# Patterns for the Design of Secure and Dependable SDN

Nikolaos E. Petroulakis[a,b,*], George Spanoudakis[b], Ioannis G. Askoxylakis[a]

[a] *Institute of Computer Science, Foundation for Research and Technology-Hellas, Heraklion, Greece*
[b] *Department of Computer Science, City University London, London, UK*

## Abstract

In an interconnected world, cyber and physical networks face a number of challenges that need to be resolved. These challenges are mainly due to the nature and complexity of interconnected systems and networks and their ability to support heterogeneous physical and cyber components simultaneously. The construction of complex networks preserving security and dependability (S&D) properties is necessary to avoid system vulnerabilities, which may occur in all the different layers of Software Defined Networks (SDN) architectures. In this paper, we present a model based approach to support the design of secure and dependable SDN. This approach is based on executable patterns for designing networks able to guarantee S&D properties and can be used in SDN networks. The 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. To evaluate our pattern approach, we have implemented executable pattern instances, in a rule-based reasoning system, and used them to design and verify wireless SDN networks with respect to availability and confidentiality. To complete this work, we propose and evaluate an implementation framework in which S&D patterns can be applied for the design and verification of SDN networks.

*Keywords:* Design Patterns, SDN, Wireless Networks, Security, Dependability, Drools

[*] Corresponding author
*Email address:* `npetro@ics.forth.gr` (Nikolaos E. Petroulakis )

## 1. Introduction

The design of complex system networks is of paramount importance due to their increasing role in the implementation of Cyber-Physical Systems (CPS) and Software Defined Networks (SDN) involving integrated ICT and physical components and devices. However, the design of such networks adequate encounters difficulties which need to be resolved. These difficulties 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. The design and verification methods for developing secure and dependable system networks is necessary and should be considered at design level to guarantee security and mitigate safety threats, on remote monitored and managed networks. Especially, with the fast growing of SDN and the integration with 5G network architectures [1], the design of networks enters in a new era and makes necessary a careful investigation of the new security and dependability risks, which have not been relevant in legacy systems. One of the challenges of future networks is to develop SDN capabilities tailored to CPS and drive the reconfiguration of these capabilities through network configuration specifications embedded in critical infrastructures.

SDN allow network programmability and control to be decoupled from the forwarding plane and the forwarding plane to be directly programmable by the control plane. 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. In this paper, we give a detailed description of the scheme for specifying S&D patterns and their use for the design of S&D preserving SDN networks. The main contribution of the approach is that encodes designs of network topologies, which are proven to satisfy S&D properties, as design patterns. In addition, S&D patterns can be used for the definition of optimal paths which are able to guarantee S&D properties in deployed networks. A first definition of our pattern-based approach for designing reliable cyber-physical systems was given in [2]. This paper extends the original approach by developing a pattern framework in which we can evaluate and emulate S&D executable patterns on SDN-based network designs. It also presents an application framework in which S&D patterns can insert and modify flow rules through the controller to the programmable switches of SDN infrastructures.

The remainder of this paper is organized as follows. In Section 2 an overview of related work is presented. In Section 3, we present the schema of the pattern execution form. In Section 4, we introduce abstract specification instances of patterns with respect to confidentiality and availability encoded also to a rule-based reasoning language. In Section 5, we propose an implementation framework in which S&D network patterns can be applied in order to design and verify SDN network architectures. In Section 6, we emulate our proposed network patterns for the design of wireless SDN-based network architectures able to provide security against physical layer attacks and failures at design or at runtime in hostile environments. Finally, Section 7 provides conclusions and future work.

## 2. Related Work

The main focus of network design relies on specification analysis, design, verification, and validation of systems that include hardware/software, data, procedures, and facilities. Driven from software development methodology, Model-Driven Engineering (MDE) [3] can be used to analyze certain aspects of models, synthesize various types of artifacts and design secure and dependable systems. An MDE framework for architecting wireless networks is presented in [4]. The design of system is simplified through the modelling of design patterns. MDE applies design patterns [5, 6] as solutions for reusable designs and interactions of objects by the use of formal proven properties[7]. The development of S&D patterns may benefit from the current implementations of software patterns as described in the literature in a variety of works

[8, 9, 10, 11, 12]. The concept of component-based architecture composition is mainly applied on software components and service oriented architecture but it can be used successfully for designing networks [13, 14]. Security workflow patterns, for service compositions based on enabling reasoning engines such as Drools, are also described in [15, 16]. Drools enabling reasoning appeared to be also an efficient rule engine to represent our network workflow patterns. Workflow pattern for QoS aggregation for web service composition have been proposed in [17]. In our approach, executable workflow patterns are used for backward chaining for network compositions.

Especially with the softwarization of networks in SDN, design patterns can be applied in all the different layers of SDN architectures. One of major objectives of SDN is to provide Quality of Services (QoS) and on-demand services [18]. Authors in [19] present an end-to-end orchestration of IoT services using SDN-enabled edge nodes. The construction of network topologies includes also the definition of network and traffic patterns. Traffic engineering and patterns in SDN are presented in [20]. Flow policy patterns as expressed by Frenetic languages, can generate flow rules able to be installed in programmable switches of SDN networks [21]. In our approach, we can provide paths as flow rules based on the security requirements. Design patterns can also be used in northbound interface using RESTful API as proposed in [22]. Our proposed pattern framework is able to interact with the controller using also the RESTful. Furthermore, Service Function Chaining (SFC) [23] aims to provide end-to-end security in SDN following security function compositions. Our approach is able to provide a step forward by creating dynamic security chains following a backward chaining. Finally, the concept of intent-based engineering in SDN appears to enforce security policies [24] as proposed by our S&D patterns.

## 3. S&D Pattern Schema

The design and implementation of SDN infrastructures can be based on an architectural framework where the network elements are integrated through patterns with proven capability to enable the semantic interoperability, and to preserve end-to-end and link-to-link security, privacy, and dependability. S&D patterns can be used as an instrument for designing, verifying and altering the topology of SDN networks, at design time or runtime. At design time, the procedure includes the definition of a design problem and the required S&D property that needs to be guaranteed by the SDN to be designed. In verification, an existing SDN network design (topology) and the required S&D properties are provided, and patterns are applied to analyse the former and establish if the latter are satisfied. The analysis is based on checking if the topology of the pattern matches the network design or some part of it and that 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 patterns are applied to alter the topology and forwarding rules of an operational network in order to ensure the satisfaction of S&D properties. The pattern specification schema is defined as follows:

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

The constituents (a)-(d) of the S&D pattern schema are discussed in more detail below.

### *3.1. Pattern Topology*

S&D patterns 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 an SDN. The compositions defined by S&D patterns can be both vertical and horizontal, i.e., they can involve components at the same (horizontal) or different layers (vertical) layer in the reference architecture of SDN. To do so, S&D 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. In SDN, components can be either hosts, forwarding devices or controllers. Paths may include single step links between two edge nodes or link compositions with at least one intermediate and two edge nodes.

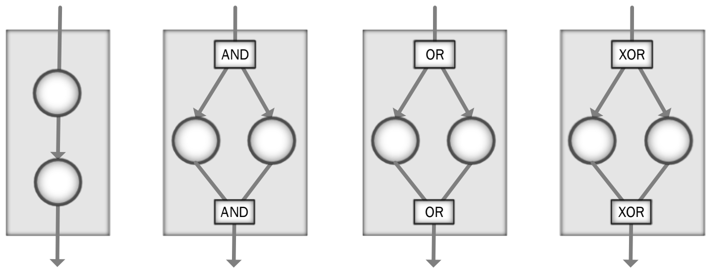

Figure 1: Basic SDN pattern logical topologies: (a) sequence (b) parallel-split-join (c) multi-choice-join (d) exclusive-choice-join

In our S&D patterns so far, we have focused on the logical architecture of the network representing end-to-end connectivity, security and dependability. The basic building blocks for forming logical network topologies are the same as those identified for process workflows in [25]. As it can be seen in Figure 1 for example, the sequence topology depicts the sequential composition of nodes in a network defines that a process is enabled after the completion of a previous one. This topology appears as the fundamental approach for building network process blocks and the diameter/tiers of a network. 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. The parallel-split-join topology (AND-AND) allows the parallel split into two or more branches. This topology is able to provide load-balance in network transmissions. Finally, the exclusive-choice-join topology (XOR-XOR) diverges of 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.

### 3.2. Pattern Constraints

The S&D pattern schema includes also a set of constraints that should be satisfied by the individual network components composed by the pattern and/or the component composition as a whole. These constraints may represent functional requirements regarding such as the connectivity between two components and can be related to the type of components (hardware/software). Different parameters such as the distance between network nodes that is a topological constraint for a network may also be expressed through S&D patterns constraints. For instance, in wired networks this 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. Furthermore, the applications and services that make use of the network are crucial factors on the design of a network as they can affect the available resources such as computational power, available memory, storage and networking capabilities. Other constraints which may be expressed as S&D pattern schema constraints may refer to the quantity and type of nodes, interfaces per nodes, cost and energy consumption.

### 3.3. Pattern S&D Properties

S&D design patterns specify SDN designs that guarantee given security (confidentiality, integrity, availability) and dependability (reliability, safety and maintainability) properties. The satisfiability of an S&D property can be defined by a Boolean value (i.e. encryption enabled/disabled), an arithmetic measure (i.e. delay, encryption level) or probability measure (i.e. reliability/uptime availability). It should be noted that the composition of two components which preserve an S&D property does not necessarily guarantee that the composition will also preserve the same property. 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, in order to guarantee a security property, the property should be satisfied at both the output of the source node and at the input of the destination node.

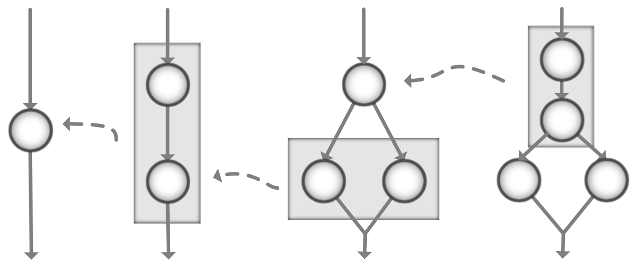

Figure 2: Stepwise Decomposition

S&D patterns can be used to recursively build component compositions or decompositions using forward or backward chaining respectively, as depicted in Figure 2. Forward chaining is useful in verification and backward chaining can be used in inference. In forward chaining, the properties satisfied by components $C_1,..C_n$ may be different from each other and different from the property $Pro$ that the composition satisfies. When $C_1,..,C_n$ are components that satisfy properties $Pro_1,..,Pro_n$ respectively, then the composition $C$ formed of these components can satisfy $Pro$ when the following implication can be proved:

$$\text{When } (C_1 \text{ satisfy } Pro_1) \wedge (C_2 \text{ satisfy } Pro_2) \wedge \cdots \wedge (C_n \text{ satisfy } Pro_n) \rightarrow C \text{ satisfies } Pro$$

On the other hand, backward chaining appears to be more important in system design with respect to a required property. When $C$ is required to satisfy a property $reqPro$, then suitable components $C_1,..,C_n$ should be found to satisfy $reqPro_1,..,reqPro_n$:

$$reqPro(C) \rightarrow reqPro_1(C_1) \wedge \cdots \wedge reqPro_n(C_n)$$

As an example, let's consider a sequential composition of two components: $C \rightarrow C_1 \wedge C_2$. If a required property should be guaranteed by the $C$, the subcomponents $C_1$ and $C_2$ should satisfy the condition $reqPro_1(C_1) \wedge reqPro_2(C_2)$. If there are no atomic components to guarantee the required $reqPro_1(C_1) \wedge reqPro_2(C_2)$, a recursive procedure is used in which successive (sub-) compositions are generated until the atomic components bound to them satisfy the required properties. The decomposition can be analyzed as follows:

$$\left.\begin{array}{l} reqPro_1(C_1) \rightarrow reqPro_{1_1}(C_{1_1}) \wedge reqPro_{1_2}(C_{1_2}) \\ reqPro_2(C_2) \rightarrow reqPro_{2_1}(C_{2_1}) \wedge reqPro_{2_2}(C_{2_2}) \end{array}\right\} \cdots \rightarrow \begin{array}{c} \text{until nodes } C_{1_1}, C_{1_2}, C_{2_1}, C_{2_2} \\ \text{that satisfy } reqPro \text{ are found} \end{array}$$

### *3.4. Pattern Rules*

Once proven, relations between pattern component properties can be expressed as production rules to enable reasoning. In implementing our approach, we have selected Drools [26] to express S&D patterns as rules because this rule engine supports backward and forward chaining inference and verification by implementing and extending the Rete algorithm [27]. Drools rules can encode the topology of a pattern and the process of finding suitable component compositions in order to guarantee the required property. Drools production rules are stored in the production memory and are used to process data inserted in the working memory (Knowledge Base) as facts by pattern matching. Each rule consists of two parts: the *when* condition and the *then* actions. When a network that matches the topology of an S&D pattern does not satisfy the required property, the pattern may be used to substitute, add or remove components from it in order to satisfy the property.

A Drools rule that encodes an S&D pattern includes the inputs of the pattern's components, the type of composition and the required S&D property in Left Hand Side (LHS). When the conditions in the LHS are satisfied, then the rule is fired to execute the actions as described in its Right Hand Side (RHS). In the RHS, the new requirements of the compositions or atomic components can be inserted, updated or deleted.

## 4. S&D Pattern Instances

In this section we present specifications of S&D patterns instances, which are able to guarantee confidentiality and availability in network infrastructures based on the pattern specification approach discussed in Section 3.

### *4.1. Link-to-link Confidentiality Pattern*

Confidential transmission on the infrastructure layer focuses on keeping information private and ensuring that only the right people will have access to it [28]. In the following, we define an S&D pattern that can guarantee this property, called *link-to-link confidentiality pattern*.

**Pattern Topology:** The topology of the link-to-link confidentiality network pattern with two nodes $N_1$ and $N_2$ is sequential. This is expressed by path $P$ between $N_1$ and $N_2$: $P = Path(source = N_1, destination = N_2)$. $P$ may be either an atomic link or path composition. The decomposition phase (as shown in Figure 3) can be analyzed as follows: $P = Path(source = N_1, destination = N_2) = Path(source = N_1, destination = N_3) \wedge Path(source = N_3, destination = N_2)$.

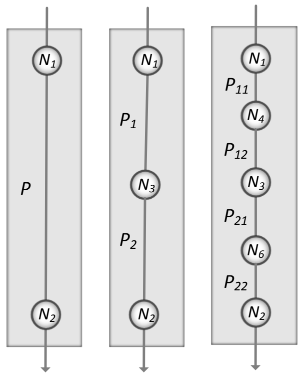

Figure 3: Sequence Decomposition

**Pattern Constraints:** Further constraints of the link-to-link confidentiality pattern relate to the distance between edge nodes. A constraint of this type expresses that if the maximum link range is $r$, the distance between edge nodes of a link-to-link composition should be $r \geq Distance(N_1, N_2)$.

**Pattern S&D Property:** Link-to-link encryption protects traffic flows from monitoring since all data (payload and headers) are encrypted/decrypted in every hop. When two nodes $N_1$ and $N_2$ are connected following the sequence pattern, the path is confidential when both nodes are able encrypt and to share encrypted data. This is expressed by the relation: $Path(N_1, N_2, encryption = true) \rightarrow Node(N_1, encryption = true) \wedge Node(N_2, encryption = true)$.

**Pattern Inference Rule:** The confidential rule (Rule 1) encodes the sequence workflow pattern topology. In the LHS of this pattern, the rule matches two nodes (*lines 3-4*) and a path *$P* with source the *$N1* and destination the *$N2* (*line 5*). The constraint of the pattern topology defines that the link range *$r* should be less or equal to the distance between *$N1* and *$N2*. The S&D property *$reqPro* that the pattern should guarantee is presented in *lines 6-7*. When the topology constraint and the S&D property are not satisfied the rule will enter in the RHS of the rule. In the RHS, a new node *$N3* should be inserted between the *$N1* and *$N2* (*lines 9-10*). Moreover, two new paths $P1 and $P2 (*lines 11,14*) and two new requirements $R1 and $R2 (*lines 12-13,15-16*) for these paths will be inserted in the knowledge base. Finally, the rule will modify the requirement of the satisfaction to true (*line 17*). The recursive procedure will complete when the minimum number of nodes satisfy the distance constraint and therefore the S&D requirement.

Rule 1: Inference Rule of Link-to-Link Confidentiality Pattern

```
1  rule "Link-to-link Confidentiality Inference Rule"
2   when
3     $N1: Node($id1:id, $p1:position, encryption==true)
4     $N2: Node($id2:id, $p2:position, encryption==true)
5     $P:  Path(source==$N1, destination==$N2, $r:range, $d:distance, $r<=$d)
6     $R:  Requirement(path == $P, property.name=="Encryption",
7                        $reqPro:property.value, satisfied==false)
8   then
9     Node $N3 = new Node($id1+$id2, new Position($N1,$N2), true);
10    insert($N3);
11    Path $P1 = new Path($N1,$N3); insert($P1);
12    R1= new Requirement($P1,new Property("Encryption"),$r,false);
13    insert(R1);
14    Path $P2 = new Path($N3,$N2); insert($P2);
15    R2 = new Requirement($P2,new Property("Encryption"),$r,false);
16    insert(R2);
17    modify($R){satisfied=true};
18 end
```

**Pattern Verification Rule:** The second confidentiality rule (Rule 2) expresses the verification procedure in case of an existing SDN network design. The paths can be given by the use of an algorithm such as the depth-first algorithm. The purpose of this verification rule is to select the path/paths (*line 3*) in which S&D confidentiality property

$\$Pro$ is guaranteed. When the path encryption is enabled ($\$reqPro = \$Pro$), as defined by the requirement $\$R$ (*line 4-6*), the rule in the RHS will modify the requirement satisfaction to true (*line 8*).

Rule 2: Verification Rule of Link-to-Link Confidentiality Pattern

```
1 rule "Link-to-Link Confidentiality Verification Rule"
2  when
3    $P:  Path($N1:source,$N2:destination, $Pro: property)
4    $R:  Requirement(path.source==$N1, path.destination == $N2,
5              property.name=="Encryption", $reqPro: property.value,
6              $reqPro==$Pro, satisfied== false)
7  then
8    modify($R){satisfied=true};
9 end
```

## 4.2. *Redundancy Availability Pattern*

Network availability is the ability of a system to be operational and accessible when required for use [28]. Availability patterns can be used for the discovery and the verification of composition of network elements with guaranteed availability properties. The description of the redundancy availability pattern is following:

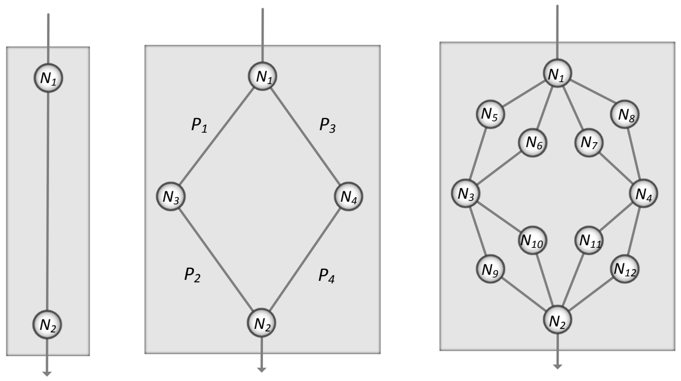

Figure 4: Redundancy Pattern Decomposition

**Pattern Topology:** The topology of the redundancy pattern follows both the sequence and the multi-choice-join structure. 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: $P_1 = Path(source = N_1, destination = N_3)$, $P_2 = Path(source = N_3, destination = N_2)$, $P_3 = Path(source = N_1, destination = N_4)$, $P_4 = Path(source = N_4, destination = N_2)$. The decomposition phase can be analyzed as follows: $P = Path(source = N_1, destination = N_2) = (Path(source = N_1, destination = N_3) \wedge Path(source = N_3, destination = N_2)) \vee (Path(source = N_1, destination = N_4) \wedge Path(source = N_4, destination = N_2))$. The decomposition procedure (Figure 4) can continue until atomic links are found.

**Pattern Constraints:** The constraint of redundancy availability pattern refers to the connectivity between nodes, expressing that if the maximum link range is $r$, the distance $d$ between these nodes should be $r \geq d$. When constructing a network from a source node $N_1$ to a destination host $N_2$, any nodes that are added to create the network between $N_1$ and $N_2$ are assumed to be on the same straight line, even they have been created by the multi-choice-join composition pattern.

**Pattern S&D Property:** The availability guaranteed by this pattern is related to the sequence and multi-choice path composition. When $m$ are the number of parallel paths $P$, $n$ are the number of sub-paths of each parallel path and $Pro(P)$ is the probabilistic availability of each sub-path, the probabilistic availability $Pro$ of the composition can be given by the following formula:

$$Pro = 1 - \prod_{i=1}^{m}(1 - \prod_{j=1}^{n} Pro(P_{i_j}))$$

Since the topology of redundancy availability pattern consists of two parallel paths with two sub-paths in sequence $((P_1 \wedge P_2) \vee (P_3 \wedge P_4))$, the availability $Pro$ will be equal to: $Pro = 1 - (1 - Pro(P_1) \cdot Pro(P_2))(1 - Pro(P_3) \cdot Pro(P_4))$. When the required availability property of the entire path is $reqPro$, the network availability should satisfy the following condition: $reqPro \leq Pro$. In case of equal uptime probability of each sub-path ($Pro(P_1) = Pro(P_2) = Pro(P_3) = Pro(P_4) = Pro(P)$), the required availability should satisfy the equation: $reqPro \leq Pro = 1 - (1 - Pro(P)^2)^2 \Rightarrow Pro(P) \geq \sqrt{1 - \sqrt{1 - reqPro}}$. If there is not any atomic path with this availability, 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 availability of each new path will be: $reqPro' = \sqrt{1-\sqrt{1-reqPro}}$. It can easily be proven that the $reqPro' \leq reqPro$ applies for requested path availability greater than 62%. Finally, the recursive execution of the pattern will increase network availability and will guarantee the required path availability.

Rule 3: Inference Rule of Redundancy Availability Pattern

```
1  rule "Redundancy Availability Inference Rule"
2   when
3     $N1: Node($id1:id,$p1:position)
4     $N2: Node($id2:id,$p2:position)
5     $P:  Path($N1==source,$N2==destination,$r:range,$d:distance,$r<=$d,
6               Pro.name=="Availability",$Pro:Pro.value)
7     $R:  Requirement(path==$P,property.name=="Availability",
8                 $reqPro:property.value,$Pro<$reqPro, satisfied==false)
9   then
10    Node $N3 = new Node($id1+$id2,new Position($N1,$N2)); insert($N3);
11    Node $N4 = new Node($id1+$id2,new Position($N1,$N2)); insert($N4);
12    Path $P1 = new Path($N1,$N3,$r,$Pro);        insert($P1);
13    insert(new Requirement($P1,new Property("Availability",
14            Math.sqrt(1-Math.sqrt(1-$reqPro))),false));
15    Path $P2 = new Path($N3,$N2,$r,$Pro); insert($P3);
16    insert(new Requirement($P2,new Property("Availability",
17            Math.sqrt(1-Math.sqrt(1-$reqPro))),false));
18    Path $P3 = new Path($N1,$N4,$r,$Pro ) ;insert($P2);
19    insert(new Requirement($P2,new Property("Availability",
20            Math.sqrt(1-Math.sqrt(1-$reqPro))),false));
21    Path $P4 = new Path($N4,$N2,$r,$Pro); insert($P4);
22    insert(new Requirement($P4,new Property("Availability",
23            Math.sqrt(1-Math.sqrt(1-$reqPro))),false));
24    modify($R){satisfied=true};
25  end
```

**Pattern Inference Rule:** The pattern rule encodes the described redundancy topology (Rule 3). In the LHS of this pattern, the rule matches two nodes *$N1* and *$N2* (*lines 3-4*). The pattern also matches a path *$P* with source the *$N1* and destination the *$N2* (*lines 5-6*). The constraint of the pattern topology defines that the link range *$r* between the nodes should be less or equal to the distance between *$N1* and *$N2*. The S&D property *$reqPro* that the pattern should guarantee is specified in *lines 7-8*. When the topology constraint and the S&D property are not satisfied the rule will enter in the

RHS of the rule. In the RHS, two nodes *$N3* and *$N4* should be inserted in parallel between the *$N1* and *$N2* (*lines 10-11*). Moreover, four new paths and requirements will be inserted in the knowledge base (*line 12-23*). Finally, the rule will modify the requirement satisfaction to true. The recursive procedure will be completed when the distance constraint and required availability property are satisfied.

Rule 4: Verification Rule of Redundancy Availability Pattern

```
1 rule "Verification of Redundancy Availability Rule"
2  when
3    $P:   Path($N1:source,$N2:destination ,$Pro:property.value)
4    $R:   Requirement($P==path ,property.name=="Availability",
5              $reqPro:property.value,$Pro>= $reqPro ,satisfied==false)
6  then
7    modify($R){satisfied=true};
8  end
```

**Pattern Verification Rule:** Rule 4 expresses the verification process, which according to the redundancy availability pattern can be followed to check a network's availability. The rule is able to discover suitable paths (*line 3*) with verified availability properties (*line 4-5*) by the use of a predefined set of paths applied in a depth first manner.

## 5. Implementation and Tool Support

Design patterns can be used for the design of the SDN infrastructure layer or the verification of existing SDN infrastructures. To give a proof of concept of our approach and evaluate its applicability for the design and verification of SDN networks we have developed a prototype implementing our framework. In the next subsections, the analysis of each implementation phase of the S&D Pattern Framework will be presented.

### *5.1. Architectural Framework*

The architectural framework used in our approach includes the developed *S&D Pattern Framework* which applies in an SDN architecture, as shown in Figure 5. The SDN network architecture consists of three high level layers: the infrastructure layer, the control layer and the application layer. The infrastructure layer is responsible for the data forwarding functionality of the network between end hosts through programmable switches. 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. The control layer is connected with the infrastructure layer through the so-called *Southbound Interface (SBI)*. Finally, the application layer includes SDN applications, SDN/network management and security/dependability management. It is also able to interact with the control layer through the *Northbound Interface (NBI)*.

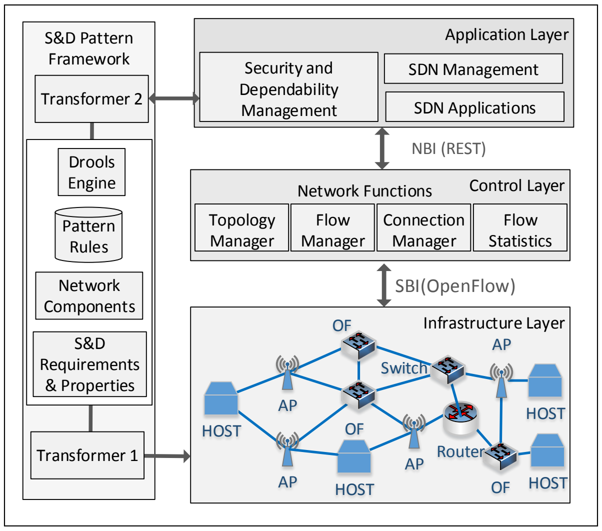

Figure 5: Framework Architectural Diagram

The S&D Pattern Framework contains *Drools Engine* and suitable Java classes. The rules defining S&D patterns were deployed in Eclipse Modelling Tool (4.5) with the JBoss Drools 6.3. 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*. The framework can interact with SDN architecture using suitable transformers. The *transformer 1* can export network topologies, as generated by Drools, into a custom format acceptable by Mininet[1], an emulator which is able to create realistic virtual SDN networks. The created cus-

[1]http://mininet.org/

tom configuration file may contain nodes (i.e., hosts and switches) and links of the network. Especially with the use of simulators such as Mininet-WiFi [29] and NS3[2], it is possible to include not only switches and hosts, but also OpenFlow-enabled access points. The *transformer 2* is able to import existing network topologies and flows from the inventory list of the *controller* such as OpenDaylight[3]. It is also 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 have been developed in Java as part of our framework.

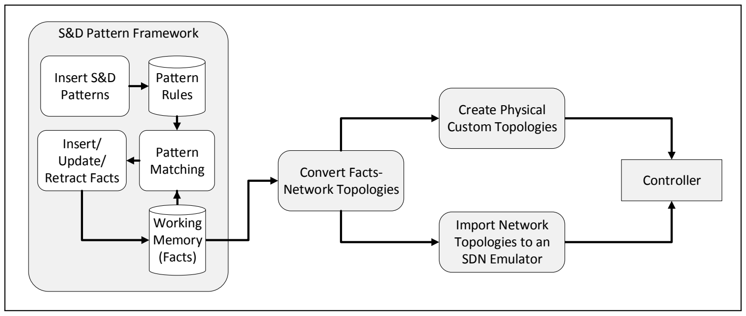
S&D Pattern Framework
Insert S&D Patterns
Pattern Rules
Insert/ Update/ Retract Facts
Pattern Matching
Working Memory (Facts)
Convert Facts-Network Topologies
Create Physical Custom Topologies
Import Network Topologies to an SDN Emulator
Controller

Figure 6: Process Diagram of S&D Pattern Framework for SDN Design

## 5.2. *Design and Verify SDN using the Framework*

The framework that we have developed can be used to design an SDN infrastructure that satisfies particular S&D properties as shown in the process diagram in Figure 6. More specifically, a network designer can *Insert S&D Patterns* in our tool as *Pattern Rules* and descriptions of network and S&D network requirements and constraints as *Facts* in the *Working Memory* of the framework. The tool then uses Drools to apply *Pattern Matching* and identify if a network can be formed out of the available types of nodes that satisfies 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 Network Topologies* to an SDN emulator. Finally, the created network will be inserted in the inventory list of the *Controller*.

[2]http://www.nsnam.org/
[3]https://www.opendaylight.org

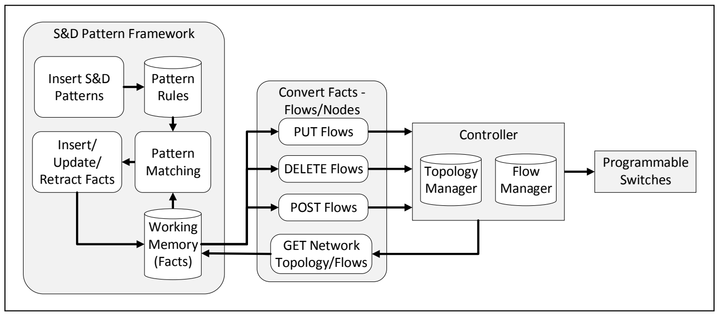

Figure 7: Process Diagram of S&D Pattern Framework for SDN Verification and Adaptation

The verification of an existing SDN network with regards to S&D, is supported by our S&D pattern framework as shown in Figure 7. 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 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. Finally, 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 a S&D property is not guaranteed. The use of our framework at runtime can not only verify a network but also re-construct it to restore required S&D properties in cases where such properties have been violated.

## 6. Evaluation and Experiments

The implementation described in Section 5 has been used for an evaluation of our approach in two different SDN design scenarios. These scenarios and the outcomes of the evaluation are presented in the following subsections.

### *6.1. Scenario 1 - Design of SDN Networks*

The first scenario involves the transmission between two host nodes (source and destination) using wireless-enabled network nodes. Apart from the source and the destination, all the other nodes act as relays that send the received data continuously. To design an S&D network able to avoid attacks on the communication medium such as eavesdropping and DoS, the confidentiality and availability patterns that were discussed in Section 4 are applied. The inputs to the pattern based network design tool for both S&D patterns are: (a) the distances between source and destination node of the network are 500m, 1.000m, 2.000m, 5.000m, 7.000m and 10.000m and (b) the maximum range of communication link is 100m. The outputs that the tool generates are: (i) the network nodes, (ii) their position (i.e., the tier in which the nodes should be placed) and (iii) the number of links.

*Link-to-Link Confidentiality Pattern:* The pattern requires that the exchanged data on the communication channel should be encrypted. Therefore, each node should be able to encrypt/decrypt data by applying link-to-link encryption.

*Redundancy Availability Pattern:* The pattern implies that the path availability is related to the probability of an attack. In our experiments we considered 99% the uptime probability and the required network availability is 99.999% (or less than one-minute daily network downtime) network.

Table 1: Results of Conducted Experiments

| | Confidentiality Pattern | | Availability Pattern | |
|---|---|---|---|---|
| Distance (metres) | Required Relay Nodes | Exec.Time (msec) | Required Relay Nodes | Exec.Time (msec) |
| 500 | 4 | 44 | 12 | 56 |
| 1.000 | 8 | 58 | 44 | 81 |
| 2.000 | 16 | 60 | 170 | 192 |
| 5.000 | 32 | 85 | 684 | 1487 |
| 10.000 | 64 | 101 | 2734 | 7530 |

The results of the tool, after applying the above two patterns, are presented in Table 1. The table shows the number of nodes of each pattern and the time that was needed to execute each pattern. The number of required nodes produced by the confidentiality pattern, represents also the minimum number of nodes for a functional network, although the purpose of this pattern is to enable link-to-link encryption. On the other hand, the requirement of 99.999% availability suggests that a great number of nodes should be installed, especially for long distance links.

The developed S&D network topologies can be transformed to an SDN network by the use of Mininet emulator, as discussed in Section 5. The created SDN infrastructure can include hosts and OpenFlow-enabled (wired or wireless) switches as obtained by the S&D patterns. Then, the emulator is able to forward the topology to a remote controller such as OpenDaylight. Figure 8 depicts the outputs (nodes and links) of the redundancy pattern when the distance between the source and the destination is 500m, the range is 100m and the uptime probability is 99%.

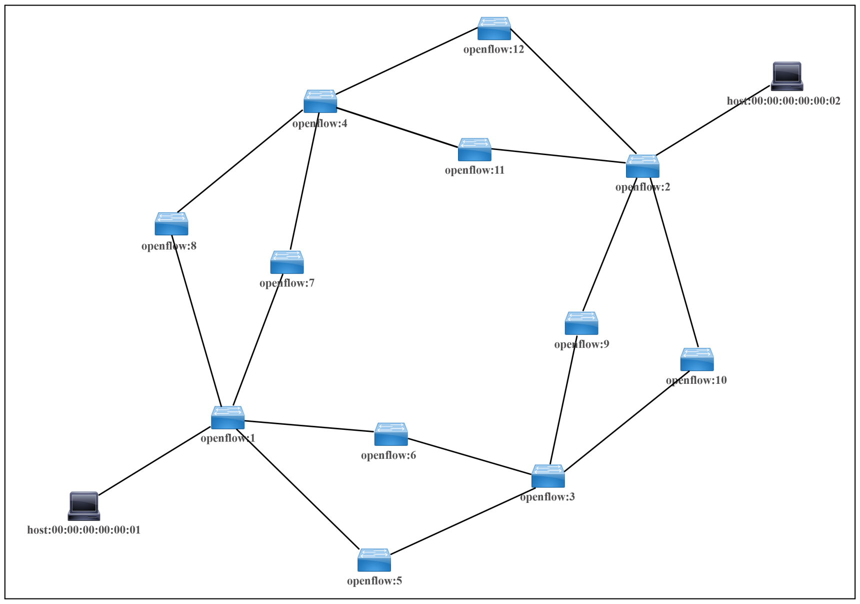

Figure 8: OpenDaylight SDN Infrastructure Topology

The scenario could be extended by adding multiple end-host. However, the networks that would be generated between the same source node and multiple end-hosts will be disjoint. This is due to the fact that the current set of patterns *place* nodes on a straight line between a source and a destination (this point has been clarified in Section 4 in the paper). Thus, adding end hosts merely adds disjoint networks (and their *costs* should be added).

### 6.2. Scenario 2 - Verification and Runtime Adaptation of SDN Networks

The second scenario includes the verification of an existing SDN infrastructure (such as the network topology shown in Figure 8). The initial network topology (nodes and links) can be obtained from the topology manager of OpenDaylight controller as discussed in Section 5. However, in this scenario, network nodes and links have different channel availability and encryption level as presented in Table 2.

Table 2: S&D Properties of Network Topology

| Links | $l_1(n_1,n_5)$ | $l_2(n_1,n_6)$ | $l_3(n_1,n_7)$ | $l_4(n_1,n_8)$ | $l_5(n_5,n_3)$ | $l_6(n_6,n_3)$ | $l_7(n_7,n_4)$ | $l_8(n_8,n_4)$ | $l_9(n_3,n_9)$ | $l_{10}(n_3,n_{10})$ | $l_{11}(n_4,n_{11})$ | $l_{12}(n_4,n_{12})$ | $l_{13}(n_9,n_2)$ | $l_{14}(n_{10},n_2)$ | $l_{15}(n_{11},n_2)$ | $l_{16}(n_{12},n_2)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Encrypted | ✓ | ✓ | X | ✓ | X | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | X | ✓ | ✓ |
| Availability (%) | 99 | 99 | 97 | 99 | 98 | 99 | 99 | 98 | 99 | 97 | 98 | 99 | 98 | 99 | 97 | 99 |

S&D verification patterns can be executed to define paths and convert them to OpenFlow rules with high priority. As an evaluation test of our approach, the following requirements were inserted in the working memory:
*R1= Requirement(path.source == \$n1, path.destination == \$n2, property.name == "Availability", property.value == 0.95, satisfied == false)* and
*R2 = Requirement(path.source == \$n1, path.destination == \$n2), property,name == "Encryption", property.value == true, satisfied == false).*

After the execution of confidentiality and availability verification patterns, a number of possible solutions were produced. The paths that guarantee both properties are presented in the following expression in which the $\wedge$ represents the sequence composition and $\vee$ represent the parallel composition:

$$S_{R_1} \wedge S_{R_2} = l_4 \wedge l_8 \wedge ((l_{11} \wedge (l_{15}) \vee (l_{12} \wedge l_{16}))$$

Verification patterns can also be executed to support runtime SDN adaptation in cases like DoS attacks. In a network fall for example, new alternative network paths must be found. However, the most important factors for runtime adaptation appear to be both the detection to identify an attack or failure and the reaction time to transfer the new flow rules to the controller and the switches. This can be done by the use of node connector statistics as fetched from the OpenFlow-enabled switches to the OpenDaylight controller. The statistics include receive and transmit packets, errors, drops CRC errors and collisions. S&D patterns can retrieve these statistics and react immediately as an intrusion detection mechanism in case of malicious adversaries that create DoS attacks and forward traffic to different secure paths.

## 7. Conclusion and Future Work

In this paper, we proposed an S&D patterns based framework for the design and verification of networks that need to satisfy given security and dependability properties. This is crucial for the design of highly interconnected ICT and CPS that need to preserve S&D properties as the network is the backbone for ensuring such properties. Our approach is based on S&D patterns which express what 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. It also includes processes for applying the S&D patterns for this purpose. Our approach is aimed at minimizing the effects of passive and active attacks on physical layer. To prove the applicability of our S&D patterns approach, we developed a prototype tool supporting the design and verification of SDN network infrastructures and evaluated in initial experiments involving design and verification scenarios involving the security properties of confidentiality and availability. Our future plans involve the development of the patterns specification scheme and pattern instances suitable to guarantee properties not only at the infrastructure layer, but also at the control and application layer. We are also planning to extend our framework with more S&D patterns and new functional capabilities to cover not only horizontally layered designs but also vertical layers of SDN architectures.

## References

[1] P. Agyapong, M. Iwamura, D. Staehle, W. Kiess, A. Benjebbour, Design considerations for a 5g network architecture, Vol. 52, IEEE, 2014, pp. 65–75.

[2] N. E. Petroulakis, G. Spanoudakis, I. G. Askoxylakis, A. Miaoudakis, A. Traganitis, A pattern-based approach for designing reliable cyber-physical systems, in: Globecom 2015, IEEE, 2015.

[3] D. C. Schmidt, Model-driven engineering, in: Computer Society, Vol. 39, 2006, pp. 286–298.

[4] K. Doddapaneni, E. Ever, O. Gemikonakli, I. Malavolta, L. Mostarda, H. Muccini, A model-driven engineering framework for architecting and analysing wireless sensor networks, in: SESENA, 2012.

[5] C. Preschern, N. Kajtazovic, C. 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, ACM, 2012, p. 5.

[6] B. Hamid, C. Percebois, D. Gouteux, A methodology for integration of patterns with validation purpose, in: Proceedings of the 17th European Conference on Pattern Languages of Programs, ACM, 2012, p. 8.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns: elements of reusable object-oriented software, Pearson Education, 1994.

[8] G. Spanoudakis, S. Kokolakis, Security and Dependability for Ambient Intelligence, Springer Science & Business Media, 2009.

[9] H. Mouratidis, Software Engineering for Secure Systems: Industrial and Research Perspectives: Industrial and Research Perspectives, IGI Global, 2010.

[10] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad, Security Patterns: Integrating security and systems engineering, John Wiley & Sons, 2013.

[11] E. Fernandez-Buglioni, Security patterns in practice: designing secure architectures using software patterns, John Wiley & Sons, 2013.

[12] B. Hamid, J. Geisel, A. Ziani, J.-M. Bruel, J. Perez, Model-driven engineering for trusted embedded systems based on security and dependability patterns, in: SDL 2013: Model-Driven Dependability Engineering, Springer, 2013, pp. 72–90.

[13] H. Petritsch, Service-oriented architecture (soa) vs. component-based architecture, 2006.

[14] G. Gössler, J. Sifakis, Composition for component-based modeling, 2005.

[15] L. Pino, K. Mahbub, G. Spanoudakis, Designing Secure Service Workflows in BPEL, in: ICSOC 2014, Paris, France, 2014.

[16] L. Pino, G. Spanoudakis, A. Fuchs, S. Gürgens, Discovering secure service compositions, in: 4th International Conference on Cloud Computing and Services Sciences, Barcelona, Spain, 2014.

[17] M. C. Jaeger, G. Rojec-goldmann, M. Gero, QoS Aggregation for Web Service Composition using Workflow Patterns, no. Edoc, 2004.

[18] E. Liotou, G. Tseliou, K. Samdanis, D. Tsolkas, F. Adelantado, C. Verikoukis, An sdn qoe-service for dynamically enhancing the performance of ott applications, in: Quality of Multimedia Experience (QoMEX), 2015 Seventh International Workshop on, IEEE, 2015, pp. 1–2.

[19] R. Vilalta, A. Mayoral, D. Pubill, R. Casellas, R. Martínez, J. Serra, C. Verikoukis, R. Muñoz, End-to-end sdn orchestration of iot services using an sdn/nfv-enabled edge node, in: Optical Fiber Communication Conference, Optical Society of America, 2016, pp. W2A–42.

[20] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in sdn-openflow networks, Vol. 71, Elsevier, 2014, pp. 1–30.

[21] J. Reich, C. Monsanto, N. Foster, J. Rexford, D. Walker, Modular sdn programming with pyretic, 2013.

[22] W. Zhou, L. Li, M. Luo, W. Chou, Rest api design patterns for sdn northbound api, in: Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on, IEEE, 2014, pp. 358–365.

[23] J. Halpern, C. Pignataro, Service function chaining (sfc) architecture, Tech. rep. (2015).

[24] R. Cohen, K. Barabash, B. Rochwerger, L. Schour, D. Crisan, R. Birke, C. Minkenberg, M. Gusat, R. Recio, V. Jain, An intent-based approach for network virtualization, in: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, IEEE, 2013, pp. 42–50.

[25] W. Van Der Aalst, A. Ter Hofstede, B. Kiepuszewski, A. Barros, Workflow patterns, Vol. 14, Springer, 2003.

[26] Drools, Business rules gement system solution, `www.drools.org`.

[27] C. Forgy, Rete: A fast algorithm for the many pattern/many object pattern match problem, Vol. 19, Elsevier, 1982, pp. 17–37.

[28] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, F. Springsteel, IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries, 1991.

[29] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, C. E. Rothenberg, Mininet-wifi: Emulating software-defined wireless networks, in: 11th International Conference on Network and Service Management (CNSM), IEEE, 2015.