Test Derivation for the Software Defined Networking Platforms: Novel Fault Models and Test Completeness

Nina Yevtushenko[1,2], Igor Burdonov[2], Alexandre Kossachev[2], Jorge López[3], Natalia Kushik[3], Djamal Zeghlache[3]

[1] *Department of Information Technologies, Tomsk State University, Lenin str. 36, Tomsk, Russia*

[2] *Software Engineering Department, Institute for System Programming of the Russian Academy of Sciences, 25 Alexander Solzhenitsyn str., 109004, Moscow, Russia*

[3] *SAMOVAR, CNRS, Télécom SudParis, Université Paris-Saclay, 9 rue Charles Fourier 91011 Evry, France*

{evtushenko, igor, kos}@ispras.ru,

{jorge.lopez, natalia.kushik, djamal.zeghlache}@telecom-sudparis.eu

Abstract

Software Defined Networking (SDN) platforms are used in many applications and thus, should be thoroughly verified and tested before deployment; there are a number of research papers proposing various approaches for their verification. The next step is to test whether all the items are proceeding properly, i.e., to test the platform. As the requests can be considered as sets of paths which have to be properly implemented, the inputs of the System Under Test (SUT) are appropriate graphs (and packets traversing those graphs) while the output that can also be considered as a set of paths with appropriate parameters, is obtained via the appropriate platform monitoring. Correspondingly, for testing, we use a so-called proactive approach: check a submitted set of paths after forwarding a packet with appropriate parameters. We define several fault models (under different testing assumptions) with respect to the underlying resource connectivity graph (RNCT). Some conditions for deriving a complete test suite as well as the complexity upper bounds with respect to defined fault models are also established.

1. Introduction

As information technologies progress rapidly, more attention is paid towards various virtualization aspects. Virtual networks are built on the basis of classical / 'physical' ones, with respect to the user / tenant demands. Users can request rather simple networks as well as complex service function chains where network functions or services are executed in the network nodes [1]. Therefore, an important question arises: how to guarantee that the user request is respected and it has been properly implemented? The reply to this question can be found through a thorough verification and / or testing of virtualization platforms and their components [2-5].

In this paper, we focus on Software Defined Networking (SDN) [3] frameworks or infrastructures that are crucial components of virtualization platforms. In fact, SDN frameworks 'are responsible' for creating necessary virtual links between the network nodes. Such framework therefore includes an SDN controller that (together with the corresponding application) processes a user request for a virtual network or a given path, and the data plane, which is managed by the controller and where switches or forwarding devices connect various hosts between each other. We discuss how such SDN framework can be tested with guaranteed fault coverage. In fact, we focus on different equivalence classes of potential requests, taking into account different testing assumptions, such as for example: i) each requested path always starts and finishes at a host; ii) switches are only used to forward the packets; iii) the implementation of one of requested paths does not affect the others, etc.

Proper assumptions allow to reduce the enumeration of the possible user requests to guarantee the correctness of their implementation. In particular, we define different fault models in this paper; these models reflect the completeness of the corresponding test suites with respect to certain equivalent classes. We thus introduce so called host- and switch-connectivity complete test suites together with the

switch-forwarding complete ones considering various packet / switch parameters that affect the packet processing. For each of the cases, we estimate the corresponding complexity as the length of the test suite being derived.

We note that we are not aware of any test generation strategies with guaranteed fault coverage for SDN frameworks whenever various packet parameters are taken into consideration. A pioneering work has been recently published in [4] where the authors have stated the corresponding problem and have considered the necessary path generation as inputs to the SDN controller (or its proper application). We however note that these paths were simplified, for example, switch port numbers or even source / destination addresses for a packet were not modelled at all. In this paper, we decrease the abstraction level to obtain an adequate model by augmenting the fault models by necessary details and provide the contributions listed above. Correspondingly, this paper is up to some extent, a continuation of the paper [4], however in this paper, we go deeper and consider the features of a packet which should be carried on the set of paths of a test suite. In other words, paths become augmented with appropriate parameters which are important for the functioning of the SDN data plane.

The paper is structured as follows. Section 2 presents the necessary background. Novel fault models for testing SDN frameworks based on the appropriate graph enumeration are presented in Section 3. Section 4 concludes the paper and describes future research directions.

2. Preliminaries

In the SDN architecture (Fig. 1), the instructions for the data plane for packets' forwarding are provided by SDN-controller (-s). The network devices in the data plane are usually considered as hosts which can generate packets, and switches which transfer these packets through the data plane to another host. Switches have forwarding rules for an obtained packet according to different parameters such as an item where the packet came from, destination node, etc. [5]. A forwarding rule has a priority, a preamble and a postamble: the header parameters are analyzed in the preamble and the compatible rule with the highest priority is determined; this rule postamble assigns the neighbor device where the packet will be forwarded. An SDN-controller accepts sets of paths which should carry on corresponding packets, i.e., those paths can have appropriate parameters according to which the packets are then forwarded. In this paper, we consider paths which have not more than two parameters. Nevertheless, the results of the paper can be extended to bigger sets of parameters which should be taken into consideration.

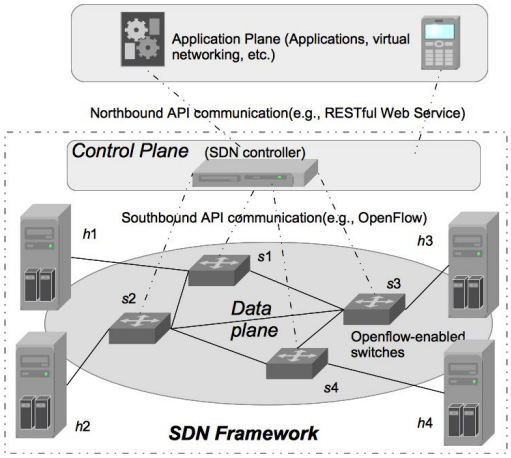

Figure 1. Example of an SDN architecture [4]

When a forwarding rule is issued to an SDN-enabled switch, a virtual link from and to other node (-s) adjacent to the switch is created, i.e., a packet accepted from adjacent nodes (hosts or switches) is forwarded to a (corresponding) set of ports that are connected to appropriate ports of other nodes.

Consider a small example in Fig. 2. Let switch $s1$ be connected with host $h1$ and have the following rules:

1) T1:100 -- Input port = 1 ^ UDP DST port = 5060 → output port = 2
2) T1:100 -- Input port = 1 ^ TCP DST port = 80 → output port = 3

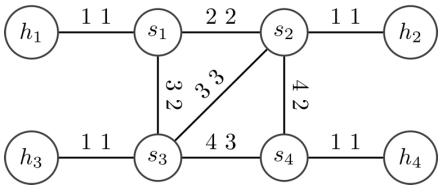

Figure 2. Example of an SDN data plane as an augmented graph

When a packet with the corresponding header arrives to $s1$ from $h1$, the packet will be forwarded to output 2 (or 3) depending on the header the network packet carries, either the Session Initialization Protocol (SIP) using the User Datagram Protocol (UDP) or web traffic (HTTP) using the Transmission Control Protocol (TCP). In general, this dependence

can be more complex, for example, an output port can depend on the destination host, etc.

Similar to [4], the data plane is referred to as the resource network connectivity topology (RNCT) with the SDN elements such as hosts, switches and links between them. An RNCT is represented as a directed graph $G = (V, E)$ without multiple edges, where the set V of nodes represents network devices. Edges of the graph (the set E) represent connections between two nodes (links) in a network and without loss of generality, we represent each link as two *ordered edges* or simply *edges* for short.

3. Fault Models for Testing SDN Architectures

For assuring the quality of the SDN architectures, test derivation methods and techniques should be developed. In this paper, we consider an active testing mode where test cases or test sequences are applied to an SDN framework that is a system under test (SUT), and the conclusion about the correctness of the SUT is made after applying corresponding test cases (appropriate requests) together with packets which induce a corresponding traffic based on observations via monitoring. In model based testing techniques, test suites with guaranteed fault coverage are usually generated, i.e., in model based testing, the derived test suites are complete under appropriate testing assumptions. We later on make certain testing assumptions to deliver complete test suites for SDN frameworks under various conditions.

3.1. Testing assumptions

As usual, testing can be performed at different levels. In this paper, we consider the requests which have (parameterized) paths over a given RNCT and the task is to test whether each requested path is correctly implemented by the SDN-controller (-s) and forwarding devices. A virtual path (simply a *path* throughout the paper) is a sequence of directed edges whose head and tail nodes are hosts and all other intermediate nodes are switches. This assumption is reasonable as the goal of any data network is to share data between computing devices, through data forwarding devices. Moreover, edges are not repeated in a path (due to potential infinite loops). We assume traffic can be generated from any host to any other, and the observed path that the data follow, is considered as an 'output'. This is reasonable as well since otherwise, it is impossible to conclude about the real data paths in the RNCT. We assume that the hosts in the RNCT, do not act as switches and switches do not act as hosts. The RNCT model does not consider such possibilities even if underlying virtualization technologies allow this; the reason is that for the considered model, a device acting as a switch and a host can be modelled as two separate devices, and furthermore, this configuration is irrelevant for the model. The forwarding rules construct a virtual partial path or a set of those. Some examples of the paths obtained from an RCNT are illustrated in Fig. 3. In this case, the RNCT is a dense network with four switches as shown in Fig. 1. Each host is connected to a single switch; however, a switch can be connected with several hosts as well as with other switches. In this paper, we consider that while executing the tests, switches do not consult with SDN-controllers. If no rule can be applied to a network packet then the packet will be dropped and this action can be observed via monitoring. This assumption is reasonable as the potential re-configuration can be modelled as a new set of paths. However, considering the re-configuration of the forwarding devices is out of the scope of this paper and left for future work.

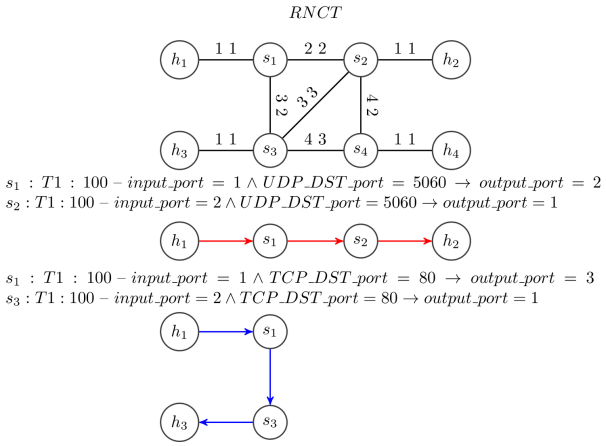

Figure 3. Example of SDN augmented paths

Given the SDN infrastructure similar to that in Fig. 1, we continue the work started in [4] proposing new Fault Models (FM) for testing the SDN framework, including the controller (-s), switches, and connections between them. A tester sends specific requests to the SDN controller application asking for different paths to be implemented in the RNCT. Those paths can have a number of parameters that determine which path the packet should take. This information will help when generating the traffic and performing monitoring as the monitoring output should follow the proper paths. The parameters are mostly related to the packet headers, the list of parameters depends on the SDN controller. For example, for the ONOS controller

this list can be found in [5]. According to our assumptions, the inputs that need to be generated in order to guarantee that the SDN infrastructure is functioning properly are the paths limited by the RNCT. In order to derive complete test suites, the following testing assumptions are made.

1) The RNCT model has no multiple edges and /or loops and is connected. In such infrastructures, multiple edges are sometimes included for increasing the communication bandwidth, the resulting link is a logical bond.
2) Moreover, the RCNT is permanent and is not changed during the request implementation.
3) Switches are capable of cloning, i.e., they can output a received packet to multiple ports; however, in this paper we do not consider such switches. Such consideration is left for the future work.
4) As mentioned above, each path is started and finished at a host.
5) A system under test (the controller and data plane, possibly, with some application (-s)) gets a path (a set of paths) which should be implemented via pushing forwarding rules to switches and this can be checked by generating and observing the appropriate network traffic.
6) A packet is not changed while traversing a path. This also is a reasonable assumption as not all forwarding devices have such capabilities, for example, to perform Network Address Translation (NAT).
7) Given two paths A and B, the result does not depend on the order how they are checked: A after B or B after A, and also it does not *matter* if these paths are implemented in parallel. Moreover, we assume that if the paths A and B are correctly implemented then the set $\{A, B\}$ is also correctly implemented and vice versa. This assumption seems reasonable if the paths A and B do not intersect in the same node with the same parameters. We assume, the requested set of paths is *well-formed* [2]. However, thorough investigation of these issues is out of the scope of the paper and left for the future work.
8) As already mentioned above, it is also assumed that switches do not query a controller about a packet that does not satisfy any rule. The switch just drops the packet, i.e., a switch implements some function f: $D_1 \times \ldots \times D_n \rightarrow P$ where D_i is the domain for the i-th parameter, and P is the set of ports of the switch.

Similar to [4], a fault model in this case is represented by a pair <=, *FD*>, where = means that the set of implemented paths in the RNCT has to be equal to the requested one. *FD* is the fault domain, a set of potential implementations, i.e., the set of all possible paths that can be implemented by the use of the given RNCT. A test case is a path that should be implemented (an example is in Fig. 3) and a *complete test suite* is the set of paths applied one by one such that the following holds. If each path of the test suite is correctly implemented then any requested path will be also correctly implemented under the given testing assumptions, i.e., up to some extent. subsection, we define three fault models which correspond to the above testing assumptions.

3.2. Defining Fault Models

FD contains all possible paths of the RNCT augmented with all possible parameters' values. A *test case* is a (parameterized) path that after erasing all parameter values is a path of the RNCT and a *test suite* is a finite set of (parameterized) paths. As the domain of each parameter is finite, the set of all RCNT paths augmented with all permissible parameter's values is a complete test suite w.r.t. <=, *FD*>. This statement is in fact, the extension of Proposition 1 in [4]. However, the number of such even non-parameterized paths is big enough; and it is bigger when augmented with all possible parameters' values. As stated in [4], this number reaches 36 even for an example in Figure 1 and will be much bigger for real SDN-networks where the RNCT graph can have hundred of nodes. For this reason, we refine the notion of a complete test suite in order to have a test suite of reasonable size and still be able to capture some critical faults of the SDN framework, since as it is shown, for example in [6], different faults models allow to capture different faults.

I. We first define a so-called *host-connectivity-(hc)-complete* test suite. A finite set *TS* of parameterized paths is an *hc*-complete test suite w.r.t. <=, *FD*> if for each pair (h_1, h_2) of different hosts h_1 and h_2, the *TS* has a parameterized path which starts at h_1 and finishes at h_2. The following statement can be immediately established.

Proposition 1. A set of parameterized paths such that for each ordered pair (h_1, h_2) of different hosts h_1 and h_2, the *TS* has a parameterized path which starts at h_1 and finishes at h_2, is an *hc*-complete test suite w.r.t. <=, *FD*>.

At least, after passing such an *hc*-complete test suite it can be guaranteed that there is an opportunity to implement a path between any two different hosts while the test suite contains exactly $n(n\text{-}1)$ test cases when n is the number of different hosts.

II. In order to add more assurance about the SDN framework implementation, the checking of the switch connectivity is desirable. A finite set *TS* of parameterized paths is a *switch-connectivity*-(*sc*)-complete test suite w.r.t. <=, *FD*> if for each pair of neighbor switches (s_1, s_2), the *TS* has a parameterized path that has an edge (s_1, s_2). Another statement immediately follows.

Proposition 2. A set of parameterized paths such that for each pair of two neighbor switches (s_1, s_2), the *TS* has a parameterized path with an edge (s_1, s_2) is an *sc*-complete test suite w.r.t. <=, *FD*>.

In other words, similar to [4], we assume that two paths are (s_i, s_j)-*equivalent* if each switch processes packets independently of many parameters only paying attention from which RNCT node a packet came. In other words, we assume that a packet obtained by the switch s_i from the node s_j is always processed correctly or wrongly independent of other parameters. By definition, a *sc*-complete test has a path of each (s_i, s_j)-equivalent class. The number of such classes equals $2P$ where P is the number of edges between switches and thus, the number of equivalent classes does not exceed $S(S-1)$ where S is the number of switches. Moreover, if there are not used links in the RNCT then we can limit ourselves only with classes which correspond to utilized links.

III. Another option is to consider that switches' rules are implemented correctly with respect to more parameters, for example, we can include the destination node into the set of parameters.

Let the output port of a switch be specified only by the neighbor node where a message came from and a destination node. A finite set *TS* of parameterized paths is a *switch-forwarding*-(*sf*)-*complete* test suite w.r.t. <=, *FD*> if for a switch s_i and each pair of its neighbor nodes n_j and n_k each of which can be a switch or a host, the *TS* has a parameterized path that has to forward a packet obtained by switch s_i from the node switch n_j to the neighbor node n_k if the destination node is d. In this case, two paths are (s_i, n_j, n_k, d)-*equivalent* if each switch processes inputs independently of many parameters only paying attention where a message came from and what is the destination port.

Proposition 3. A set of parameterized paths such that for each switch s_i, each pair of its neighbor nodes s_j and s_k, and each host d, there is a path of the (s_i, s_j, s_k, d)-equivalent class, is an *sf*-complete test suite w.r.t. <=, *FD*>.

The number of equivalent classes can be evaluated as follows: this does not exceed $S \cdot (S-1)^2 \cdot H + S \cdot (S-1) \cdot H^2 + S \cdot (S-1) \cdot H = O(S^3 \cdot H + S^2 \cdot H^2)$, where H is the number of hosts and S is still the number of switches. The upper bound depends on the relationship between the number of hosts and switches. As the number of hosts is usually much bigger than that of switches, the upper bound can be considered as $O(S^2 \cdot H^2)$ or simply $O(H^2)$. Here, we notice that the upper bound is still big enough and thus, possibly it is worth to analyze which 4-tuples are critical in order to guarantee appropriate fault coverage of a derived test suite.

In the same way, more parameters can be taken into account. If a switch is considered as a stateless component [7] then the behavior of the switch s_i can be represented as a function $f_i(p_1, \ldots, p_m)$: $D_1 \times \ldots \times D_m \rightarrow P$ where D_j is the domain for the j-th parameter, and then the f-equivalence of two paths can be taken into account. Investigating this equivalence relation is also left for the future work.

4. Conclusion

In this paper, we proposed three fault models for testing an SDN framework. Complete test suite with respect to proposed fault models can guarantee that if each path of the test suite is correctly implemented then any requested path will be also correctly implemented under appropriate testing assumptions, i.e., up to some extent.

We also mention that our approach can be applied for refined testing assumptions, i.e., to the situations when more parameters are considered and / or switch cloning is possible and / or when a switch can 'discuss' with the controller packet processing for which there are no appropriate rules, etc. These directions are left for the future work.

Another direction for the future work is to experiment with real SDN infrastructures in order to evaluate which kinds of critical faults can be detected when deriving complete test suites for the above fault models.

5. References

[1] M. Mechtri, C. Ghribi, O. Soualah, D. Zeghlache, "NFV Orchestration Framework Addressing SFC Challenges", *IEEE Communications Magazine*, 55(6), 2017, pp. 16-23.

[2] J. López, N. Kushik, N. Yevtushenko, D. Zeghlache, "Analyzing and Validating Virtual Network Requests", *Proceedings of ICSOFT*, 2017, pp. 441-446.

[3] Open-Networking-Foundation [Electronic resource] Openflow switch specification v1. 4.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[4] A. Berriri, J. López, N. Kushik, N. Yevtushenko, D. Zeghlache, “Towards Model based Testing for Software Defined Networks”, *Proceedings of ENASE*, 2018, pp. 440-446.

[5] ONOS Community [Electronic resource] Flow Rules. https://wiki.onosproject.org/display/ONOS/Flow+Rules#FlowRules-Keyandfielddescription

[6] S. A. Smolov, J. López, N. Kushik, N. Yevtushenko, M. M. Chupilko, A. S. Kamkin, “Testing logic circuits at different abstraction levels: An experimental evaluation”, *Proceedings of EWDTS*, 2016, pp. 1-4.

[7] J. López, N. Kushik, A. Berriri, N. Yevtushenko, D. Zeghlache, “Test derivation for SDN-enabled switches : A logic circuit based approach”, *Accepted for publication at ICTSS*, 2018.

6. Acknowledgements

The work was partially supported by the Russian Science Foundation (RSF), project № 16-49-03012, as well as by the Celtic-Plus European project SENDATE, ID C2015/3-1.