MATHEMATICAL METHODS FOR ANALYSIS OF SOFTWARE-DEFINED NETWORKS

Tkachova Olena, Issam Saad, Mohammed Jamal Salim

***Abstract**: Methods and mathematical tools for formalization analysis and verification of Software-Defined Network protocols are suggested in the paper. E-networks apparatus and linear temporal logics are proposed to be used as mathematical tools for modeling and analysis. These mathematical tools can be transparently used to analyze critical requirements and properties of Software-Defined Networks implementation - OpenFlow protocol. The suggested methodology is based on the modeling approach and sequence analysis of changes of OpenFlow protocol elements states. Apparatus of E-networks models the element states of SDN implementation. Such approach allows taking into account the asynchronous nature of complex processes in OpenFlow protocol and performing stateful inspection. The use of models of E-network allows analyzing such properties as reachability, boundedness, liveliness, find loops and deadlock.*

The given mathematical method also allows finding difference between specification and designed networks. The method is based on the verification of temporal logic within the context of relationships of satisfaction. Using temporal logic formalisms allows to analysis and verify inconsistent of specification's requirements. For this purpose formalism was designed that allows checking out the requirements form different specification versions of OpenFlow protocol.

***Keywords**: Openflow, FlowVisor, slice, E-network, temporal logic*

***ACM Classification Keywords**: C.2.1 Network Architecture and Design - Network communications*

Introduction

The paradigm of Software-Defined Network (SDN) is designed to simplify the process of data transfer and network infrastructure management [Open Networking Foundation, 2012]. One or more controllers manage multiple switches, responsible for the transmission of packets between network devices in SDN architecture.

The management protocols in the modern information networks must perform multiple simultaneous tasks for the successful functioning of the network infrastructure: from the routing and monitoring data to the access control and load balancing on network elements. According to SDN concepts, the

specified network protocols provide orchestration and choreography between management elements. OpenFlow protocol is used for implementing a wide variety of network tools and protocols. There is including routing circuits witch and packet-switched traffic over the same switch, in-network load balancers, wireless sensor networks and optical network [Software-Defined Network, 2012].

While OpenFlow provides powerful means for specifying control-plane logic and protocols, the resulting networks may not satisfy necessary safety conditions and QoS guarantees [OpenFlow Switch Specification, 2012]. The value of service fields in OpenFlow protocol may be different depending on the requirements of the network infrastructure and services, operating in a network. OpenFlow protocol has no final realization to date. OpenFlow based on a number of different specifications that have significant differences between them.

Numbers of inconsistencies may occur in the process of development and implementation of SDN, leading the emergence of critical errors in the operation of the network. To eliminate errors and checking correct operation of the protocol is necessary to carry out its full analysis in the early design stages. These flaws could be extremely damaging if not detected before system deployment, especially in applications like vehicular control networks.

The use of formal methods in the context of the OpenFlow analysis and verification problems allows to clearly describing the set of properties that should be possessed in protocol, check their feasibility and compliance.

The advantage of formal methods is that the formalism of orchestration and choreography of processes, which corresponds to the correct operation of the protocol OpenFlow, can be used in the subsequent development as prototypes. Using prototypes allows to reduces time and costs of development.

OpenFlow. Main requirements and formalization

OpenFlow is an emerging standard for Software-Defined networking that enforces a clean separation of the data and control planes. The key elements of the protocol are OpenFlow switch and controller. OpenFlow switch acts as the custom side, controller acts as a central control unit. An OpenFlow switch routes data plane packets based on flow tables: ordered lists of rules guarded by a pattern to be matched over a packet header. These rules are installed by a controller, which is connected to each switch via a secure, dedicated link. The controller handles packets sent to it by OpenFlow switches and installs flow rules in the flow tables. The functionality of the controller determined completely by the application that it is being used to implement, but all controller programs must communicate with switches only by installing flow rules [Architecture SDN, 2014].

SDN provides a powerful, scalable infrastructure that used to tailor networks to specific tasks. To date, the concept of SDN used for building distributed network of educational institutions („campus" network), data centers, and distributed intelligent networks (networks with distributed data repositories). Example of Software-Defined Network topology (control plane level) shown on Figure 1.

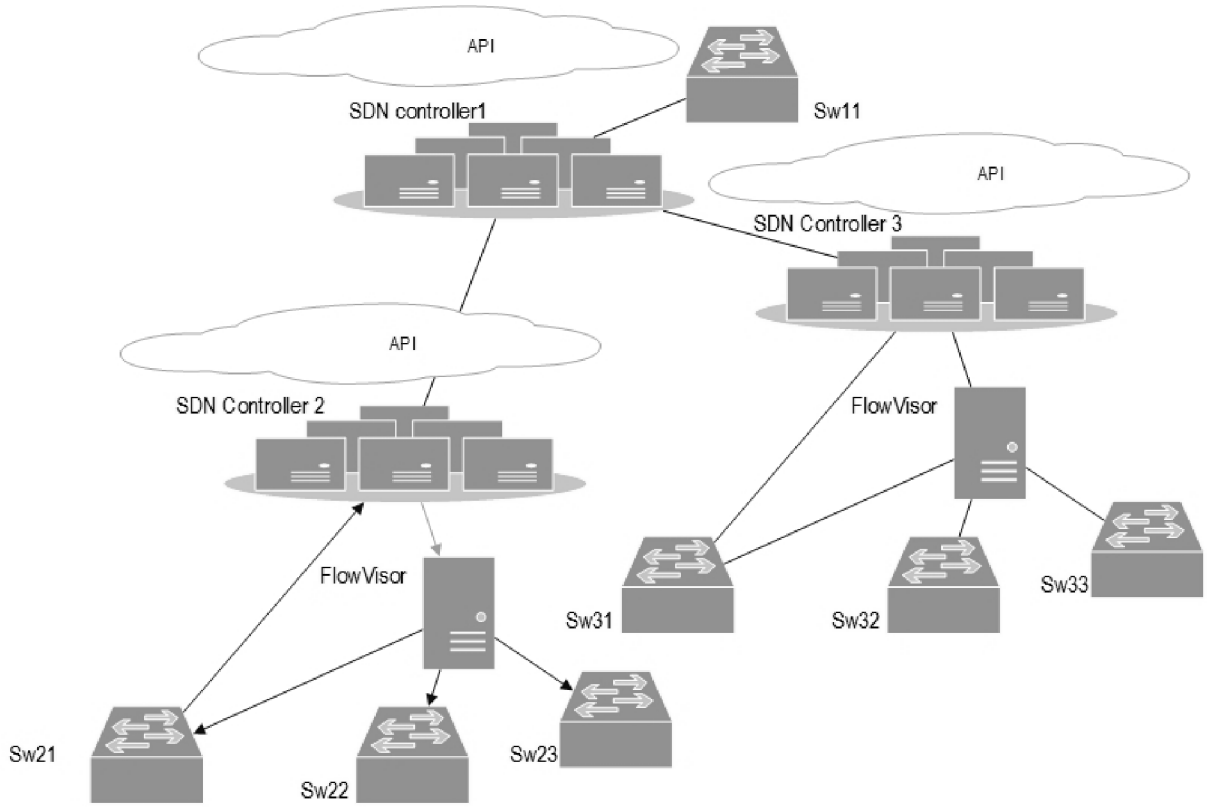


Figure 1. Example of Software-Defined Network topology

SDN network configuration (controller and switches states) has dynamic parameters. Their correct interaction depends on the number of factors: the formation of the same type of routing tables that supports the same version of OpenFlow protocols, application of the same network operating system on controllers (NOX or POX), using the same packet processing mechanisms and other [Gude et al, 2008].

Each network represents as set of number parameters and their attributes. Transmission of information from node A (belonging to the field vision Sw_1), a node B (belonging to the field of vision Sw_2) can be defined as a formulation:

$$FI(T):(SDN_1,C_1)\xrightarrow{j_1}(SDN_2,C_2)\xrightarrow{j^{\ni}_1}(SDN_3,C_3),$$

SDN_i – network configuration, C_i - temporary controller state, where i=1, 2, 3, etc.; j - message (instant) state.

Control plan according to SDN structure is based on the software, in particular on software implementations and OpenFlow controller switches (Sw). All SDN network design and planning beyond physical connectivity can be done in software, specifically in the design of the OpenFlow controller program. This allows for significantly expand network processing and routing functionality introduces the possibility that (like any other software program) logical or design errors could lead to unsafe or incorrect behavior.

Regardless of the structure and scope of the network infrastructure must be complied all requirements that match a specified level quality of service (QoS). The requirements that we will analyses such properties as:

- No losses in the process of routing and packet forwarding;
- No loops: any packet that enters the network will eventually exit the network;
- Network liveliness: all network elements involved in the transmission of data must be available and active;
- Correct work: all network conditions to ensure correct operation of the protocol to be achieved;
- Correct data paths formation: the search for the optimum data path, taking into account changes in the routes of transmission and mobility of end users;
- Safety: no unauthorized processes in the operation of the protocol;
- Quality of Services.

The first three requirements ensure reliably function, correct operation of the network (protocol OpenFlow), and ensure the provision of services. Correct formation data paths, data safety and providing the specified quality of service properties are individual for each SDNs: software implementation dependent OpenFlow switches and controllers, as well as the version of the protocol OpenFlow.

Overview of implemented SDN solutions [OpenFlow Switch Specification (Series), 2014] showed that the majority of errors occur is due to different interpretations of OpenFlow standards.

In order to analyze and verify the properties expressed above, it is necessary to represent formal methods that makes it possible to solve problems and eliminate errors. Specifically, in order to specify

correct behavior of systems (or any set of defined algorithms), it is first necessary to define mathematical formalisms.

The firstly, it is necessary to choose a set of processes P (include terminal processes P_t), that is defined in the OpenFlow specification and correspond to network elements S_{SDN} (where S_{SDN} - state changing). Correspondence processes and states can be represented as a function $f : P \to S_{SDN}$.

Since SDN, is dynamic in nature, each entity OpenFlow protocol S'_{SDN} from set S: $S'_{SDN} \in \{S\}$, is correspond to individual set of processes P', $P' \in \{S\}$. Logical system that determines interaction of OpenFlow elements and the sequence change of their states (f, S_{SDN}, P) allows performing step-by-step verification services of protocol.

The required parameters guaranteed if and only if the requirement below is met:

$$(f, S_{SDN}, P)_{spec} \equiv (f, S_{SDN}, P)_{\text{mod}} \quad (1)$$

During the development and implementation of SDN is necessary to verify correspondence and consistency requirements [Wiatkowska et al, 2011]. Formal verification specification consistency of commands and rules can be represented as follows:

$$(f, S_{SDN}, P)'_{spec} \equiv (f, S_{SDN}, P)''_{\text{spec}} \quad (2)$$

SDN by nature are complex distributed systems. The number of network elements may vary considerably. Model approach is used as part of the formal submission for this class of systems. Its advantage is clear formalization of the elements and their interactions, and the accounting structure (topology) and behavioral properties of processes.

Using E-network tools for modeling process of OpenFlow protocol

For modeling OpenFlow protocol is proposed to use the mathematical apparatus of E-networks [Losev, 2002]. E-networks have several advantages: allows the modeling of synchronous and asynchronous nature. Markers invariants takes into account the peculiarities of the protocol OpenFlow. OpenFlow operation defined by a set of attributes specified E-network.

E-network can be applied to tasks of different levels of abstraction: at the level of change in the OpenFlow switch routing tables (FlowTable) and its interaction with the SDN controller, routing data analysis, which are determined by switching rules. All network conditions and commands execution are modeling by transitions when using the E-networks, change switches and controller state displays positions.

The disadvantage of E-networks is absence of possibility to identify controller workload, and some of the characteristics related to time of execution (delay, executing time, packet arrival time). However, within of tasks, this factor is not taken into account.

As an example, the E-model network that depicted on the Figure 2. This model corresponds to the initial stage of the exchange of control messages between the controller and the switch [Software-Defined Network, 2012].

The State „Switch learning (porti)" corresponds to the new message that arrives on the switch ports (porti). State „Don't find" corresponds to the process of forwarding new control packet to controller. Switch forwards the message to the address that is specified in the destination source-forwarding table (state „Forwarding data"). State „Change table" corresponds to the processing of messages (change counters, priority destination). State „Change Command" corresponds to the transfer of the processed packet switch, which can be carried out directly or using FlowVisor.

The packets can be delivered to the switch that initiated the connection is established or all switches in the network in accordance with the OpenFlow specification. State „Receive all ports" defines this possibility in the given E-network model.

Consider the E-network transitions. Transition P_0 corresponds to the process of a new message arrives (packet) of data devices. The transition P_1 is a managed; it corresponds to a process of „learning" the switch. A control r(x) predicate contains information about the data listed in the forwarding table. Next rule must be met: message is forwarded to the controller only in the case when switch could not find any matches in the flow table. The process P_2 contains preconditions perform: it made the transfer destination or the subsequent processing of the message fields. Transitions P_3 and P_4 corresponds to the process of packet processing controller. Transition P_5 corresponds to the transfer of the same route. Transition P_6 corresponds to encapsulate the packet by FlowVisor. Transitions P_t and P_t' corresponds to processing a packet that is received from the controller in accordance with an instruction set: entry in the forwarding table modification $Tf(f_{idin}(Modif, t))$ and packet drop $Tf(Drop)$.

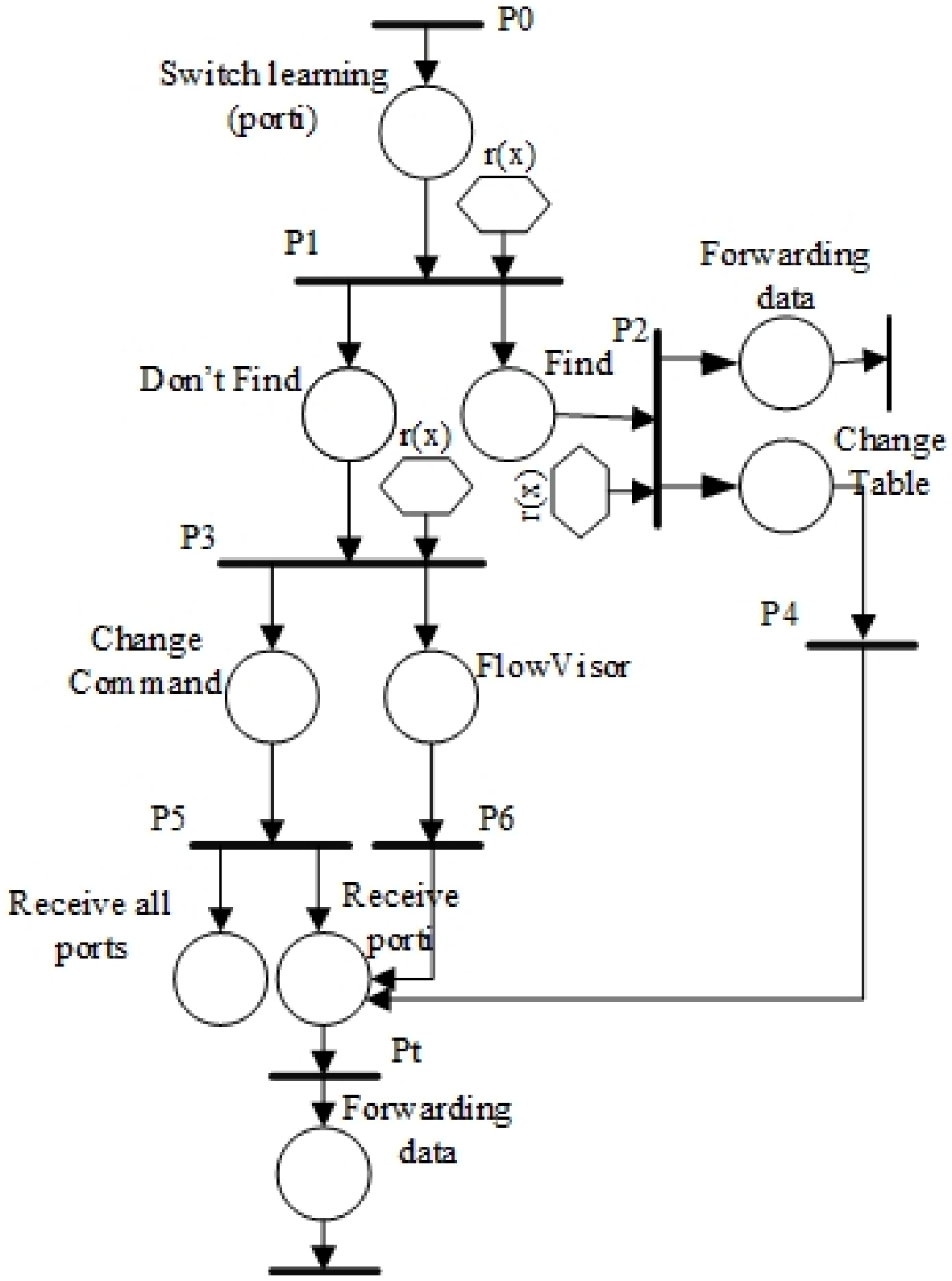


Figure 2. E-network model for OpenFlow initialization process (control plane)

Using temporal logic for analyzing and verification OpenFlow protocol

OpenFlow specifications are represented by verbal methods and UML [OpenFlow Switch Specification (Series), 2014]. The main task of formalization is mapping verbal description of protocol behavior in terms of temporal logics. Let us consider two formalisms for describing properties of modeled SDN systems: Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). LTL formulas are capable of expressing temporal properties of existence the set of model states. The first four parameters that are defined above (no losses in the process of routing and packet forwarding; no loops, network liveliness) are expressible as LTL formulas over a predicate that is true if and only if the corresponding model state is enabled [Jackson, 2002].

As has been noted, the development and modification of protocols may experience a whole series of defects in contrary to the requirements specification. Each temporal logic formula must be specified in terms of relationship satisfaction. This action will verify each temporal logic formulas for consistency requirements (|=) [Clarke & Emerson, 1981].

Definition 2.1. The formula is defined between state s and an element of the formula φ, which must be true in this state (s|=φ). The formalism s|=φ is true if and only if all of the true state. The formalists are associated by attitude of satisfaction:

$$s \models p \Leftrightarrow p \in Label(s);$$

Definition 2.2. Two temporal logic formulas are inconsistent, if there are no such conditions for which formulas are mutually exclusive. Consistency is verified on the way the formulas and formulas for state.

For example, the two statements are „reserved channel cannot be blocked" $H.reserv \models Gactive$ and „after timeout reserved channel is blocked" $H.reserv \models time-out \rightarrow \neg(active)$.

Each SDN controller manages only its logical part of network and cannot affect the operation of the other network part. The separation of the network into logical networks (isolation or slicing) is quite time-consuming task.

FlowVisor is implemented as an OpenFlow proxy that intercepts messages between OpenFlow-enabled switches and OpenFlow controllers. FlowVisor defines a slice as a set of flows running on a topology of

switches. For this purpose, FlowVisor sits between each OpenFlow controller and the switches. FlowVisor, like any software networking solution has its own parameters and characteristics.

FlowVisor modification and configuration sections depend on the logic of the program code, network characteristics and version of the specification. To date, FlowVisor functionality supported OpenFlow version 1.1 only, the implementation of other versions often contain errors due to inaccuracies in the specification.

As example, we consider the slice creation process. Perform analysis of all interaction process between FlowVisor and controllers. Each slices restricted by affecting flows in their flows pace. The FlowVisor performs message rewriting to transparently ensure that a slice only has control over its own flows and cannot affect other slices flows. Not all rules can be rewritten to fit to a slice: the FlowVisor will only make rules more specific.

Slices configured in accordance with specific rules that depend on the requirements for virtualization and specifications of protocol.

Basic FlowVisor rules for each slice are [FlowVisor, 2009]:

- Data (control information) is read-only and forward;
- Modification of the data is permitted;
- Data should be discarded.

Usually slices are configured in accordance with the rules and policies laid down at work SDN. However, some of the rules, depending on the specification may overlap, which is contrary to the rules of traffic isolation. As an example of FlowVisor bandwidth slicing, consider the interaction between the SDN controller (S) and FlowVisor (Fl). The process depicted on Figure 3. FlowVisor and controller have two type of buffer: input and output buffer. In our example we are looking for interaction between Flow Visor's output buffer (out) and controller input buffer (in).

SDN controller S sends a message m to Fl. Firstly controller puts the message in the output buffer. FlowVisor receives messages by removing them from your input buffer in [FlowVisor, 2009]. For this example, the temporal logic alphabet consists of a set of atomic propositions: message exist in the buffer „in" and message exist in the buffer „out": AP = $\{(m, m_1, \ldots, m_n) \in out\}$, $=\{(m', m'_1, \ldots, m'_n) \in in\}$.

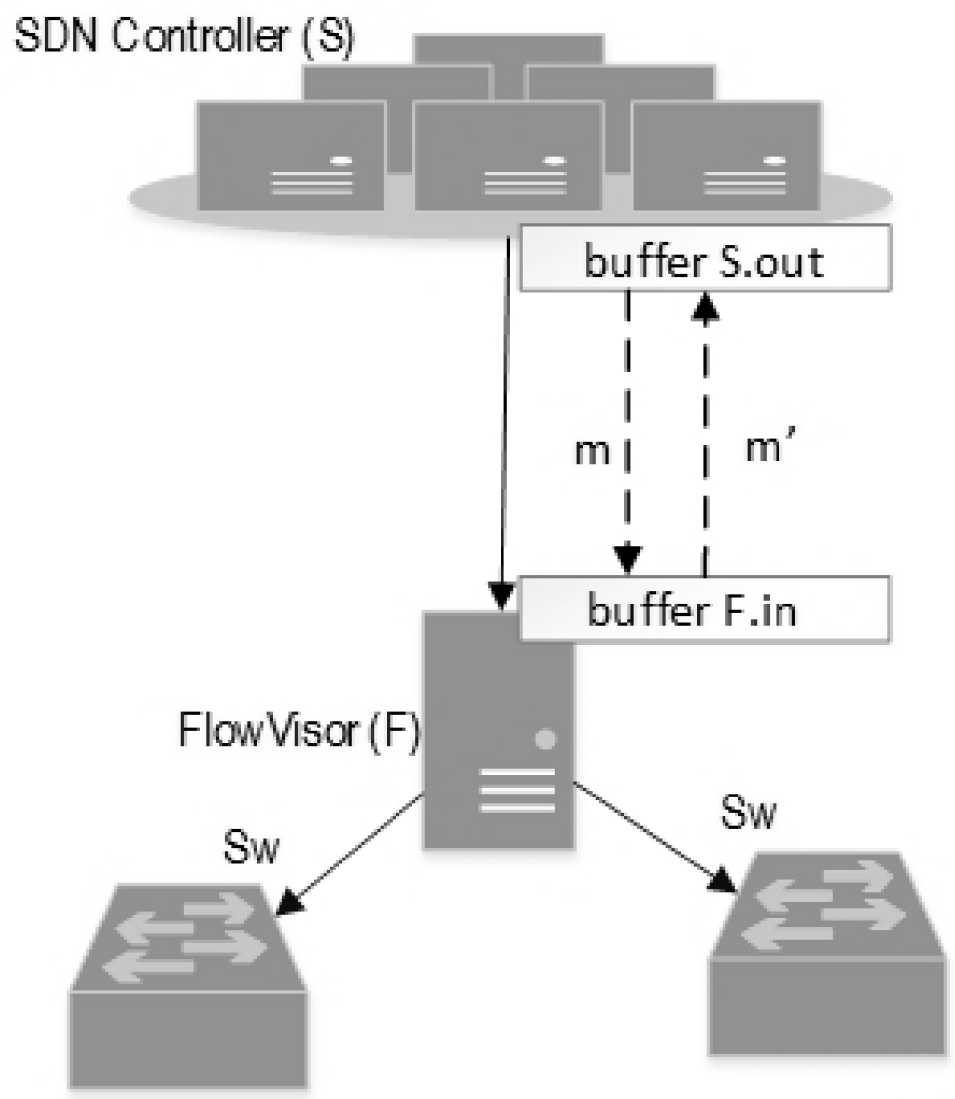


Figure 3. Example SDN Controller-FlowVisor interaction

The formulas of temporal logic for FlowVisor slice isolation can be represented as noted below:

„The message cannot be in both buffers at the same time":

$$G(m:out \wedge \neg m:in) \tag{3}$$

„FlowVisor do not lose messages":

$$G(m:out \rightarrow F(m:in)) \tag{4}$$

„After receiving confirmation that the message has been delivered to the recipient, there is its removal from the buffer .out":

$$G((m:out \wedge m:in)U\neg(m:out)) \quad (5)$$

„Messages sequence is stored in the slice":

$$G(m:out \wedge \neg m':out \wedge F(m':out) \rightarrow F(m:in \wedge \neg m':in) \wedge F(m':out))) \quad (6)$$

The composed of the following rules to check for consistency. Rules are true throughout the interaction between FlowVisor and SDN controller and correspond to the formulas (8 - 11):

$$S.out \models G(m:out \wedge \neg m:in) \Leftrightarrow S.out \models m:out \wedge S.out \models \neg m:in; \quad (7)$$

$$S.hannel \models G(m:out \rightarrow F(m:in)) \Leftrightarrow m:out \neq (m:in); \quad (8)$$

$$S.out \models G((m:out \wedge m:in) \rightarrow F\neg(m:out)) \Leftrightarrow$$

$$\Leftrightarrow S.out \models (m:out \wedge m:in) \wedge S.out \not\models \neg(m:out) \quad (9)$$

$$S.hannel \models G(m:out \wedge \neg m':out \wedge F(m':out) \rightarrow F(m:in \wedge \neg m':in) \wedge F(m':out))) \Leftrightarrow$$

$$\Leftrightarrow m:out \wedge \neg m':out \wedge F(m':out) \neq (m:in \wedge \neg m':in) \wedge F(m':out) \quad (10)$$

Formula 8 and 10 contain a contradiction:

$$\Leftrightarrow S.out \models (m:out \wedge m:in) \wedge S.out \not\models \neg(m:out) \text{ and}$$

$$S.out \models G(m:out \wedge \neg m:in) \Leftrightarrow S.out \models m:out \wedge S.out \models \neg m:in.$$

Thus, the use of temporal logic allows detecting contradictions in the original specification. The cause of the loops and packet drop is the absence of mechanisms for timely warning SDN controller of overcrowding and lack of control commands.

Conclusion

The basic principles and properties of Software-Defined Network are described in the paper. Such properties as liveliness, lack of deadlocks and looping must be executed for correct work of SDN. We analyzed problems that lead to emergence of contradictions and the conflicts in work of the protocol.

The main problems are distinctions of functional requirements in different versions of specifications and incorrect representation of specification requirements. The suggested method is based on the modeling approach and sequence analysis of changes of OpenFlow protocol elements states. The E-network tools were used as a method for modeling and analyzing the SDN properties and their components. The formulas of temporal logic were used for the analysis of the specification in relationship satisfaction context.

Two processes of OpenFlow operation were considered: the process of new flow initialization and logical slice formation. Analysis of the E-network model for logical slice formation shows the place where deadlocks and loops can accrue. The sequence of action (formulas 8-9) leads to apparent of loop.

Bibliography

[Architecture SDN, 2014] Architecture SDN [Electronic resource] // Open Networking Foundation, 2014. Mode of access: https://www.opennetworking.org/

[Clarke & Emerson, 1981] E. M. Clarke and E. A. Emerson, “Synthesis of synchronization skeletons for branching time temporal logic”, In Logic of programs: Workshop, Yorktown Heights, NY, Springer-Verlag, 1981.

[FlowVisor, 2009] FlowVisor: A Network Virtualization Layer [Electronic resource] // Open Networking Foundation, 2009, Mode of access: http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf.

[Gude et al, 2008] N. Gude at all, “Nox: towards an operating system for networks”, ACM SIGCOMM Computer Communication Review, 38(3), 2008, pp. 105–110.

[Jackson, 2002] D. Jackson, “Alloy: a lightweight object-modelling notation”, ACM Transactions on Software Engineering and Methodology, 11(2), 2002, pp. 256–290.

[Losev, 2002] Ju. Losev, “Application methods of analysis for E-networks models SOD” / Ju. Losev, S. Shmatkov, I. Duravkin // Radiotechnics: All-Ukrainian. Science.-technical, 321(2), 2002, pp.49 – 151

[Open Networking Foundation, 2012] Software-Defined Networking, The New Norm for Net works // Networking Foundation, 2012, Web. 3 January 2012, Mode of access: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[OpenFlow Switch Specification (Series), 2014] OpenFlow Switch Specification (Series) [Electronic resource] // Open Networking Foundation, 2014, Mode of access: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow/

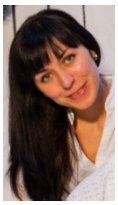
[OpenFlow Switch Specification, 2012] OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04) [Electronic resource] // Open Networking Foundation, 2012, Mode of access: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[Software-Defined Network, 2012] Software-Defined Network: The New Norm for Networks [Electronic resource] // Open Networking Foundation, 2012. Mode of access:

https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[Wiatkowska et al, 2011] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of Probabilistic Real-time Systems”, In 23rd International Conference on Computer Aided Verification, Springer, 2011, pp. 585–591

Authors' Information



Tkachova Olena *– PhD, senior lecturer, Kharkiv National University of Radio Electronics, Faculty of Telecommunications and Instrumentation, Kharkov, Lenin Ave, 14, Ukraine,*

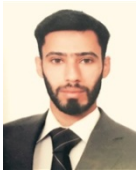
e-mail: korov4enko@mail.ru

Major Fields of Scientific Research: Methods of analysis and verification of distributed information systems and information protocols. Analyze and verification of Sortware-Defined Networks

Issam Saad *– postgraduate student, Kharkiv National University of Radio Electronics, Faculty of Telecommunications and Instrumentation, Kharkov, Lenin Ave, 14, Ukraine,*

e-mail: 71dkh@ukr.net

Major Fields of Scientific Research: Methods of analysis and verification of distributed information systems and information protocols.



Mohammed Jamal Salim *– postgraduate student, Odessa National Academy of Telecommunications named after O.S.Popov, Odessa Kowalska St., 1, Ukraine;*

e-mail: tkachova@ukr.net

Major Fields of Scientific Research: Methods to ensure the availability of services in multiservice networks