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 at design or runtime level. In particular, with the deployment of Software Defined Networks (SDN), S&D weaknesses may occur in all the different layers of SDN architectures. In order to design SDN network architectures with respect to S&D, a model-based approach such as design patterns can be used adequately. In this work, we define executional patterns for designing networks able to guarantee S&D properties in legacy and SDN networks. To evaluate our defined pattern approach, we implement executable pattern instances, encoded in a rule-based reasoning system, able 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

[*] 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 implementation on cyber-physical networks. However, the design of such systems encounters difficulties which need to be resolved as the nature of the systems are time-critical, distributed, intelligent and heterogeneous. The validation and the verification methods for developing secure and dependable system networks are 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 introducing revolutionary concepts like Software Defined Network (SDN), integrated on 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 cyber-physical applications and drive the reconfiguration of these capabilities through network configuration specifications embedded through the cyber-physical 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. Design patterns can be used to design networks or to validate SDN network infrastructures and identify suitable paths and nodes with respect to S&D properties. S&D patterns can be used to design SDN infrastructures determining also the type, the location and the connectivity of end nodes with forwarding devices. At the control layer, design patterns can ensure secure connectivity between the controllers and the programmable switches. In this paper, we give a detailed description of the pattern definition form for the design of S&D legacy and 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 this pattern-based approach for designing reliable cyber-physical networks was given in [2]. However in this work, we developed a pattern framework in which we can evaluate and emulate S&D executable patterns on legacy and SDN-based network designs. Finally, an application framework is proposed, 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 3, we present the schema of the pattern execution form. In Section 2 an overview of related work is presented. 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 legacy and 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 steps of our work.

2. Related Work

The main focus of existing work for the design of systems and networks focuses 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. The design of system is simplified through the modelling of design patterns. MDE applies design patterns [4, 5] as solutions for reusable designs and interactions of objects by the use of formal proven properties[6]. An MDE framework for architecting wireless networks is presented in [7]. Especially, the construction of network topologies includes the definition of network and traffic patterns. Authors in [8] address a set of optimal patterns to achieve full coverage in wireless networks. Especially with the softwarization of networks in SDN, design patterns can be applied in all the different layers of SDN architectures. Traffic engineering and patterns in SDN are presented in [9]. Flow policy patterns as expressed by Frenetic languages, can generate flow rules able to be installed in programmable switches of SDN networks [10]. In our approach, we can provide paths as flow rules based on the security requirements. Finally, design patterns can also be used in northbound interface using REST API as proposed in [11].

The development of S&D network patterns may benefit from the current implementations of software patterns as described in the literature in a variety of works [12], [13], [14], [15], [16]. 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 [17, 18]. Safety and reliability patterns are presented in [19]. The author describes a set of reliable patterns able to offer redundancy on data transmissions for real-time and embedded systems. Workflow pattern for QoS aggregation for web service composition have been proposed in [20]. In our approach, executable workflow patterns are used for backward chaining for network compositions. Security workflow patterns, for service compositions based on enabling reasoning engines such as Drools, are also described in [21, 22]. Drools enabling reasoning appeared to be also an efficient rule engine to represent our network workflow patterns.

3. Network Pattern Schema

SDN infrastructures are based on an architectural framework where the network elements can be 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. Design patterns can be used 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. Finally, at runtime patterns are applied to the design of an operational network that is already known to satisfy given S&D properties. The pattern specification schema is defined as follows:

Definition 1. *An S&D network patterns schema is an abstract structure of specifying S&D network patterns 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.*

3.1. Pattern Topology

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 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, 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 evidence) 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.

In our patterns so far, we have focused on logical architecture of 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 [23]. As it can be seen in Figure 1, 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

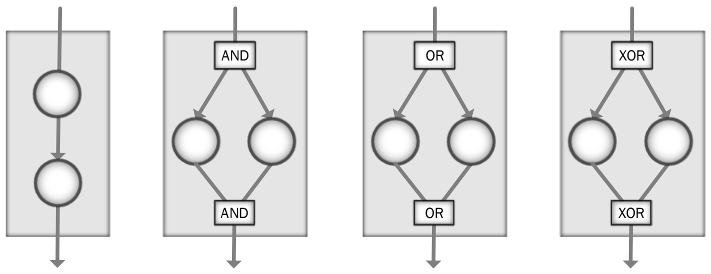


Figure 1: Basic SDN pattern logical topologies: (a) sequence (b) parallel-split-join (c) multi-choice-join (d) exclusive-choice-join

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. Exclusive disjunction can be used in networks in order to avoid flooding and for conditional routing.

3.2. Pattern Constraints

The SDN 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. In order to complete the topology of the pattern specification, the function constraints should be included. The constraints represent the functional requirements in which a pattern should hold. The connectivity between two components is one of the most critical functional requirements especially for network. This can be related to the type of components (hardware/software). Different parameters such as distance or other constraints are crucial points for the definition of the placeholders. 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, running applications and services are crucial factors on the design of networks. These applications are related to the available resources such as computational power, available memory, storage and networking capabilities. Other constraints which should be considered are 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, the medium such as a wireless link cannot be modified, in order to guarantee a security property of the system, the property should be satisfied at both the output of the source node and at the input of the destination node.

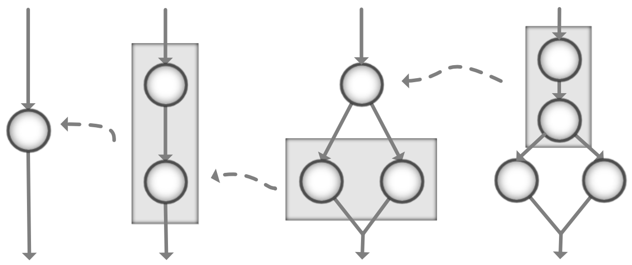


Figure 2: Stepwise Decomposition

S&D patterns can be used to recursively build component compositions or decompositions using forward or backward chaining respectively, as depicted in Figure 2. In forward chaining, when $C_1,..C_n$ are components that satisfy a property *Pro* then the composition *C* formed of these components can satisfy *Pro* when the following form can be proved:

$$\text{When } C_1 \wedge \cdots \wedge C_n \text{ satisfy } Pro \rightarrow C \text{ satisfies Pro}$$

However, 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*:

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

Let a sequential composition of two components: $C \rightarrow C_1 \wedge C_2$. If a required property should be guaranteed by the composition, the subcomponents should satisfy the condition $reqPro(C) \rightarrow reqPro(C_1) \wedge reqPro(C_2)$. If there are no atomic components to guarantee the required property, 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:

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

3.4. Pattern Rules

Drools production system was chosen 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 [24]. Drools rules can encode the topology of a pattern which main target is to find 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 the system does not satisfy the required property, the pattern has to substitute, add or remove components. A Drools rule includes the inputs of the involved components, the type of composition and the required S&D property in Left Hand Side (LHS). When the conditions of a rule in the LHS are satisfied, then the rule is fired to execute the actions as described in the Right Hand Side (RHS). In the RHS, the new requirements of the compositions or atomic components can be inserted, updated or deleted in the knowledge base.

4. Network Pattern Specification Forms

In this section we present different abstract specification pattern instances able to guarantee confidentiality and availability in network infrastructures based on the previous described pattern form.

4.1. Link-to-link Confidentiality Pattern

Confidential transmission on the infrastructure layer focuses on keeping information private ensuring that only the right people will have access to it[25]. In the following, we define the link-to-link confidentiality pattern.

Pattern Topology: The topology of the link-to-link confidentiality network pattern follows arrangements of nodes N_1 and N_2 as described by the sequence pattern. There is also a 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 can be analyzed as follows: $P = Path(source = N_1, destination = N_2) = Path(source = N_1, destination = N_3) \wedge Path(source = N_3, destination = N_2)$. The decomposition procedure is depicted in Figure 3.

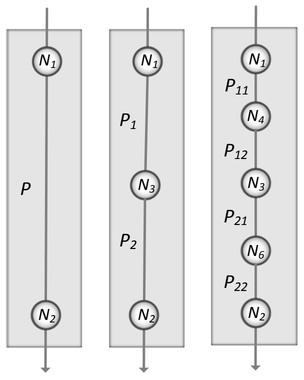


Figure 3: Sequence Decomposition

Pattern Constraints: The constraint of link-to-link confidentiality pattern includes the connectivity between nodes. Especially for atomic links, the network connectivity is related to the distance between edge nodes. When the maximum link range is r, the distance between these nodes should be $r \geq Distance(N_1, N_2)$.

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

```
1  rule ”Inference of Link−to−link Confidentiality”
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    insert(new Requirement($P1,new Property(”Encryption”),$r,false));
13    Path $P2 = new Path($N3, $N2); insert($P2);
14    insert(new Requirement($P2,new Property(”Encryption”),$r,false));
15    modify($R){satisfied=true};
16  end
```

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 to share encrypted data: $Path(N_1,N_2,encryption = true) \rightarrow Node(N_1,encryption = true) \wedge Node(N_2,encryption = true)$

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

```
1  rule ”Verification of Link−to−Link Confidentiality”
2   when
3     $P:  Path($N1:source,$N2:destination, $Pro: property)
4     $R:  Requirement(path.source==$N1, path.destination == $N3,
5                property.name==”Encryption”, $reqPro: property.value,
6                $reqPro==$Pro, satisfied== false)
7   then
8     modify($R){satisfied=true};
9  end
```

Pattern Inference Rule: The confidential rule 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 and two new requirements $R1 and $R2 for these paths will be inserted in the knowledge base (*line 11-14*). Finally, the rule will modify the requirement of the satisfaction to true. The recursive procedure will complete when the minimum number of nodes satisfy the distance constraint and therefore the S&D requirement.

Pattern Verification Rule: The second confidentiality rule, includes the verification procedure in case of an existing SDN network design. The paths in which confidentiality property is satisfied can be given by the use of the depth-first algorithm. The required S&D confidentiality can be validated by the implication of Rule 2.

4.2. Redundant Availability Pattern

Network availability is the ability of a system to be operational and accessible when required for use [25]. Availability patterns can be used for the discovery of composition of network elements with verified availability properties. The description of the redundant availability pattern is following:

Pattern Topology: The topology of the redundant pattern follows both the sequence and the multi-choice-join pattern. The topology 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. The pattern 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 redundant availability pattern includes the connectivity between nodes. Especially for atomic nodes and paths, the network connec-

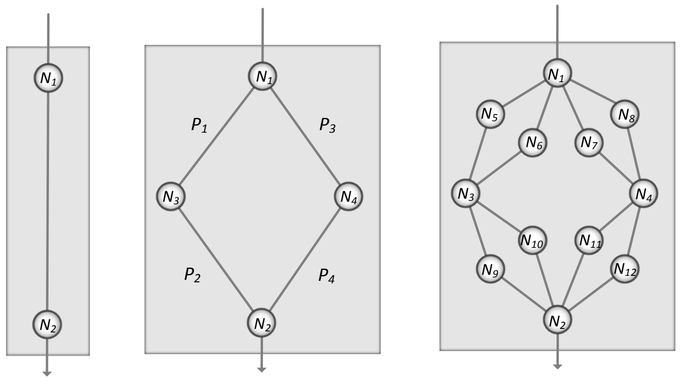


Figure 4: Redundant Pattern Decomposition

tivity is related to the distance between nodes. When the maximum link range is r, the distance d between these nodes should be $r \geq d$.

Pattern S&D Property: The availability of this pattern is related to the sequence and multi-choice node composition. Therefore, the availability of the redundant pattern Pro is 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: $reqPro \leq Pro$. More precisely, in case of equal atomic uptime probability ($Pro_{P_1} = Pro_{P_2} = Pro_{P_3} = Pro_{P_4}$), the required availability of each path should satisfy the following equation: $Pro \geq \sqrt{1-\sqrt{1-reqPro}}$. It can easily be proven that the recursive application of the pattern will increase network availability when the atomic link probability is $Pro \in (0.62, 1)$

Pattern Inference Rule: The pattern rule encodes the described redundant topology. 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 *$reqPro* that the pattern should guarantee is presented in *lines 7-8*. When the topology constraint and the S&D property are not satisfied the rule will enter in the RHS of the rule. In the RHS, two nodes *$N3* and *$N4* should be inserted in parallel between the *$N1* and *$N2* (*lines 10-11*). Moreover, four new paths and requirements will be

Rule 3: Inference Rule of Redundant Availability Pattern

```
1  rule "Inference of Redundant Availability"
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
```

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 Redundant Availability Pattern

```
rule "Verification of Redundant Availability"
 when
   $P:  Path($N1: source, $N2: destination, $Pro: property.value)
   $R:  Requirement($P==path, property.name=="Availability",
            $reqPro:property.value, $Pro>= $reqPro, satisfied== false)
 then
   modify($R){satisfied=true};
 end
```

Pattern Verification Rule: The verification availability rule, includes the validation of an existing network topology with respect to availability. The required availability property can be validated by the implication of Rule 4.

5. Implementation and Tool Support

Design patterns can be used for the design of SDN infrastructure layer or for the verification of existing SDN infrastructures. To give a proof of concept of our approach, we evaluate the applicability of the S&D pattern for the design and verification of SDN networks. The proposed framework is presented in Figure 5. In the next subsections, the analysis of each implementation phase of the pattern network will be presented.

5.1. Design Patterns and Tools

The rules defining S&D patterns were deployed in Eclipse Modelling Tool (4.5) with the JBoss Drools 6.3.0 extension. The class diagram of the created Java classes which are used by the Drools rules are depicted in Figure 6. Each *Link* includes a source and a destination *node*. In addition, a *Path* can include a *Link* or a set of interconnected links.

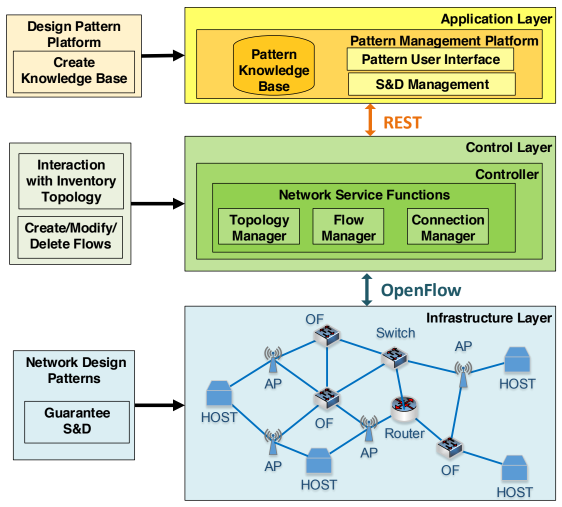


Figure 5: Proposed SDN Pattern Based Infrastructure

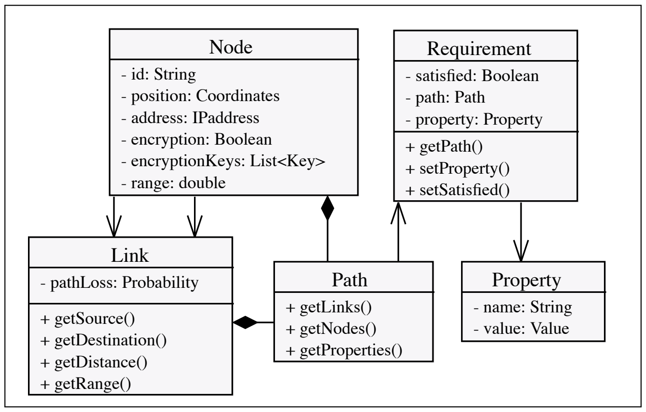


Figure 6: Class Diagram of Classes Available for the S&D Production Rules

5.2. Design SDN using a Pattern Framework

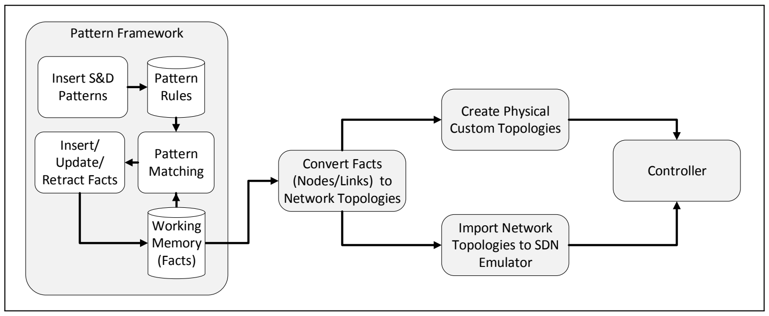
To design SDN infrastructures, we propose a pattern framework tool as shown in the activity diagram of Figure 7. A network designer can insert in our tool S&D patterns as Drools rules and descriptions of network and S&D network requirements and constraints as facts in the working memory of the pattern framework. The tool then uses Drools to apply the S&D pattern expressing rules and identify if a network can be formed out of the available types of nodes that satisfies the required S&D property. The topology generated by the patterns is exported into a custom format acceptable by the Mininet[1], an emulator which is able to create realistic virtual SDN networks. The created custom 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 [26] and NS3[2], it is possible to include not only switches and hosts, but also OpenFlow-enabled access points. .

5.3. Verify SDN using a Pattern Framework

The verification of an existing SDN network with regards to S&D, can be validated using our proposed pattern framework as depicted in Figure 8. In this framework, a designer is able to insert S&D patterns in the pattern rules production memory. Node and

[1]http://mininet.org/

[2]http://www.nsnam.org/



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

Figure 7: Activity Diagram of Pattern Framework for SDN Design

links together with the S&D requirements can be inserted in the working memory as facts. Suitable Java classes have been developed able to obtain (GET) nodes and links from the inventory list of the OpenDaylight controller[3]. After the execution of S&D patterns, new paths can be inserted (PUT) or current paths can be deleted (DELETE) or modified (POST). These paths can be expressed as flow rules in a XML format transferred to the controller using the RESTful interface. By 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. on link failure or when a S&D property is not guaranteed.

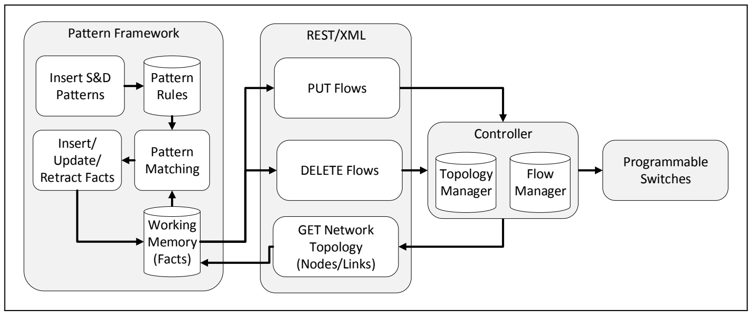


Figure 8: Activity Diagram of Pattern Framework for SDN Verification and Adaptation

[3]https://www.opendaylight.org

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 a S&D network able to avoid attacks on the communication medium such as eavesdropping and DoS, the described in Section 4 confidentiality and availability patterns 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, (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: It implicates 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.

Redundant Availability Pattern: It implicates 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.

After the execution of S&D patterns, the outcomes of the tool are presented in Table 1. As it can be seen in the table, apart from the number of nodes of each pattern, it can also provide the needed time to execute each pattern. The number of nodes produced by the confidentiality pattern, represents also the minimum number of nodes for a functional network. However, 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.

Table 1: Results of Conducted Experiments

Distance	Confidentiality Pattern		Availability Pattern	
(metres)	Nodes	Exec.Time (msec)	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 developed S&D network topologies can be transformed to SDN network by the use of Mininet emulator. The created SDN infrastructure can include hosts and OpenFlow-enabled (wired or wireless) switches as obtained by S&D patterns. Then, the emulator is able to forward the topology to a remote controller such as OpenDaylight. Figure 9 depicts the outputs (nodes and links) of the redundant pattern when the distance between the source and the destination is 500m, the range is 100m and the uptime probability is 99%.

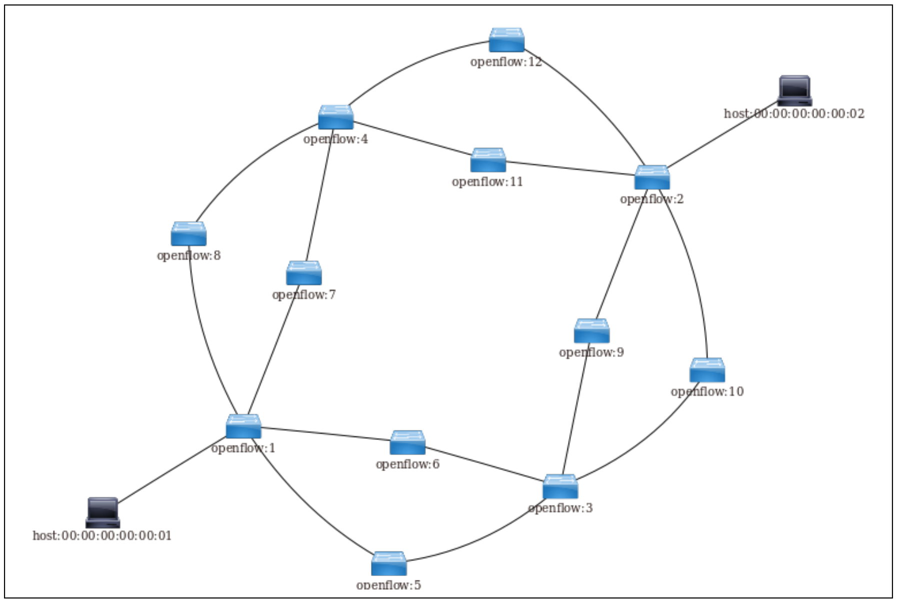


Figure 9: OpenDaylight SDN Infrastructure Topology

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 9) by the use the proposed pattern framework of Section 5.3.The initial network topology (nodes and links) can be obtained from the topology manager of OpenDaylight controller. 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 proactively to define paths and convert them to OpenFlow rules with high priority. To proceed with the evaluation of our apporach, the following requirements were inserted in the working memory: *R1= Requirement(path.source == $n1, path.desti-nation == $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, as defined by Drools rules, 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:

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

S&D verification patterns can also be executed at runtime adaptation such as a DoS attack. In case the a network fall, new alternative network paths must be found. A depth-first algorithm will have to re-evaluate the topology of the system. 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. Finally, this mechanism of S&D patterns can also be used as an intrusion detection mechanism in order to react in case of malicious adversaries that create DoS attacks.

7. Conclusion and Future Work

In this paper, we proposed a pattern framework in which S&D pattern can be used for the design and verification of network. More specifically, the development of legacy and SDN networks on critical infrastructures has introduced new challenges including security and dependability. Consequently, network modelling is crucial for design network with respect to S&D properties. Our work included a methodology on how to preserve a S&D property through patterns, encoded as rule-based reasoning. Our pattern-based approach aimed to minimize the effects of passive and active attacks on physical layer. To prove the applicability of our executable patterns, we developed an implementation able to design and validate SDN network infrastructures. Our future plans contain the completion of the pattern form, developing also pattern instances suitable to guarantee properties not only at the infrastructure layer, but also at the control and application layer. Finally, our future work will include the development of a complete framework 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] 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.

[5] 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.

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

[7] 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.

[8] Z. Yu, J. Teng, X. Bai, D. Xuan, W. Jia, Connected coverage in wireless networks with directional antennas, Vol. 10, ACM, 2014, p. 51.

[9] 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.

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

[11] 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.

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

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

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

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

[16] 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.

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

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

[19] B. Douglass, Real-time design patterns: robust scalable architecture for real-time systems, Vol. 1, Addison-Wesley Professional, 2003.

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

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

[22] 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.

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

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

[25] 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.

[26] 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.