gpe",
30            "ip": "192.168.10.20",
31            "port": 6633
32          }
33        ],
34        "rest-uri": "http://192.168.10.20:5000"
35      },
36      {
```

```
37     "name": "scada-ids-1",
38     "nsh-aware": true,
39     "ip-mgmt-address": "192.168.10.40",
40     "type": "qos",
41     "sf-data-plane-locator": [
42         {
43             "name": "scada-ids-1-dpl",
44             "service-function-forwarder": "SFF3",
45             "transport": "service-locator:vxlan-gpe",
46             "ip": "192.168.10.40",
47             "port": 6633
48         }
49     ],
50     "rest-uri": "http://192.168.10.40:5000"
51 },
52 {
53     "name": "ids-1",
54     "nsh-aware": true,
55     "ip-mgmt-address": "192.168.10.30",
56     "type": "ids",
57     "sf-data-plane-locator": [
58         {
59             "name": "ids-1-dpl",
60             "service-function-forwarder": "SFF3",
61             "transport": "service-locator:vxlan-gpe",
62             "ip": "192.168.10.30",
63             "port": 6633
64         }
65     ],
66     "rest-uri": "http://192.168.10.30:5000"
67 }
68 ]
69 }
70 };
```

service-function-classifiers.json

```
1
2 {
3   "service-function-classifiers": {
4     "service-function-classifier": [
5       {
6         "name": "Classifier1",
7         "scl-service-function-forwarder": [
8           {
9             "name": "Classifier1",
10            "interface": "veth-br"
11          }
12        ],
13        "acl": {
14          "name": "ACL1",
15          "type": "ietf-access-control-list:ipv4-acl"
16        }
17      },
18      {
19        "name": "Classifier2",
20        "scl-service-function-forwarder": [
21          {
22            "name": "Classifier2",
23            "interface": "veth-br"
24          }
25        ],
26        "acl": {
27          "name": "ACL2",
28          "type": "ietf-access-control-list:ipv4-acl"
29        }
30      }
31    ]
32  }
33 }
```

service-function-forwarder.json

```
1
2 {
3   "service-function-forwarders": {
4     "service-function-forwarder": [
5       {
6         "name": "Classifier1",
7         "ip-mgmt-address": "192.168.10.50",
8         "sff-data-plane-locator": [
9           {
10            "name": "sff0-dpl",
11            "data-plane-locator": {
12              "transport": "service-locator:vxlan-gpe",
13              "port": 6633,
14              "ip": "192.168.10.50"
15            },
16            "service-function-forwarder-ovs:ovs-options": {
17              "remote-ip": "flow",
18              "key": "flow",
19              "nshc3": "flow",
20              "nshc4": "flow",
21              "nshc1": "flow",
22              "nshc2": "flow",
23              "nsi": "flow",
24              "exts": "gpe",
25              "dst-port": "6633",
26              "nsp": "flow"
27            }
28          }
29        ],
30        "service-node": "classifier1",
31        "service-function-forwarder-ovs:ovs-bridge": {
32          "bridge-name": "br-sfc"
33        },
34        "rest-uri": "http://192.168.10.50:5000"
35      },
```



```

36     {
37         "name": "SFF1",
38         "ip-mgmt-address": "192.168.10.70",
39         "service-node": "sff1",
40         "sff-data-plane-locator": [
41             {
42                 "name": "sff1-dpl",
43                 "data-plane-locator": {
44                     "transport": "service-locator:vxlan-gpe",
45                     "port": 6633,
46                     "ip": "192.168.10.70"
47                 },
48                 "service-function-forwarder-ovs:ovs-options": {
49                     "remote-ip": "flow",
50                     "key": "flow",
51                     "nshc3": "flow",
52                     "nshc4": "flow",
53                     "nshc1": "flow",
54                     "nshc2": "flow",
55                     "nsi": "flow",
56                     "exts": "gpe",
57                     "dst-port": "6633",
58                     "nsp": "flow"
59                 }
60             }
61         ],
62         "service-function-dictionary": [
63             {
64                 "name": "firewall-1",
65                 "sff-sf-data-plane-locator": {
66                     "sff-dpl-name": "sff1-dpl",
67                     "sf-dpl-name": "firewall-1-dpl"
68                 }
69             }
70         ],
71         "service-function-forwarder-ovs:ovs-bridge": {

```

```
72     "bridge-name": "br-sfc"
73   },
74   "rest-uri": "http://192.168.10.70:5000"
75 },
76 {
77   "name": "SFF3",
78   "ip-mgmt-address": "192.168.10.90",
79   "service-node": "sff3",
80   "sff-data-plane-locator": [
81     {
82       "name": "sff3-dpl",
83       "data-plane-locator": {
84         "transport": "service-locator:vxlan-gpe",
85         "port": 6633,
86         "ip": "192.168.10.90"
87       },
88       "service-function-forwarder-ovs:ovs-options": {
89         "remote-ip": "flow",
90         "key": "flow",
91         "nshc3": "flow",
92         "nshc4": "flow",
93         "nshc1": "flow",
94         "nshc2": "flow",
95         "nsi": "flow",
96         "exts": "gpe",
97         "dst-port": "6633",
98         "nsp": "flow"
99       }
100     }
101   ],
102   "service-function-dictionary": [
103     {
104       "name": "ids-1",
105       "sff-sf-data-plane-locator": {
106         "sff-dpl-name": "sff3-dpl",
107         "sf-dpl-name": "ids-1-dpl"
```

```

108         }
109     },
110     {
111         "name": "scada-ids-1",
112         "sff-sf-data-plane-locator": {
113             "sff-dpl-name": "sff3-dpl",
114             "sf-dpl-name": "scada-ids-1-dpl"
115         }
116     }
117 ],
118 "service-function-forwarder-ovs:ovs-bridge": {
119     "bridge-name": "br-sfc"
120 },
121 "rest-uri": "http://192.168.10.90:5000"
122 },
123 {
124     "name": "Classifier2",
125     "ip-mgmt-address": "192.168.10.60",
126     "sff-data-plane-locator": [
127         {
128             "name": "sff4-dpl",
129             "data-plane-locator": {
130                 "transport": "service-locator:vxlan-gpe",
131                 "port": 6633,
132                 "ip": "192.168.10.60"
133             },
134             "service-function-forwarder-ovs:ovs-options": {
135                 "remote-ip": "flow",
136                 "key": "flow",
137                 "nshc3": "flow",
138                 "nshc4": "flow",
139                 "nshc1": "flow",
140                 "nshc2": "flow",
141                 "nsi": "flow",
142                 "exts": "gpe",
143                 "dst-port": "6633",

```

```
144         "nsp": "flow"
145     }
146 }
147 ],
148 "service-node": "classifier2",
149 "service-function-forwarder-ovs:ovs-bridge": {
150     "bridge-name": "br-sfc"
151 },
152 "rest-uri": "http://192.168.10.60:5000"
153 },
154 {
155     "name": "SFF2",
156     "ip-mgmt-address": "192.168.10.80",
157     "service-node": "sff2",
158     "sff-data-plane-locator": [
159         {
160             "name": "sff2-dpl",
161             "data-plane-locator": {
162                 "transport": "service-locator:vxlan-gpe",
163                 "port": 6633,
164                 "ip": "192.168.10.80"
165             },
166             "service-function-forwarder-ovs:ovs-options": {
167                 "remote-ip": "flow",
168                 "key": "flow",
169                 "nshc3": "flow",
170                 "nshc4": "flow",
171                 "nshc1": "flow",
172                 "nshc2": "flow",
173                 "nsi": "flow",
174                 "exts": "gpe",
175                 "dst-port": "6633",
176                 "nsp": "flow"
177             }
178         }
179     ],
```

```

180     "service-function-dictionary": [
181     {
182         "name": "dpi-1",
183         "sff-sf-data-plane-locator": {
184             "sff-dpl-name": "sff2-dpl",
185             "sf-dpl-name": "dpi-1-dpl"
186         }
187     }
188 ],
189 "service-function-forwarder-ovs:ovs-bridge": {
190     "bridge-name": "br-sfc"
191 },
192 "rest-uri": "http://192.168.10.80:5000"
193 }
194 ]
195 }
196 };

```

service-function-chains.json

```

1 {
2     "service-function-chains": {
3         "service-function-chain": [
4             {
5                 "name": "SFC4",
6                 "symmetric": true,
7                 "sfc-service-function": [
8                     {
9                         "name": "firewall-abstract1",
10                        "type": "firewall"
11                    },
12                    {
13                        "name": "dpi-abstract1",
14                        "type": "dpi"
15                    }
16                ]
17            }
18        ]
19    }
20 }

```

```
17     },
18     {
19         "name": "SFC3",
20         "symmetric": true,
21         "sfc-service-function": [
22             {
23                 "name": "firewall-abstract1",
24                 "type": "firewall"
25             },
26             {
27                 "name": "ids-abstract1",
28                 "type": "ids"
29             }
30         ]
31     },
32     {
33         "name": "SFC2",
34         "symmetric": true,
35         "sfc-service-function": [
36             {
37                 "name": "firewall-abstract1",
38                 "type": "firewall"
39             },
40             {
41                 "name": "scada-ids-abstract1",
42                 "type": "qos"
43             }
44         ]
45     },
46     {
47         "name": "SFC1",
48         "symmetric": true,
49         "sfc-service-function": [
50             {
51                 "name": "firewall-abstract1",
52                 "type": "firewall"
```

```
53         },
54         {
55             "name": "ids-abstract1",
56             "type": "ids"
57         }
58     ]
59 },
60 {
61     "name": "grgre"
62 }
63 ]
64 }
65 };
```