



# How Formal Methods Can Contribute to 5G Networks

María-del-Mar Gallardo, Francisco Luque-Schempp, Pedro Merino-Gómez<sup>(✉)</sup>,  
and Laura Panizo

Universidad de Málaga, Andalucía Tech, Málaga, Spain  
{gallardo,fls,pedro,laurapanizo}@1cc.uma.es

**Abstract.** Communication networks have been one of the main drivers of formal methods since the 70's. The dominant role of software in the new 5G mobile communication networks will once again foster a relevant application area for formal models and techniques like model checking, model-based testing or runtime verification. This chapter introduces some of these novel application areas, specifically for Software Defined Networks (SDN) and Network Function Virtualization (NFV). Our proposals focus on automated methods to create formal models that satisfy a given set of requirements for SDN and NFV.

**Keywords:** 5G networks · Software Defined Networks · Network Function Virtualization · New Internet Protocols · Model checking · Model-based testing

## 1 Introduction

Formal method techniques and tools have been very close to protocol engineering since the 70's, when the first errors in communication protocols were detected with reachability analysis over their models in finite state machines [31]. In the 80's and 90's, standardization bodies in information and communication technologies, like ISO and ITU, recognized the benefits of formal modelling and automatic verification with languages like LOTOS, ESTELLE, SDL and, later, the sequence charts in UML. The paper by Bochmann et al. [3] is a good summary of the history of Protocol Engineering in those decades. Another relevant milestone at the beginning of the 21st century is the ACM System Award to SPIN, a tool originally designed to support protocol design and validation [14, 15]. However, in recent years, the applications of formal methods for communication networks seem to be less significant. In this position paper, we argue that 5G mobile communication networks could once again reinforce the role of communication networks and protocols as a relevant application domain for formal methods.

---

This work is partially supported by the projects EuWireless and 5GENESIS. These projects have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No. 777517 and No. 815178, respectively.

Mobile communication networks are evolving towards new paradigms in which softwarization is one of the key aspects. In particular, one of the most important features of 5G networks is the shift of vendor-locked networks to cloud-based systems to dynamically adapt the network to the changing user needs. To achieve these objectives, 5G networks implement the so-called *network slice*, which is similar to a private network tailored to run a service with specific *Key Performance Indicators* (KPI). For instance, a critical service, such as telesurgery, imposes hard constraints on packet latency (under 5 ms) in order to control a surgeon robot over a cellular network. Other services, such as high-resolution content delivery, require high downlink or uplink speed. In 5G networks, each kind of service will run on a different network slice that satisfies its performance and quality of service requirements, but all these network slices will be deployed over the same underlying infrastructure.

The implementation of the network slices will be feasible mainly thanks to two technologies: Software-Defined Networks (SDN) and Network Function Virtualization (NFV). SDN makes the key components in the network, the switches and routers, programmable and that way a single program can control aspects like traffic priorities, firewall capabilities or forwarding rules from a single point. NFV removes the vendor-locked network elements and provides a cloud-like method to deploy software versions of these components on demand.

The increasing presence and complexity of software in 5G networks will introduce new risks related to reliability and security, and will make it more difficult to predict the behavior of the whole system in case of an accident or on-purpose malfunctioning of some parts of the network. Continuous monitoring and other techniques from network operators will be more complex, but they will be useful at production time. Like in other application domains, formal methods could provide tremendous benefits by helping to design, deploy and operate such a complex system with analysis before the deployment of the network. Recently, some authors have been using different formal methods to ensure the correct deployment and configuration of SDNs [1, 2, 4, 5, 17, 18, 20, 21]. One of the first works is the NICE [4] tool, that combines symbolic model checking and different search strategies to find errors, for instance, host reachability problems or undesirable packet lost. Compared to a general purpose model checking tool, NICE can directly analyze the code of the software governing the network. Although NFV technology is more recent, there are also proposals [22, 28, 34] to integrate Verification and Validation processes in the NFV deployment cycle, as well as not only to find functional bugs in the network but also to ensure the performance requirements [29, 30], which is a very important issue in 5G networks. Most of these works are not connected to testbeds where some realistic experimentation can be combined with the models and the automatic analysis.

In the chapter, we propose the use of formal methods focusing on two aspects. First, modelling the SDN component of the network and checking that there are model instances satisfying a set of requirements, and second, the use of model-based testing and runtime verification to help in the placement and reconfiguration of Virtual Network Functions (VNFs) deployments. Both applications can

be allocated in the area of automated synthesis of formal models, and in the wider context of Formal Methods for Industrial Critical Systems (FMICS). The term FMICS and the FMICS Working Group<sup>1</sup> is the main link between the first authors of the paper and Stefania Gnesi. Stefania was one of the founders and chair of FMICS WG and we have collaborated with her in the organization of the annual FMICS workshops and special publications. A relevant collaborative work co-led by Stefania is the book *Formal Methods for Industrial Critical Systems: A Survey of Applications* [11]. Actually, the SDN and VFV technologies addressed in this chapter are the evolution of the active networks paradigm that we presented as a chapter in Stefania's book in 2013 [10].

The paper is organized as follows. Section 2 introduces the 5G architecture and the relevant concept of network slicing. Section 3 provides a description of the SDN technology and our proposal to use the formal language Alloy [16] as the modelling language to support the synthesis of valid models from a given set of requirements. In Sect. 4, we present Network Function Virtualization as support technology of network slicing, and we propose combining model-based testing and runtime verification to solve the problem of placement and reconfiguration of VNFs in the network. Finally, in Sects. 5 and 6 we review the related work and provide some conclusions.

## 2 Background on 5G Networks

### 2.1 Architecture of the Network

One of the objectives of 5G networks is to support network operators to rapidly and flexibly deploy new services in order to meet customers' and verticals' needs. In this context, verticals refers to industrial sectors such as transport, media or manufacturing, whose digitalization and innovation relies on services with specific and different requirements. The 5G network architecture [9] aims to address these challenges using SDN and NFV as technical enablers. Figure 1 shows an overview of the network architecture. As can be seen, it is built on the three main domains that are also present in the previous 4G technology: radio access network (composed of the user equipment and the gNB access nodes), the transport network (a logical connection thanks to switches, aggregation points and communication links) and the core network. The main differences with respect to 4G networks are the technologies in each domain and the deployment of the services as part of the network (AF component in the figure). In the radio part, the new standard is called 5G NR (5G new radio) and offers features like lower latency and higher capacity than 4G networks. In the core network, the functionality is implemented with a number of software modules and interfaces following the standards to create a 5G core. It is expected that these 5G core VNFs will be deployed as VNFs in a central cloud. In the transport network, the switches are replaced with programmable OpenFlow switches, and the whole network is defined with apps running on top of the SDN controller.

<sup>1</sup> <http://fmics.inria.fr/>.

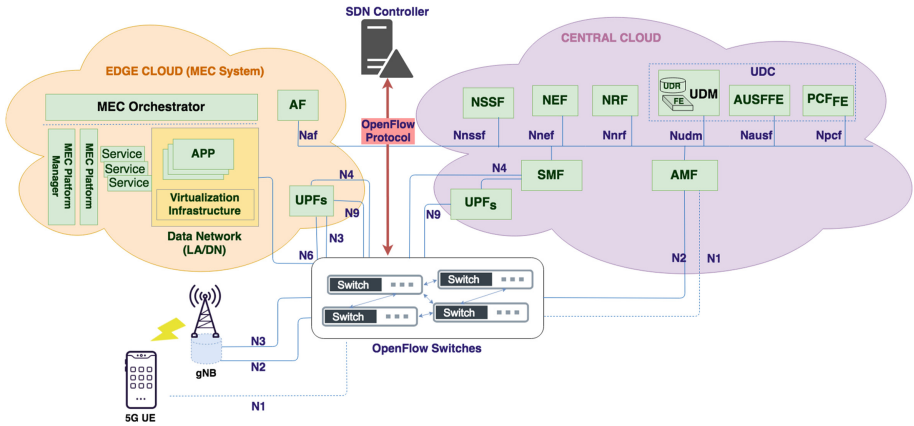


Fig. 1. 5G architecture

From the service provider perspective, the main novelty is the integration of the software to implement the service as part of the network thanks to the deployment of their own VNFs jointly with the 5G core VNFs and other network-oriented VNFs (e.g. caches and firewalls) in the same cloud. *Multi-access Edge Computing* (MEC) technology appears in order to have a computing platform within the RAN and in close proximity of users. MEC provides features like low latency, high bandwidth and proximity; sometimes MEC are called *Point of Presence* (PoP), which resources are shared by multiple VNFs (virtual machines installed on top). Thanks to the MEC location at the mobile edge, routing data that previously had to be sent to the core of the network can be eliminated and achieve a very low latency.

## 2.2 Slices for Verticals

Network slicing allows *Mobile Network Operators* (MNOs) to manage multiple virtual networks using a common shared physical infrastructure. These virtual networks enable a virtual partition of the RAN (Radio Access Network), the core network and the switching and aggregation network. Roughly speaking, Fig. 1 represents one slice, and a second slice could be created by assigning part of the RAN resources and adding more VNFs in the Edge cloud and the Central cloud to provide functionalities of 5G core and services. Each virtual network is created to provide a specific service with specific requirements that usually fit three profiles:

- **enhanced Mobile Broadband** (eMBB). It has the purpose of addressing the traffic demand that increases on a daily basis due to the number of users and new applications with increasingly demanding traffic requirements. Use cases related to multi-media content and data that are a very high throughput requirement (e.g. augmented reality or 4k video streaming).

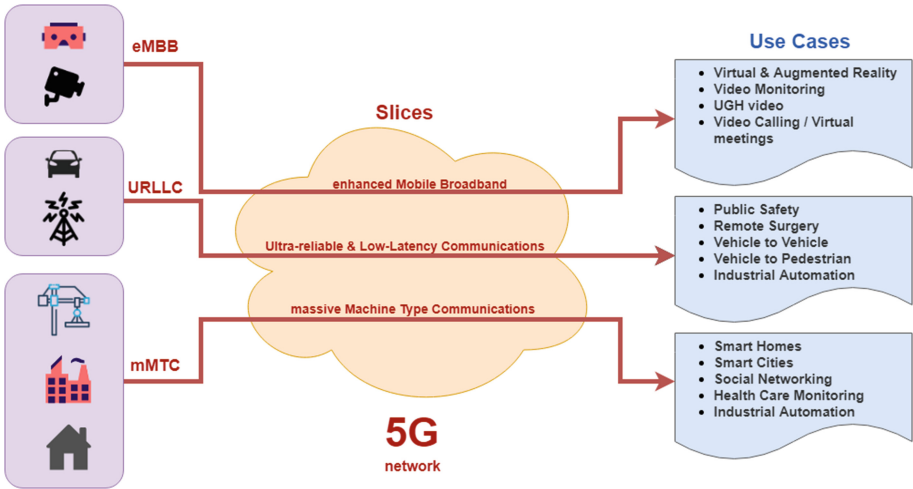


Fig. 2. 5G usage scenarios

- **Ultra Reliable Low Latency Communications (URLLC)**. It is oriented to low latency and high reliability transmissions, usually with low packet size. Use cases related to critical applications (e.g. remote surgery or connected cars).
- **massive Machine Type Communications (mMTC)**. It is oriented to a very large number of connected devices, usually with a low data transmission and non-delay sensitive data. Use cases related to the Internet of Things, where devices are required to have a long battery life (e.g. smart meters or sensor networks).

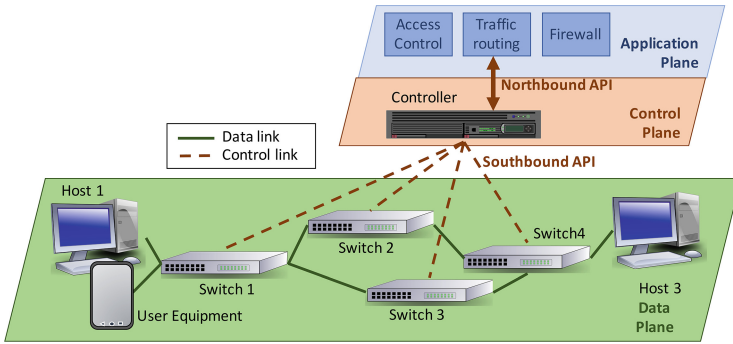
Figure 2 shows some usage scenarios where virtual networks mentioned above, known as *Network Slices*, help to provide a service that depends on use case requirements.

In the next sections, we present more details of the SDN and NFV technologies before describing the use of formal methods for both topics.

### 3 Software Defined Networks

#### 3.1 The Technology

Software Defined Network [26,32] (SDN) is a new paradigm for deploying highly programmable and flexible communication networks. To this end, the control and data planes of the network are clearly separated. In addition, from the logical point of view, the control plane is a single entity that has a global view of the network at each time instant and can modify the data plane to achieve specific goals. Figure 3 shows the high-level architecture of a SDN:



**Fig. 3.** Basic SDN architecture

- The **data plane** includes simple and programmable forwarding devices (switches) that route the traffic. To do this, each switch has a *routing table* with complex rules that specify the outgoing path for each input packet.
- The **control plane** comprises the controllers that carry out the configuration and management of the network. From the logical point of view, the control is centralized in an entity that has a global view of the network. However, the control plane can be implemented as a distributed set of controllers that communicate with each other using the east/westbound interfaces. The control plane offers interfaces to communicate with the other two layers. The southbound is used to control the data plane devices; for instance, to install the rules in the routing tables. There are different protocols that can be used in the southbound; currently the most popular is OpenFlow [23]. The northbound is used to communicate the controller with the applications. Usually, the controller provides its own API for applications. NOX [12] and POX [27] are two of the most used OpenFlow controllers.
- The **application plane** consists of a pool of applications that specify high-level management policies, such as routing, security or monitoring. SDN applications are dynamic in the sense that they can command the installation of new rules based on the state of the network

Thus, an SDN is a complex concurrent event-driven distributed system whose behavior is dynamically defined by the network applications. The separation of planes and the dependency on software introduces new challenges from the reliability point of view, such as the interaction of distributed controllers that must act as a single entity, or the dynamic update of the routing tables that lead to changes in the data plane topology. In addition, the adoption of SDN as an enabling technology of 5G networks introduces more challenging issues. For instance, 5G networks are characterized by the deployment of network slices for specific services or verticals. In this case, it is important to ensure not only the isolation of the slices but also to verify that these slices can support specific services with a predefined Quality of Service (QoS) and Quality of Experience (QoE). In the recent years, formal methods have used different approaches to

model and analyze SDNs [2, 4, 17, 21]. Most of these works focus on verifying network invariants, such as the absence of loops or host reachability. Section 5 summarizes these related works. Below, we propose a novel application of formal methods for SDN.

### 3.2 Formal Models of SDN Systems for Reachability Analysis

In this section, we propose an approach to generate valid network topologies (including the data and control plane) whose evolution over time fulfills some desired properties. To this end, we generate a meta-model of an SDN that includes the switches and the controller but also the hosts, the packets flowing through the data and control planes, and the logic behind the controller, that is, the SDN application.

To this end, we use ALLOY [16, 24], a modelling and specification language and also a tool that generates model instances satisfying a set of requirements. ALLOY is a declarative language based on sets and set relations. In addition, it uses first order relational logic to describe properties and refine the models. Although, internally, the core of ALLOY tool uses a theorem prover, from the user perspective it is a completely automatic tool (similar to a model checker) that generates models that are correct w.r.t. the specification. All these characteristics make ALLOY suitable for modelling and analyzing structurally complex systems such as SDN, which can present complex and varying relations between the different network elements. The price to pay is that ALLOY models are bounded in size, i.e., it is not possible to analyze models of arbitrarily large size. Even though this is an important restriction, in practice, small models are usually sufficient to detect errors in the system design. In order to achieve our goal, we have to follow three steps. First, we have to implement the model of the static structure of a SDN; then, we have to define the dynamic behavior of the network elements; and finally, we must define the requirements and configuration of interest and run the ALLOY tool to generate valid SDN topologies. The rest of the section describes these steps in detail.

#### Modelling the Static Structure of an SDN

The static model of an SDN defines the elements of an SDN and the (static) relations between them. The model must abstract low-level details of an SDN so that ALLOY can run the analysis and return different topologies or configurations of the network. The set of actors constitutes the ALLOY metal-model, Fig. 4 shows a graphical description, and Fig. 5 shows the corresponding ALLOY code. The main actors are *hosts* and *switches* in the data plane, and the controllers in the control plane. These three elements are abstracted as network nodes that contain ports to connect them with other nodes using port-to-port (bidirectional) links. Although it is not explicitly reflected in the meta-model, each switch always has a specific link and a port that connects it with the controller. This connection is mainly used to configure the switches. The data transmitted between nodes are called *packets*. We define two types of packets: control packets include control plane information, such as new rules that must be installed in a specific switch,

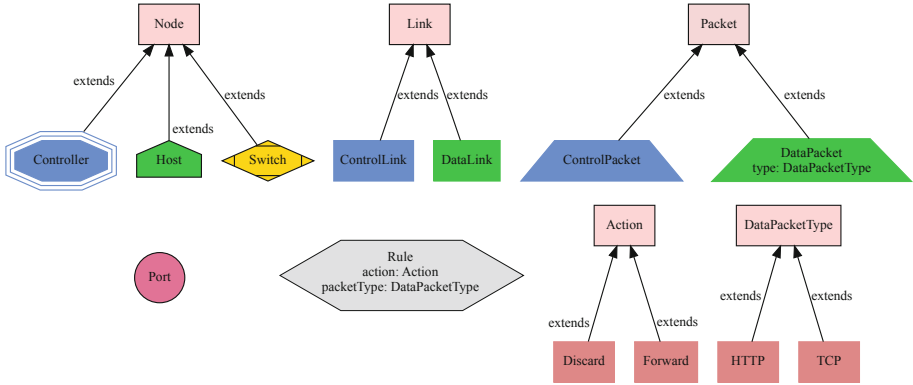


Fig. 4. Meta-model of the SDN model

or a request to know how to process a data packet. Data packets encapsulate information that must be transmitted from one host to another. Data packets also contain the source and destination hosts, their type, and their current position in the network.

Switches contain forwarding tables with *rules* that specify how to route data packets. In order to simplify the model, a rule includes the type of data packet (e.g. HTTP or FTP) and the input and output ports. The meaning of each rule is as follows: if a data packet of a particular type arrives at port  $iPort$ , it must be forwarded through port  $oPort$ . In addition, it is also possible to define rules that discard incoming data packets. When a switch has no rule to deal with a data packet, it sends a request to the controller in order to know how to process the packet. Finally, the controller can also send new rules to switches to update the routing tables.

Figure 5 shows how all these elements are defined using basic ALLOY constructors: *abstract signatures*, *signatures*, and *relations*. Signatures are sets whose elements are called *atoms*. Abstract signatures cannot have their own elements; they can only have them through their extensions. Relations are sets of tuples of the same arity. They are always defined in the context of a signature, which is the type of the first element of the tuples. For instance, the signature `Rule` has a relation `iPort` that relates each rule with the input port of the data packet. By default, the multiplicity of relations is one. In the example, each rule only has one `iPort`, but ALLOY's relations support other multiplicities such as `lone` (e.g. a `Rule` has at most one `oPort`), `some` (a `Node` is `connected` at least to another `Node`) and `set` (e.g. a `Switch` has zero or more `Rule` in its forwarding `table`).

ALLOY allows us to define both the static and dynamic behavior of an SDN. To this end, we include a special signature, called `Time`, whose atoms explicitly represent time instants. For example, we can specify that the `iBuffer` and `oBuffer` of a `Host` have zero or more `DataPackets` in each time instant.

At this point, ALLOY can generate model instances that are still far from being structurally correct. We need to add some constraints to the model, called



```

open util/ordering[Time]
sig Time{}
sig Port{}

abstract sig Link{
  p1,p2: Port
}
sig CtrLink extends Link{}
sig DataLink extends Link{}

abstract sig DataPacketT{}
one sig TCP extends DataPacketT{}
one sig HTTP extends DataPacketT{}

abstract sig Action{}
one sig Forward extends Action{}
one sig Discard extends Action{}

sig Rule{
  packetType:DataPacketT,
  iPort: Port,
  action: Action,
  oPort : lone Port
}
}

abstract sig Packet{
  position: Port lone -> Time
}
sig DataPacket extends Packet{
  type: DataPacketT,
  src,dest:Host
}
sig CtrPacket extends Packet{
  newRule: lone Rule,
  request: lone DataPacket
}

abstract sig Node{
  ports: some Port,
  connected: some Node
}
one sig Controller extends Node{}
sig Host extends Node{
  iBuffer: DataPacket set -> Time,
  oBuffer: DataPacket set -> Time
}
sig Switch extends Node{
  table: Rule set -> Time
}
}

```

Fig. 5. SDN signatures and relations in ALLOY

*facts* in ALLOY, to define, for instance, how the relations are constructed. In total, we have added 32 facts to the SDN model. Figure 6 shows some that define how links, ports and nodes are related.

### Modelling the Dynamic Behavior of an SDN

The second step is to describe how the SDN evolves over time; for instance, how a *DataPacket* can be transmitted from the source to the destination *Host*, or how the forwarding table of a *Switch* is modified by the *Controller*. These actions, or system transitions, are specified in ALLOY with predicates. Each predicate has two input parameters  $t$  and  $t'$  that denote the time instant before and after executing the predicate, which are used to clearly state the pre- and post-conditions needed to execute the predicate, and the so-called *frame conditions* that establish the parts of the model that remain unchanged during the predicate execution. Figure 7 shows two predicates that define, respectively, how a *Host* sends and receives a *DataPacket*, and the definition of the frame conditions.

We can similarly specify the actions associated to the switches and the controller. For instance, we have defined predicates to describe how a switch forwards a data packet applying a rule installed in its forwarding table, or how a new rule is installed or updated when the controller sends a command to a switch. In addition, the actions of the controller can be more elaborate. We can describe not only how the controller receives a request from a switch but also the logic or decision making process associated to an SDN application.

### Generating Valid Model Instances

The final step consists of using ALLOY analysis to generate correct model instances. These instances will differ in the network topology; that is, how the

```

fact {
  //1-the ending ports of any link are different
  all l:Link | l.p1!=l.p2
  //2- each port belongs to a node
  all p:Port | one node[p]
  //3-each port belongs at most to a link
  all p:Port | lone link[p]
  // 4- The ports of each link belong to different nodes
  all l:Link | node[l.p1]!=node[l.p2]
  //5-connected is well defined
  all n:Node | n.connected = {m:Node-Controller | some l:Link |
    node[l.(p1+p2)] = n+m}
  //6-Control links connect switches and Controller
  all l:Link | l in CtrLink implies one node[l.(p1+p2)] & Controller and
    one node[l.(p1+p2)] & Switch
  //7-Data links connect two switches or a switch and a host
  all l:Link | l in DataLink implies some node[l.(p1+p2)] & Switch and
    Controller not in node[l.(p1+p2)]
  //8-all controller links are control links
  nodeLinks[Controller] in CtrLink
  //9-the controller has exactly a link to each switch
  all s:Switch | one nodeLinks[Controller] & nodeLinks[s]
}

```

Fig. 6. Examples of facts

```

pred sendPacket(t,t':Time, h:Host, pack:DataPacket){
  //pre
  some pack & h.oBuffer.t
  //post
  some p':remotePort[h.ports] | pack.position.t'=p'
  h.oBuffer.t' = h.oBuffer.t - pack
  //frame
  tablesUnmodifiedExc[none,t,t'] and packetsUnmodifiedExc[pack,t,t']
  oBuffersUnmodifiedExc[h,t,t'] and iBuffersUnmodifiedExc[none,t,t']
}
pred receivePacket(t,t':Time,h:Host,pack:DataPacket){
  //pre
  some (pack.position.t & h.ports)
  //post
  h.iBuffer.t' = h.iBuffer.t + pack
  pack.position.t' = none
  //frame
  tablesUnmodifiedExc[none,t,t'] and packetsUnmodifiedExc[pack,t,t']
  oBuffersUnmodifiedExc[none,t,t'] and iBuffersUnmodifiedExc[h,t,t']
}
pred packetsUnmodifiedExc(pp:set Packet, t,t':Time){
  all pk:Packet-pp | pk.position.t = pk.position.t'
}
pred oBuffersUnmodifiedExc(hh:set Host, t,t':Time){
  all h:Host-hh | h.oBuffer.t = h.oBuffer.t'
}
pred iBuffersUnmodifiedExc(hh:set Host, t,t':Time){
  all h:Host-hh | h.iBuffer.t = h.iBuffer.t'
}
pred TablesUnmodifiedExc(ss: set Switch, t,t':Time){
  all s:Switch - ss | s.table.t = s.table.t'
}

```

Fig. 7. Predicates to send and receive data packets on hosts

network nodes are interconnected, and in their dynamic behaviors, for instance how packets flow through the network or how the switches' forwarding tables are updated. The ALLOY tool can run predicates and check assertions. In the first case, if the predicate is consistent, the tool generates model instances that satisfy the constraints (facts) and the predicate. In the second case, ALLOY looks for counterexamples that satisfy the model specification but not the assertion. In both cases, the ALLOY model is transformed into a set of boolean formulae that are analyzed using a SAT solver. Our objective is to produce valid SDN network topologies that can evolve over time taking into account the applications governing the SDN controller. The automatically generated network topologies and configurations can be used to test the SDN applications in real or simulated environments. To this end, we define a predicate that specifies the initial configuration of the SDN network, and the possible system transitions, which are given as non-deterministic calls to the predicates defining the dynamic behavior of the SDN. The non-deterministic choice is implemented using the disjunction logic and thus, for each time instant, only one system transition can be executed. Figure 8 shows a snapshot of a valid model instance in two different time instants (Time\$0 and Time\$5). To simplify the representation, we only show the data plane. Observe that in both time instants, the network topology is the same; that is, the interconnection of switches and host is the same, with the same links and ports. However, we observe changes in the relations that can evolve over time. For example, at Time\$0 both data packets are in the oBuffer of Host1, while in Time\$5 DataPacket1 has reached its destination and DataPacket0 is in an intermediate switch. At this point, the Switch1 does not have a rule in its table that can make the system evolve. In consequence, it has to send a request to the controller to ask how to forward the packet. The subsequent actions will depend on the SDN applications modelled.

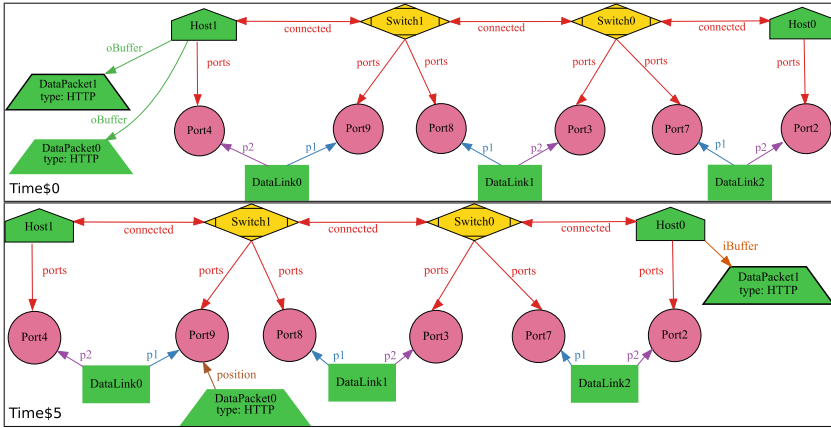
Assertion checking in ALLOY is also useful to determine if the SDN applications are correct. For instance, if the SDN application has to discard all FTP packets transmitted from a specific host, we can check if all model instances satisfy this requirement. Otherwise, ALLOY will return a counterexample that will show why some packets are not correctly discarded.

## 4 Network Function Virtualization

### 4.1 The Technology

In 5G networks, the concept of NFV is especially important since it can entail a significant transformation for this network, reducing cost or increasing flexibility, although the most important change that NFV introduces is the possibility of providing different kinds of services and requirements on a common shared physical network through network slicing (see Sect. 2.2).

The reference architecture for NFV, represented in Fig. 9, has been proposed by ETSI in [6]. The deployment and reconfiguration of the VNFs in a cloud environment is an open challenge. This task is carried out by the Management and



**Fig. 8.** Evolution over time of data packets

Orchestration entity (MANO). The left part of the figure is composed of *Operations Support System (OSS)/Business Support System (BSS)*, *Element Manager (EM) + VNF* and by finally, *NFV Infrastructure (NFVI)* is composed of virtual/hardware computing, storage and network (the hardware supporting the Edge and Central clouds in Fig. 9). OSS/BSS are components that allow monitoring, controlling and managing different kinds of network services. EM provides network management of the virtualized and physical network elements. The VNF is an implementation of a network function that can be deployed on NFVI. The right part is composed of *Management and Orchestration (MANO)* layer, differentiating between *NFV Orchestration (NFVO) + VNF Manager (VNFM)* and *Virtualized Infrastructure Manager (VIM)*. NFVO is responsible for orchestration and management of NFVI, software resources and realizing network services on NFVI. VNFM is responsible for control, management and monitorization of the VNF life cycle. It also controls EM. The VIM is the Virtualized Infrastructure Manager that, in most real deployments, is the well-known OpenStack software.

**An Example.** We illustrate the NFV architecture with an example. Figure 10 shows how a service is deployed in a network slice. We assume that the MNO offers a simple slice to deploy a video on demand service for mobile users. The slice includes network components, such as an instance of a 5G core, and some service-oriented VNFs, such as a video server and the cache function. The orchestrator (the MANO) addresses the following four phases to properly configure, deploy and terminate the service.

**Phase 1: Network Service Descriptor Processing.** The MANO “orchestrator” processes the information necessary to deploy a network slice oriented to a specific service (Netflix, in this case). This information includes the executable code of VNF (virtualized like a container, virtual machine, etc.) and the



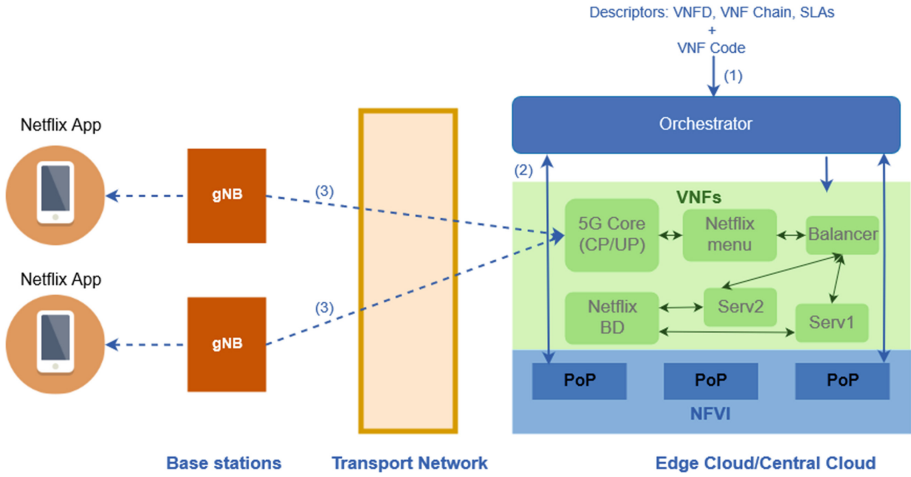


Fig. 10. Example of a slice for video on demand

```

vnfd-catalog:
  vnfd:
  - connection-point:
    - name: eth0
      type: VPORT
    ...
  description: ...
  mgmt-interface: ...
  name: slice_VoD_vnfd
  vdu:
  - count: 1
    id: vdu1
    image: UbuntuVoD
    interface:
    - external-connection-point-ref: eth0
      ...
    virtual-interface:
      ...
    monitoring-param:
    - id: metric_vdu1_cpu
      nfvi-metric: cpu_utilization
    name: slice_VoD_vnfd-VM
    vm-flavor:
      memory-mb: 2048
      storage-gb: 100
      vcpu-count: 2
    monitoring-param:
    - id: metric_vim_vnf1_cpu
      name: metric_vim_vnf1_cpu
      aggregation-type: AVERAGE
    vdu-monitoring-param:
      vdu-ref: vdu1
      vdu-monitoring-param-ref: metric_vdu1_cpu
    ...
  
```

Listing 1.1. Excerpt of a VNFD

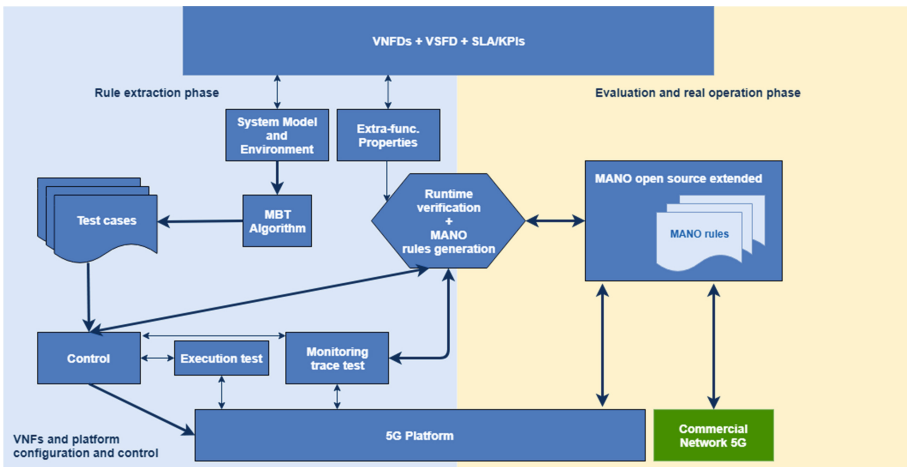
```

- SNSSAI-identifier:
  slice-service-type: eMBB/URLLC/mMTC
    
```

**Listing 1.2.** Definition of the slice service type

**Phase 2: Network Service Deployment.** The MANO performs the deployment, inter-connection and configuration of the chain of VNFs interacting with the points of presence of the operator computational infrastructure, commonly known as *Network Functions Virtualization Infrastructure* (NFVI). The deployment consists of locating each VNF at a suitable point of presence. The configuration implies the allocation of resources (CPU, RAM, disk) and the inter-connection with other VNFs and/or physical elements. Then, the MANO must apply some optimization algorithm to identify how many resources should be allocated to achieve the performance and quality of service given by the SLA.

**Phase 3: Network Service Execution and Re-configuration.** In this phase, the VNFs implementing the network and the service functionality are running, and the final service users start using the service uninterruptedly for days, weeks or months. Some examples of services are the distribution of high-resolution video (such as the popular service Netflix shown in Fig. 10), private communications for security forces, an augmented reality to offer sightseeing activities, remote control for critical infrastructures, etc. In this phase, the number of users and their location may vary, producing a changing network environment. Moreover, the network and computational resources can be modified due to the deployment or elimination of other services. The orchestrator must monitor that VNFs fulfill the SLAs so that users receive the expected quality (KPIs or QPIs). In addition, the orchestrator must re-locate or re-configure VNFs, if necessary.



**Fig. 11.** Overall approach.

Currently, the algorithms that perform these tasks still admit a quite margin of improvement.

**Phase 4: *Network Service Termination*.** This phase is not represented in Fig. 10. It is devoted to the release of the resources previously allocated to the service, as well as the elimination of VNFs images. These actions are carried out when users are not expected to be connected for a long period of time, and thus, the elimination and subsequent creation of resources is possible.

## 4.2 Model Based Testing and Runtime Verification to Support Flexible Placement and Reconfiguration of VNFs

We now face the challenge of automatically generating useful information to help the orchestrator decide about the deployment, configuration and re-configuration of VNFs. To do this, it is necessary to use the description of the VNFs and the service level agreements for each service (SLA) to predict the suitable deployment that satisfies the SLA. This problem has been previously addressed with different estimation tools [25, 29, 30, 34]. However, in these previous approaches, the main focus is the use of computational resources. They do not consider the impact of a realistic 5G network on the final QoS perceived by the users.

Our goal is to develop a novel learning method to specialize the orchestrator taking the whole end-to-end network into account; that is, considering the users and the communication component. We propose using different formal methods to generate and test the orchestrator decisions, making use of a realistic end-to-end 5G network to run the VNFs. In particular, we combine formal methods *model-based testing*, *model checking* and *runtime verification* to carry out a runtime analysis of extra-functional properties (related to time and resource utilization) which will be evaluated over event sequences which correspond to the executions of network service.

Figure 11 shows our approach to generate the so-called *book of rules* of a MANO in order to manage a specific service according to the requirements and performance specified in the SLA. The approach has two well differentiated phases. The first one is devoted to the extraction of the MANO rules using model-based testing and run-time verification techniques. The second phase focuses on the validation of the rules over an emulated 5G environment. Both phases have the VNF and service descriptors (VNFDs and VSFDs) as input, as well as the SLA. Both phases can be iterated several times in order to gradually refine the rules generated.

The first phase is shown on the left part of Fig. 11. The VNFDs and VSFDs are transformed into a model of the service combined with a non-deterministic model of the MANO that includes a wide variety of rules to be applied to each network scenario. With this model, we automatically generate test cases that show different management rules for different network scenarios. The VNFs pass these test cases in a controlled environment, a testing platform for 5G, where they can be monitored. We use runtime verification techniques to determine whether



the service and the VNFs satisfy the SLA. To do this, SLAs are translated into a set of extra-functional properties (the runtime monitors) that evaluate whether the execution of each test case matches the desired SLA. The (non-)correct test cases help us to obtain new management rules for the MANO satisfying the extra-functional properties.

The objective of the second phase (shown on the right part of Fig. 11) is to validate the synthesized rules in a production environment. To this end, the rules are installed on a real MANO that manages and orchestrates the 5G testing platform (and that can even handle a commercial 5G network). Again, using the runtime verification engine, we can check the suitability of the synthesized rules during a normal operation of the service.

We propose running multiple iterations of the approach. In each iteration, we will refine the NFV and MANO models making use of the results of the previous iteration. These refined models can be used to extract new test cases that produce more precise rules. In the following section, we present some preliminary models of the NFVs and MANO.

### 4.3 Modeling the MANO

We assume that the MANO model consists of a number of MANO (sub-)models that execute concurrently. Each sub-model manages a unique service deployed on different points of presence (PoPs) to which users are connected. In consequence, in this section, we focus on the description of one of these sub-models. In the following text, to simplify the presentation, we simply call it MANO model.

The MANO model is a state machine that responds to events, provided by the infrastructure platform, executing functions to preserve the required SLA. Thus, the MANO and the infrastructure intensively communicate with each other over time. In addition, a complementary functionality of the MANO model is to periodically inspect the state of the infrastructure to carry out reconfiguration actions, if needed, even though no events are fired.

Figure 12 contains a prototype implementation of our proposal using Uppaal timed automata. State `s0` is the initial state of the monitor. At this state, the automaton may receive events such as `alarmCPU90` and `newUser` from the infrastructure through different synchronization channels. The first event occurs when the platform detects that some instance is reaching the 90% in the CPU usage. The second one is a notification that a new user has connected to the service. The monitor responds to the first event by transiting to state `s2`. During the transition, the identifier of the PoP which has provoked the alarm is recorded in variable `eid`. The monitor also searches for a new PoP, with more resources, that can hold a new instance of the service, if needed. From `s2`, the automaton can go to states `s3`, `s4`, `s6` and `NO_RESOURCES`. This last state is an error which should never be reached. The rest of the states represent non-exclusive alternatives for the monitor: to increment the CPU resources in the PoP where the instance which fired the alarm is located, to deploy a new service instance and balance the users, to migrate the instance to a different PoP with enough CPU resources.

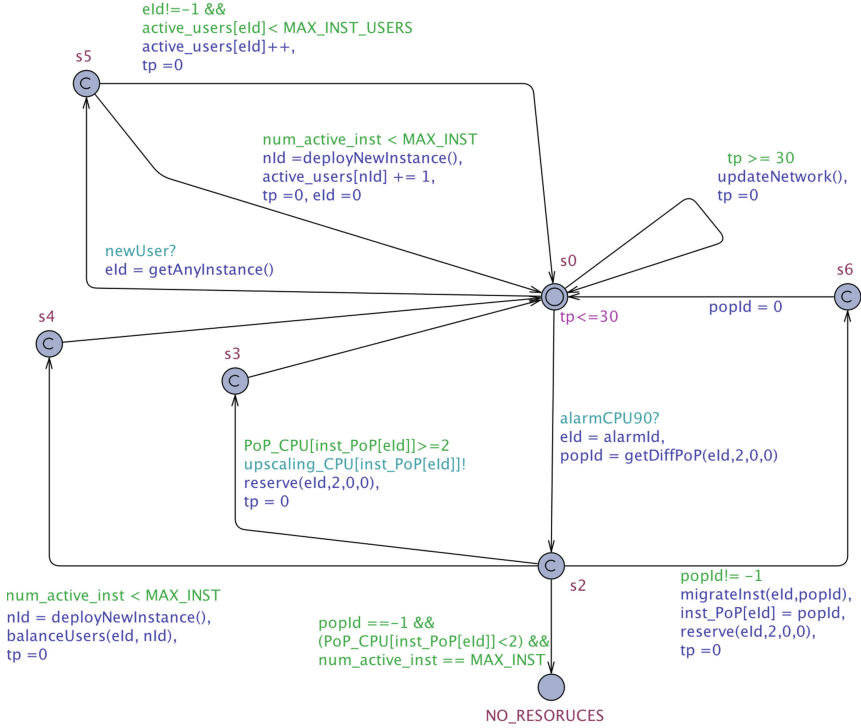


Fig. 12. Uppaal MANO model

Similarly, the monitor responds to the `newUser` event transiting to state `s5` and finding an instance which can hold it. From `s5`, the automaton can jump to the initial state through two different transitions: assigning the new user to an existing service instance or deploying a new service instance for the user. As in the previous case, if both transitions are enabled, the automaton may select any of them in a non-deterministic way.

Finally, observe that the automaton has a clock variable `tp` that is used to periodically check the state of the network and update it, if necessary. Currently, we use a bi-dimensional array with the network parameters of interest that contain the network state at three ordered previous time instants. This information may be used, for instance, to discover when a user may have disconnected to release its resources.

We have not included transitions that deal with alarms due to the RAM or HDD usage in the model of Fig. 12 to simplify the automaton. In addition, it is worth noting that the model is parametric w.r.t. a number of constants that have to be calibrated such as the thresholds to fire alarms, the maximum number of service instances and users (`MAX_INST`, `MAX_INST_USERS`) and so on.

The concurrent execution of the MANO model and the infrastructure produces a set of traces (sequence of infrastructure states) that constitute the test

cases to be analyzed against the SLA. Figure 13 contains an example of a possible infrastructure execution which can be synchronized with the monitor of Fig. 12 to produce test cases.

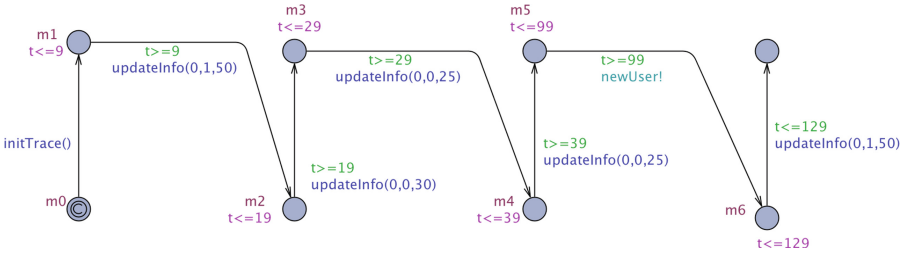


Fig. 13. An infrastructure execution

The execution shows changes in the network at certain time instants. For example, at time instant  $t=9$  the platform is already initialized and the values of HDD and RAM usage are updated (the second and third parameter of function `updateInfo`). At time instant  $t=99$ , a new user is connected and the infrastructure sends event `newUser` to the monitor.

As previously described, the analysis of test cases may be used to iteratively improve the monitor rules (the transitions in the automaton of Fig. 12).

The number of different traces generated this way is very large due to the non-deterministic character of the monitor, and the range of values of the network variables. A way of pruning these traces is to use properties (described, for instance, in some temporal logic) to discard some non-interesting behaviors. For instance, we could add the TCTL property “ $A[](\text{not monitor.s3})$ ” to the monitor to generate test cases in which the monitor has carried out at least an `upscaling_CPU` task. The Uppaal model checker tries to check if no trace is eventually at state `s3`. The counterexamples for this property are precisely the test cases of interest wrt the property specified.

Table 1. Problems faced by SDN verification tools

Problems faced	Related work
Networks with multiple protocols	[17, 20]
Networks with multiple SDN slices	[1, 17, 20]
Networks with multiple OpenFlow controllers/domains	[1]
Scalability	[4]
Unbounded space of input packets and control messages	[4, 21]
Flow tables updating at runtime in switches	[2, 5, 18, 21]
Interleaved processing of packets/events	[2, 4, 5, 13, 21]
Topology changes (robustness during an execution)	[2]

## 5 Related Work

The formal analysis of SDNs is a challenging task as they are distributed and open event-driven systems. Given the separation of data and control planes and the existence of a new application plane that provides the control intelligence, most works focus on analyzing SDN problems from the perspective of one of these planes, without considering the relation between them. In [19], we reviewed the state of the art of formal methods and tools to support SDN. Table 1 shows a list of problems addressed and related works. These proposals differ not only regarding the network plane and the problem faced, but they also use different formal methods, such as model checking [1, 4, 17, 21], theorem proving [2, 5, 20], runtime analysis [18] or traditional network debugging [13].

The efficient management and orchestration of VNFs in the context of 5G networks is an interesting and challenging problem that can be addressed from different perspectives.

Currently, some proposals focus on predicting the performance behavior of VNF chains (services deployed interconnecting VNFs) deployed in the cloud without considering the role of the mobile network. However, the network and the service users are an important part of the environment that stimulates and interacts with the VNFs, and thus, they must be taken into account to predict the service performance in terms of extra-functional properties, Service Level Agreements (SLAs), Key Performance Indicators (KPIs) and Quality Performance Indicators (QPIs).

Peuster and Karl [29] proposed a methodology to characterize the performance of VNF and VNF chains prior to service deployment that can be part of a DevOps methodology. To this end, the authors execute the VNFs in different emulated network configurations and monitor how different performance parameters evolve. The paper includes an evaluation of a profiler prototype built on the emulation platform MeDICINE, which is based on an extension of Mininet that can execute production-ready VNFs (given as Dockers containers) in user-defined network topologies. The prototype can emulate the effect of the network and other services that compete for the resources. However, the emulation is limited by the capacity of the host machine that runs Mininet, which is far from emulating a real 5G network. The author states that the profile of the VNF can be used to improve the decisions carried out by the MANO, but there is no insight of how to transform the profile into MANO rules.

Gym [30] is other framework for VNF profiling. In this case, the VNF is tested under different resource configurations of the infrastructure, which is mainly composed of servers where the VNFs runs. However, the infrastructure lacks the components of a mobile network. The authors' objective is to use the framework to build testbeds for NFVs and services extending the framework with new components.

The characterization of VNFs performance has also been addressed from the analytical point of view. For instance, the tool Probius [25] aims to detect abnormal behaviors of NFVs due to performance uncertainties. Probius automatically generates all possible service chains with the given VNFs, collects and analyzes

performance-related features of each chain, and analyzes performance problems through anomaly detection and graph-based behavior analysis, and is able to point out the reasons of the VNFs performance issues.

The project 5GTANGO [34] proposed a testing approach for VNFs based on TTCN-3 test cases that can be manually or automatically generated using model-based testing techniques. However, the authors do not discuss which entities are included in the model, or what requirements guide the test generation algorithms.

In [25, 29, 34], the VNFs, the services and the test cases run in an emulated infrastructure that cannot properly represent a 5G network.

Formal methods have been also used to verify VNFs and VNF chains against reachability and safety properties in order to determine whether services are interfering, are isolated, or are accessed by unauthorized users. In [28] and [33], the analysis of properties is based on SMT solvers, such as Z3, combined with static analysis and symbolic model checking. These approaches accept a logic formula as input and find the values (if any) that make the formula satisfiable. The main limitation of these approaches is the transformation of the VNF code into a model in the solver's input language. This task is not trivial and is error prone. To minimize this problem, in [22], a tool to automatically extract the VNF model from its code is proposed. The approach is limited to VNFs implemented using a set of Java libraries to facilitate the coding task.

## 6 Conclusions

The increasing presence of software in mobile communication networks, like 5G networks, requires the use of rigorous methods to ensure the correct behavior. The formal methods community can find here new challenges to demonstrate the applicability of well-known techniques for modelling and automatic analysis. We have introduced work in progress in the general topic of automated generation and specialization of formal models that can help the deployment of the SDN and NFV parts of the network. We use two different modelling languages because the objectives for each domain are different. The use of Alloy to model SDN overpasses the state space explosion problem of model checking when checking for valid network configurations. The use of SAT solvers in this context has been demonstrated with a middle size network; however we still need to confirm the feasibility of the approach with more complex configurations. The use of UPPAAL to model the NFV part is still a proof concept and more work will be done to produce experimental results. One future work is to analyze whether a single formal method could support the two domains considered in the 5G network. We are currently working on implementing some of these methods to be validated in realistic research networks in the context of the European H2020 research projects EuWireless and 5GENESIS.

## References

1. Al-Shaer, E., Al-Hajj, S.: FlowChecker: configuration analysis and verification of federated OpenFlow infrastructures. In: Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration, SafeConfig 2010, pp. 37–44. ACM, New York, October 2010. <https://doi.org/10.1145/1866898.1866905>
2. Ball, T., et al.: VeriCon: towards verifying controller programs in software-defined networks. SIGPLAN Not. **49**(6), 282–293 (2014). <https://doi.org/10.1145/2666356.2594317>
3. Bochmann, G., Rayner, D., West, C.H.: Some notes on the history of protocol engineering. Comput. Netw. **54**(18), 3197–3209 (2010). <https://doi.org/10.1016/j.comnet.2010.05.019>
4. Canini, M., Venzano, D., Perešini, P., Kostić, D., Rexford, J.: A NICE way to test OpenFlow applications. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2012), San Jose, CA, pp. 127–140. USENIX, April 2012
5. El-Hassany, A., Miserez, J., Bielik, P., Vanbever, L., Vechev, M.: SDNRacer: concurrency analysis for software-defined networks. SIGPLAN Not. **51**(6), 402–415 (2016). <https://doi.org/10.1145/2980983.2908124>
6. ETSI GS NFV: Network Functions Virtualization (NFV); Terminology for Main Concepts in NFV. Technical report ETSI GS NFV 003, European Telecommunications Standards Institute (ETSI), August 2018. v1.4.1
7. ETSI GS NFV-IFA: Network Functions Virtualization (NFV); Management and Orchestration; VNF Descriptor and Packaging Specification. Technical report ETSI GS NFV-IFA 011, European Telecommunications Standards Institute (ETSI), August 2018. v2.5.1
8. ETSI GS NFV-MAN: Network Functions Virtualization (NFV); Management and Orchestration. Technical report ETSI GS NFV-MAN 001, European Telecommunications Standards Institute (ETSI), December 2014. v1.1.1
9. ETSI TS 123 501: 5G; System Architecture for the 5G System. Technical report ETSI TS 123 501, European Telecommunications Standards Institute (ETSI), June 2018. v15.2.0
10. Gallardo, M.M., Martínez, J., Merino, P.: Applying formal methods to telecommunication services with active networks (2013)
11. Gnesi, S., Margaria, T.: Formal Methods for Industrial Critical Systems: A Survey of Applications, 1st edn. IEEE, Washington, D.C. (2013)
12. Gude, N., et al.: NOX: towards an operating system for networks. SIGCOMM Comput. Commun. Rev. **38**(3), 105–110 (2008). <https://doi.org/10.1145/1384609.1384625>. Tool <https://github.com/noxrepo/nox>
13. Handigol, N., Heller, B., Jeyakumar, V., Mazières, D., McKeown, N.: Where is the debugger for my software-defined network? In: Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks, HotSDN 2012, pp. 55–60. ACM, New York, August 2012. <https://doi.org/10.1145/2342441.2342453>
14. Holzmann, G.: The Spin Model Checker: Primer and Reference Manual, 1st edn. Addison-Wesley Professional, Boston (2003)
15. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice-Hall Inc., Upper Saddle River (1991)
16. Jackson, D.: Software Abstractions - Logic, Language, and Analysis. MIT Press (2006). <http://mitpress.mit.edu/catalog/item/default.asp?ftype=2&tid=10928>

17. Kazemian, P., Varghese, G., McKeown, N.: Header space analysis: static checking for networks. In: 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2012), Lombard, IL, pp. 113–126. USENIX, April 2012
18. Khurshid, A., Zhou, W., Caesar, M., Godfrey, P.B.: VeriFlow: verifying network-wide invariants in real time. *SIGCOMM Comput. Commun. Rev.* **42**(4), 467–472 (2012). <https://doi.org/10.1145/2377677.2377766>
19. Lavado, L., Panizo, L., Gallardo, M., Merino, P.: A characterisation of verification tools for software defined networks. *J. Reliable Intell. Environ.* **3**(3), 189–207 (2017). <https://doi.org/10.1007/s40860-017-0045-y>
20. Mai, H., Khurshid, A., Agarwal, R., Caesar, M., Godfrey, P.B., King, S.T.: Debugging the data plane with anteater. *SIGCOMM Comput. Commun. Rev.* **41**(4), 290–301 (2011). <https://doi.org/10.1145/2043164.2018470>
21. Majumdar, R., Tetali, S.D., Wang, Z.: Kuai: a model checker for software-defined networks. In: *Formal Methods in Computer-Aided Design (FMCAD)*, Lausanne, Switzerland, pp. 163–170. IEEE, October 2014. <https://doi.org/10.1109/FMCAD.2014.6987609>
22. Marchetto, G., Sisto, R., Virgilio, M., Yusupov, J.: A framework for user-friendly verification-oriented VNF modeling. In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 517–522, July 2017. <https://doi.org/10.1109/COMPSAC.2017.16>
23. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.* **38**(2), 69–74 (2008). <https://doi.org/10.1145/1355734.1355746>
24. Milicevic, A., Near, J.P., Kang, E., Jackson, D.: Alloy\*: a general-purpose higher-order relational constraint solver. *Formal Methods Syst. Des.* 1–32 (2017). <https://doi.org/10.1007/s10703-016-0267-2>
25. Nam, J., Seo, J., Shin, S.: Probius: automated approach for VNF and service chain analysis in software-defined NFV. In: *Proceedings of the Symposium on SDN Research, SOSR 2018*, pp. 14:1–14:13. ACM (2018). <https://doi.org/10.1145/3185467.3185495>
26. Nunes, B.A., Mendonca, M., Nguyen, X.N., Obraczka, K., Turletti, T.: A survey of software-defined networking: past, present, and future of programmable networks. *IEEE Commun. Surv. Tutorials* **16**(3), 1617–1634 (2014). <https://doi.org/10.1109/SURV.2014.012214.00180>
27. Open Networking Lab: POX (Python Network Controller) Wiki (2013). <https://openflow.stanford.edu/x/TYBr>
28. Panda, A., Lahav, O., Argyraki, K.J., Sagiv, M., Shenker, S.: Verifying isolation properties in the presence of middleboxes. *CoRR abs/1409.7687* (2014)
29. Peuster, M., Karl, H.: Understand your chains: towards performance profile-based network service management. In: *2016 Fifth European Workshop on Software-Defined Networks (EWSDN)*, pp. 7–12. IEEE Computer Society (2016). <https://doi.org/10.1109/EWSDN.2016.9>
30. Rosa, R.V., Bertoldo, C., Rothenberg, C.E.: Take your VNF to the gym: a testing framework for automated NFV performance benchmarking. *IEEE Commun. Mag.* **55**(9), 110–117 (2017). <https://doi.org/10.1109/MCOM.2017.1700127>
31. Rudin, H., West, C.H., Zafiropulo, P.: Automated protocol validation: one chain of development. *Comput. Netw.* (1976) **2**(4), 373–380 (1978)
32. Shenker, S., Casado, M., Koponen, T., McKeown, N., et al.: The future of networking, and the past of protocols. *Open Netw. Summit* **20** (2011)

33. Spinoso, S., Virgilio, M., John, W., Manzalini, A., Marchetto, G., Sisto, R.: Formal verification of virtual network function graphs in an SP-DevOps context. In: Dustdar, S., Leymann, F., Villari, M. (eds.) ESOCC 2015. LNCS, vol. 9306, pp. 253–262. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-24072-5\\_18](https://doi.org/10.1007/978-3-319-24072-5_18)
34. Zhao, M., et al.: Verification and validation framework for 5G network services and apps. In: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 321–326, November 2017. <https://doi.org/10.1109/NFV-SDN.2017.8169878>