

Verification Framework for Software-Defined Networking

Miyoung Kang, Jong Jin Cho

School of Cybersecurity, Korea University, Seoul 02841, Korea

dasuni@korea.ac.kr, jongjcho@korea.ac.kr

Abstract— Software-defined Network (SDN) is a 5G's core technology. This provides many advantages over traditional networking by separating the controller and data planes. However, the network topology changes depend on the network configuration frequency. Therefore, it requires applying consistent network rules and providing network resilience. In this paper, provide a verification framework based on the model checking, and to ensure resilience, verify both a topology and modified topology with formal verification.

Keywords— SDN, formal verification, model checking, UPPAAL, TCTL

I. INTRODUCTION

Software-defined networking (SDN) [1] is the core technology for 5G [2] and is used to connect networks of virtual machines (VMs). The 5G core network is deployed in a distributed horizontal cloud form. SDN separates the network control and forwarding functions that were originally combined, thus removing any dependency on specific hardware equipment and giving network operators more flexibility[3]. The separation of the control plane and the data plane is achieved and controlled through an interface such as OpenFlow[4]. Network functions can efficiently cope with explosive increases in traffic by appropriately distributing control functions to the central cloud and data transfer functions to the edge cloud. However, network configurations may change over time for several reasons: node failures, QoS requirements, cost savings, etc. Additionally, for the network to be resilient, it must detect network failures such as node failures quickly, and its function have to work normally based on the user's intention. Although the network environment changes, its forwarding behavior must be ensured. Furthermore, even though the rules consistently change in an SDN environment, the rules should be consistent.

This paper presents a novel approach to the verification framework of SDN infrastructures in different environments. The objective of this paper is to present a solution to the problem that the forwarding rule must remain the same even if the network environment changes, that is, the topology changes. Although the network environment does change, the forwarding rule must still apply. This way, resilience is guaranteed. This study proposes a verification framework, and this framework is modeled using switches, hosts, and controllers. This framework creates different types of environmental topologies where modeled topology with

forwarding rules applied which is part of the same network rule. This is proven using time computation tree logic (TCTL) [5] applied verification properties. In the individual modeling process, if safety, reachability, and liveness are proven in the individual modeling process, two models cannot be described as equivalent, although it appeared as 'satisfied.' However, the properties that include TCTL(Time computation tree logic) based on the modeled applied network forwarding rule in topologies. In the two different topologies, these are proven through two individual properties. Even it is "satisfied", both topologies have different forwarding rules. In two different topologies, these are proven through the same properties. It said "satisfied," both topologies have the same forwarding rules. In this study, we use a verification framework based on host, controller, and switches to create two different model topologies. It also offers TCTL properties and is verified using an UPPAAL[5] tool.

A controller rule must be applied to a changing environment based on real time. Therefore, even though the environment has changed, it has to verify that the forwarding rule is still applicable. Therefore, the contribution of this study is as follows: 1) it suggests a new formal framework for ensuring network resilience; 2) it verifies different topology environments, applies the same properties, and verifies that the behavioral result is equivalent; 3) it proposes verification properties that two topologies (SDN environment) are equivalent.

The remainder of this paper is structured as follows. In section II, some related works are reviewed. We define the proposed formal framework in Section III. Based on the network model, Section IV describes the topology modeling. In Section V, we explain the proposed properties and its verification results using the properties. Finally, we present the conclusions of the paper and discuss future research in Section VI.

II. BACKGROUND

A. Software-Defined Networking

SDN originated with OpenFlow, which was developed as a protocol for future internet infrastructure control technology [6]. However, it evolved into the SDN concept centered on the Open Networking Foundation (ONF) [7], which was established in 2011. It is now used as the core technology for 5G networks.

Figure 1 shows the SDN framework, which consists of three layers: the application layer, the control layer, and the infrastructure layer. The application layer includes network applications that can introduce new network features such as security and manageability, provide forwarding schemes, or assist the control layer in the network configuration. The application layer can provide appropriate guidance for the control layer by obtaining an abstracted global view of the network from the controllers. Examples of network applications include network management and traffic engineering; load balancing for application servers; security and network access control; network testing, debugging and verification; interdomain routing; and network virtualization. The interface between the application layer and the control layer is known as the northbound interface.

The control plane is found in the lower layer. This is involved in the programming and management of the forwarding plane. In order to achieve this, it uses information provided by the forwarding plane and defines network operations and routing. It consists of one or more software controllers communicating with forwarding network elements through a standardized interface known as a southbound interface. OpenFlow, one of the most commonly used southbound interfaces, primarily considers switches, while other SDN approaches consider other network elements such as routers. The lowest layer is the infrastructure layer, which is also referred to as the data plane. This comprises forwarding network elements. The forwarding plane is primarily responsible for forwarding data, in addition to local information monitoring and statistics collection.

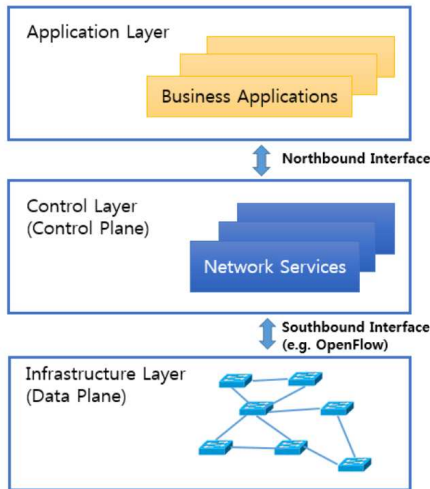


Figure 1. SDN architecture

B. Timed Automata

A timed automaton is a tuple (L, l_0, C, A, E, I) , where L is a set of locations, $l_0 \in L$ is the initial location, C is the set of clocks, A is a set of actions, co-actions and the internal transition, $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges

between locations with an action, a guard a set of clocks to be reset, and $I: L \rightarrow B(C)$ assigns invariants to locations [5]. To validate the model on timed automata, Timed Computation Tree Logic(TCTL) is used. The syntax of TCTL is as follows:

$$\phi ::= P \mid g \mid \neg \phi \mid \phi \vee \phi \mid Z. \phi \mid EX \phi \mid E[\phi \mathbf{U} \phi] \mid A[\phi \mathbf{U} \phi]$$

In TCTL, $P \in AP$ (Atomic Propositions) and g are the clock constraints. Verification attributes include reachability, safety, and liveness. Reachability is the state formula P that some reachable states can be satisfied. Safety is the property that “bad things never happen.” Liveness is the property of “good things happen someday.”

Reachability: $E \Diamond P$

Safety: $A[] P$

Liveness: $A \Diamond P$

The reachability property, $E \Diamond P$, means P is eventually satisfied along the path and should be reachable using the path formula. The safety property, $A[] P$, shows that if P is a state formula, P should be true in all reachable states with the path formula. The liveness property, $A \Diamond P$, means P is eventually satisfied.

C. Related Works

Many studies present problems and solutions for various topics related to SDN [8]-[11]. We are not the first to study formal modeling for an SDN verification.

NICE [12] is a model checking tool that uses the symbolic execution of event handlers to identify representation packets that exercise code paths on the controller. NICE detects programming errors such as no forwarding loops, no black holes, direct paths, and no forgotten packets in testing unmodified controller programs in a NOX controller.

Frenetic [13] is a high-level programming language for OpenFlow applications running on top of NOX. Frenetic allows OpenFlow application developers to express packet processing policies at a higher level than the NOX API. Frenetic also has the network language that defines the formal semantics for OpenFlow rules and improves NetCore [14] by adding a compiler.

Kazemian et al. [15] allowed the static checking of network specifications and configurations to identify important classes of failure, such as reachability failure, forwarding loops, and traffic isolation, as well as leakage problems. A framework that uses formalism, header space analysis (HSA), looks at the entire packet header as a concatenation of bits. Hassel, which is a library of HSA tools, analyzes a variety of networks and protocols. The model developed by Kazemian et al. was the starting point for the Reitblatt et al. [16] model.

Reitblatt et al. [16] developed a formal model of OpenFlow networks and proved that consistent updates preserve a large class of properties. The formal model assumed consistent network updates when transitioning between configurations. The model identified two distinct consistency levels, per-packet and per-flow, and presented general mechanisms for implementing the two levels in SDN using OpenFlow. The

verification tool, which was an implemented prototype that reduced the overhead required to perform consistent updated, checked the correctness of the controller software.

Canini et al. [17] introduced a formal model describing the interaction between the data plane and a distributed control plane that consists of a collection of fault-prone controllers. In addition, Kang et al. [18] introduced a framework in which the consistency between a specification and its implementation is checked by dead-lock detection in the parallel composition of two different CSR processes generated from two entities in different forms, one in the rule and the other in the OpenFlow table.

Xiao et al. [19] introduced the modeling and verification of SDN with multiple controllers to apply communicating sequential processes (CSP). The model checker Process Analysis Toolkit (PAT) verified that the models satisfied three properties: deadlock freeness, consistency, and fault tolerance.

Our framework differs from the others mentioned above in that network consistency and resilience are verified of properties on reachability, safety and liveness. The framework describes that it has the same forwarding rule by verifying it with the same properties in the network modeling of different topologies.

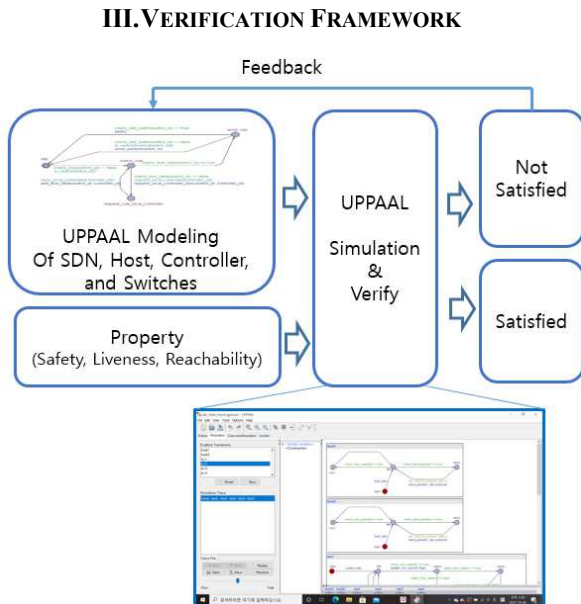


Figure 2. Formal Framework

The formal framework in Figure 2 consists of controllers, switches, and hosts at UPPAAL modeling. Simulations can be used to confirm that the model operates as intended. In particular, the safety, reachability, and liveness at Properties of the system can be verified. As a result of this verification

process, users receive either a Satisfied or Not Satisfied message. In the framework, the model and its properties are modified through feedback, and then verification is run again.

As an application of the framework, the topology is modeled with SDN components, including the hosts, controller, and switches. Furthermore, the network configuration can be changed through the alternation of the topology, depending on various network conditions such as node failure, QoS requirements, and cost reduction. Each SDN is modeled with two different topologies. The forwarding rule must be applied well, even though the network changes to a different topology. The verification properties are examined for each topology in terms of safety, liveness, and reachability. The maintenance of the forwarding rule is confirmed by verifying the same properties in different topologies. The results of “Not Satisfied” or “Satisfied” are displayed after the properties are input to UPPAAL Verify, the middle part of Figure 2. It can be confirmed that the same forwarding rule is applied if Satisfied is obtained as a result after two SDNs with different topologies are modeled and verified for the same property.

IV. TOPOLOGY MODELING

Switches can be connected in many different topologies. We start with a chain topology of two hosts and four switches in the network, as shown in Figure 3.

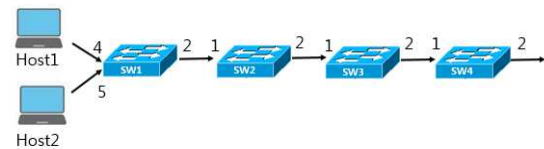


Figure 3. Host and switches in a chain topology

A. Host Modeling

The host is the starting point for a packet. Each packet in the packet starts with the address of the origin and the address of the destination. The host checks the packet reception, transmission, and transmission possibilities. The functions used for host modeling are `host_init()`, `check_rcv_packet()`, `check_send_packet()`, `rcv_update()`, and `send_packet()`, shown in Figure 4. The function `host_init()` initializes the host.

The function `check_rcv_packet()` checks whether the host receives a packet. The function `rcv_update()` initializes after receiving a packet from the host. The function `send_packet()` sends a packet from the host and randomly selects 3 or 4 as the DST (destination)

```
typedef struct {
    int host_id;
    int id;
    int src;
    int dst;
    int protocol;
    int time;
} PACKET;
```

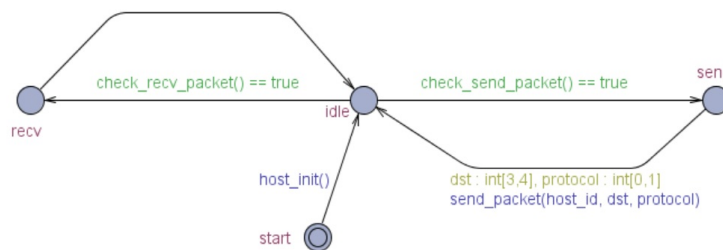


Figure 4. Host modeling

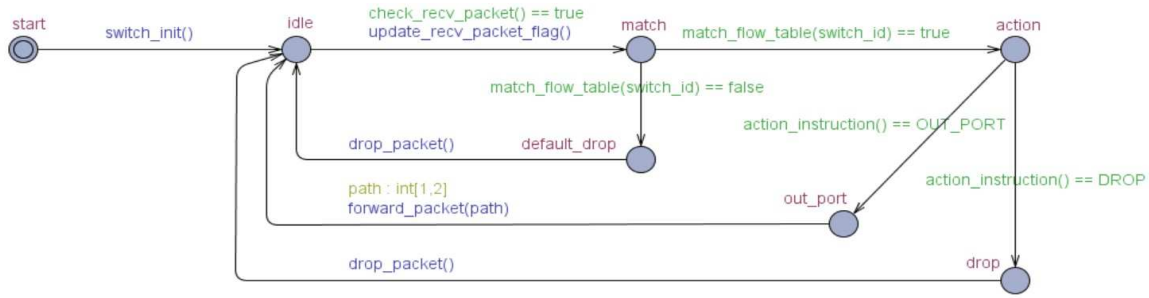


Figure 5. Switch modeling

Packets are abstracted and contain information about the host identification, packet number, source address, destination address, protocol, and time. The flow table is abstracted to contain a switch identification, a source address, a destination address, a protocol, an action, an output, a type, and a second output.

B. Switch Modeling

The switch begins in an idle location in Figure 5; after a packet arrives at the switch, the packet matches the rule to the flow table's flow entry to determine the forwarding rule for the packet. The match location checks the flow table and finds a matching field. If the flow table has matching rules($\text{match_flow_table}(\text{switch_id}) == \text{true}$), the packet goes to the action location. If the flow table does not have matching rules($\text{match_flow_table}(\text{switch_id}) == \text{false}$), it drops the packet. Switch modeling used the `switch_init()`, `check_rcv_packet()`, `update_rcv_packet_flag()`, `match_flow_table()`, `action_instruction()`, `drop_packet()`, and `forward_packet()` functions. The function `switch_init()` initializes a switch and adds a flow table. The function `check_rcv_packet()` checks whether a packet is received. `update_rcv_packet_flag()` updates the packet reception status. `match_flow_table()` checks whether the received packet has a matching entry in the flow table. If the packet is not in the flow table, it is dropped, and if it is in the flow table, it is executed as the specified action. The function `action_instruction()` checks whether the action matched in the flow table is `OUT_PORT` or `DROP`. The function `forward_packet()` forwards the packet and randomly processes the path. Each host (host1, host2) and each switch (sw1, sw2, sw3, sw4) are connected as follows:

- `host1=Host(1,1,4); //Connect host1 to port4 of sw1;`
- `host2=Host(2,1,5); //Connect host2 to port5 of sw1;`
- `sw1=OFSW(1,2,2,SINGLE,0,0); //Uplink to port 2 of sw1 and connect sw2;`
- `sw2=OFSW(2,3,2,SINGLE,0,0); //Uplink to port 2 of sw2 and connect sw3;`
- `sw3=OFSW(3,4,2, SINGLE,0,0); //Uplink to port 2 of sw3 and connect sw4;`
- `sw4=OFSW(4,0,0, SINGLE,0,0); //sw4 has no uplink.`

C. Changed Topology

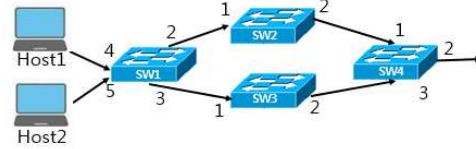


Figure 6. Changed topology

The network topology changes depending on the network environment. For instance, the topology can change from chain (Figure 3) to diamond (Figure 6). In Figure 6, host 1 and host 2 forward the packet to sw1, then the topology is split to sw2 and sw3. The packet also is forwarded from port 2 of sw1 to port1 of sw2 in the topology. Then sw1 forwards the packet to part 1 of sw3. The packet of sw2 and sw3 is forwarded to sw4.

Each host (host1, host2) and each switch (sw1, sw2, sw3, sw4) are connected as follows:

- `host1=Host(1,1,4); //Connect host1 to port4 of sw1;`
- `host2=Host(2,1,5); //Connect host2 to port5 of sw1;`
- `sw1=OFSW(1,2,2,MULTI,3,3); //Uplink to port 2 of sw1 and connect sw2; uplink to port 3 and connect sw3 using multiple port;`
- `sw2=OFSW(2,4,2,SINGLE,0,0); //Uplink to port 2 of sw2 and connect sw4;`
- `sw3=OFSW(3,4,2, SINGLE,0,0); //Uplink to port 2 of sw3 and connect sw4;`
- `sw4=OFSW(4,0,0, SINGLE,0,0); //sw4 has no uplink.`

V. VERIFICATION

In our study, the verification framework specifies and verifies each property with TCTL [5]. The query language in TCTL consists of path formulae and state formulae. State formulae describe individual states, whereas path formulae quantify the paths or traces of a model. Path formulae can be classified for reachability, safety, and liveness. The formal framework that we propose verifies three properties in two different topologies. And simulation was also performed as shown in Figure 7.

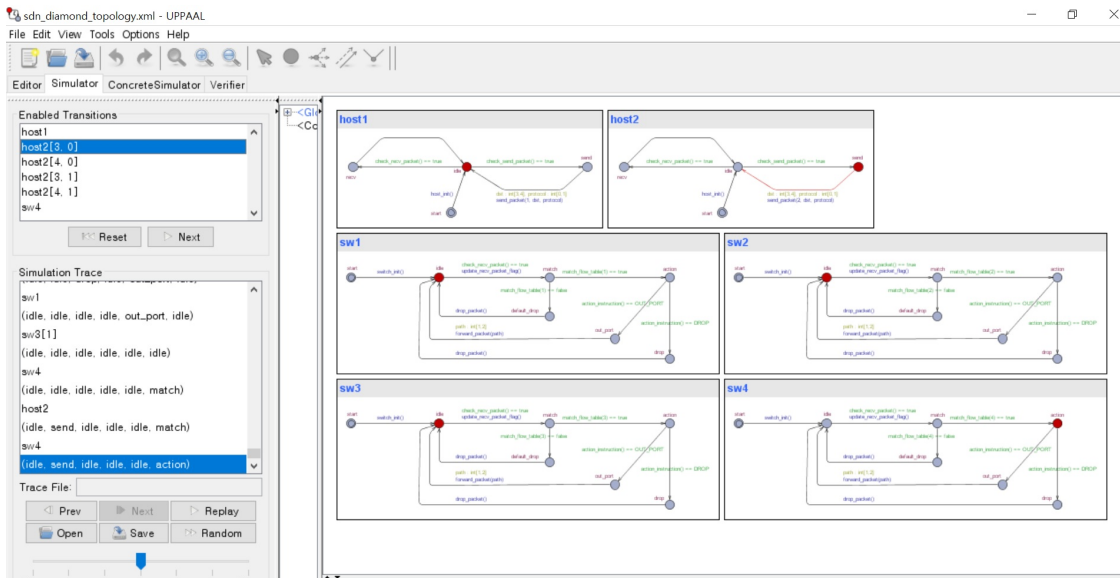


Figure 7. Simulation of the topology

A. Reachability

Reachability properties are often used to design a model to perform sanity checks and validate the basic behavior of the model [5]. They ask whether a given state formula, P , can be satisfied by any reachable state. Another way of stating this is: Does there exist a path starting at the initial state, such that P is *eventually* satisfied along that path. We express that some state satisfying P should be reachable using the path formula $E<>P$ [5]. The following formulae verify that a packet reaches the same destination for two different topologies:

Reachability : $E<>P$

$E<> sw1.match \ \&\& \ sw2.match \ \&\& \ sw3.match$

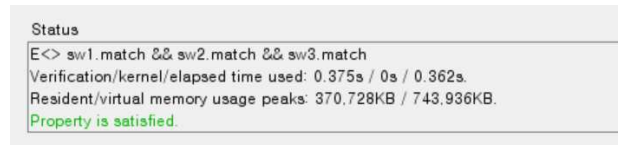


Figure 8. Verification result of Reachability

Figure 8 is verification results for “Property is satisfied.” In the other words, a packet P on a path reaches match locations of Switch 1, 2 and 3 on each topology in order to match the rule on flow table.

$E<>host1.send \ \&\& \ sw4.match$

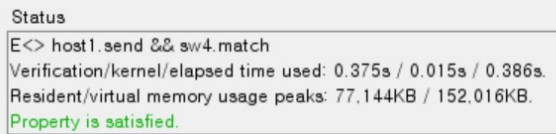


Figure 9. Verification result of the other Reachability

Figure 9 is also verification results for “Property is satisfied.” It means that a packet P on a path reaches send location of Host1 and match location of Switch 4 on each topology. This shows that whether the packet sent from Host 1 matches Switch 4.

B. Safety

Safety properties are based on the concept that “something bad will never happen.” In UPPAAL, this is formulated positively, e.g., something good is invariantly true. Let P be a state formula. We express that P should be true in all reachable states with the path formula $A[]P$, whereas $E[]P$ says that there should exist a maximal path such that P is always true[3].

Safety : $A[]P$

The packets should not always be dropped.

$E[] \text{not}(sw1.drop \ \&\& \ sw2.drop)$

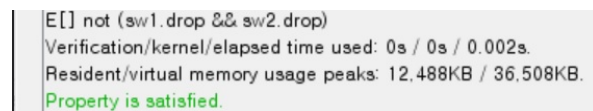


Figure 10. The verification result of Safety

Figure 10 shows the verification results for “Property is satisfied.” On the other words, a packet P on a path should not be in the each drop locations of switches 1 and 2.

C. Liveness

Liveness properties are based on the concept that “something will eventually happen.” In its simple form, liveness is expressed with the path formula $A \leadsto P$, meaning P is eventually satisfied. The more useful form is the “leads to” or response property, written $P \rightarrow Q$, which is read as “whenever P is satisfied, then Q will eventually be satisfied”[5].

Liveness: $A \leadsto P, P \rightarrow Q$

$A \Diamond \text{not}(\text{sw1.out_port} \ \&\& \ \text{sw1.drop})$

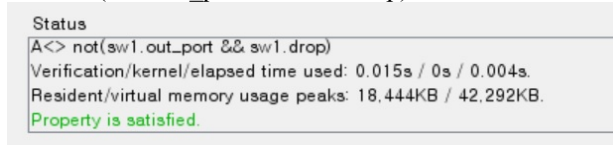


Figure 11. Verification result of Liveness

Figure 11 is also verification results for “Property is satisfied.” This means that a packet P on all path could not go out_port and drop locations of Switch 1 in order to liveness property.

VI. CONCLUSION

The network in the SDN can change for various reasons, including QoS, traffic management, and energy management. Therefore, the network resilience has to be ensured because network is changed for some reason. In this paper, to verify the resilience, we proposed a new formal framework. The formal framework is divided into two parts, which are modeling and verification. The process models network topology and verifies resilience. The resilience verification process verifies reachability, safety and liveness. To verify the reachability, we checked that a packet P reaches match locations of Switch 1, 2 and 3. To describe the safety, we show that a packet P does not reach at drop locations of Switch 1 and 2. Finally, to confirm the liveness, we verify that a packet does not reach at out_port and drop locations of Switch 1 in order to check if a packet drops in Switch 1, the packet does not forward to next Switch. Additionally, it verifies that different topologies with the same properties both apply the forwarding rule.

For our future research, we will conduct realistic models base on the verification framework and focus on game theory applied to the SDN and NFV based on VNF’s rule verification tool.

ACKNOWLEDGMENT

Basic Science Research Program supported this research through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2019R111A1A0106293913) and (2018R1A6A3A01012955).

REFERENCES

- [1] N. KcKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, “OpenFlow: enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol.28, pp69-74, 2008.
- [2] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, “NFV: State of the Art, Challenges and Implementation in Next Generation Networks(vEPC),” IEEE Network Mag., 2014
- [3] H. Kim and N. Feamster, “Improving network management with software defined networking,” IEEE Commun. Mag., vol 51, pp 114–119, 2013.
- [4] Open Networking Foundation. OpenFlow® Switch Specification Ver 1.5.1. [Online]. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>
- [5] G. Behrmann, A. David, and K. G. Larsen, “A Tutorial on Uppaal”, Formal Methods for the Design of Real-Time Systems, SFM-RT2004, pp200-236, 2004.
- [6] SDN Homepage. [Online]. Available: <https://www.opennetworking.org/sdn-definition/>
- [7] ONF Homepage. [Online]. Available: <https://www.opennetworking.org/>
- [8] D. Kreutz, F.M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no 1, pp.14-76, 2014.
- [9] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, “A survey on software-defined networking,” *IEEE Communications Surveys & Tutorials*, vol. 17, no 1, pp. 27-51, 2014.
- [10] S. Bera, S. Misra, and A. V. Vasilakos, “Software-defined networking for internet of things: A survey,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994-2008, 2017.
- [11] M. Karakus and A. Duresi, “A survey: Control plane scalability issues and approaches in software-defined networking (sdn),” *Computer Networks*, vol. 112, pp. 279-293, 2017.
- [12] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, “A NICE way to test Open-Flow applications,” *NSDI*, 2012.
- [13] N. Foster, R. Harrison, M. L. Meola, M. J. Freedman, J. Rexford, D. Walker, “Frenetic: A high-level language for OpenFlow networks,” *PRESTO’10*, 2011.
- [14] C. Monsanto, N. Foster, R. Harrison, D. Walker, “A Compiler and Run-time System for Network Programming Languages,” *POPL’12*, pp.217-230, 2012.
- [15] P. Kazemian, G. Varghese, N. McKeown, “Header Space Analysis: Static checking for networks,” *NSDI*, 2012.
- [16] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker, “Abstraction for network update,” *SIGCOMM Comput. Commun. Rev.*, vol.42, no.4, pp323-334, 2012.
- [17] M. Canini, P. Kuznetsov, D. Levin, S. Schmid, “Distributed and Robust SDN Control Plane for Transactional Network Updates,” *INFOCOM’15*, pp.190-198, 2015.
- [18] M. Kang, J. Choi, I. Kang, H. Kwak, S. Ahn, M. Shin, “A Verification Method of SDN Firewall Applications” *IEICE Trans. on Communications*, VOL.E99-B, NO.7, 2016.
- [19] Xiao, L., Xiang, S., Zhu, H., “Modeling and Verifying SDN with Multiple Controllers,” *Proceedings of SAC 2018: Symposium on Applied Computing*, 2018.



Miyoung Kang received the M.S. degree in the Department of Computer Science and Engineering at Dongguk University and the Ph.D. degree in the Department of Computer Science and Engineering at Korea University, Seoul, Korea. She is a research professor at the graduate school of Sybersecurity at Korea University, Seoul, Korea. Her research interests are Formal Methods, process algebras, software-defined networking (SDN), and security in networks.



Jong Jin Cho received B.S in Security and Risk Analysis, Cybersecurity at the Pennsylvania State University, University Park, PA, USA, in 2019. In addition, he had two years of professional IT industry experience in New York City, Chicago, and Seoul. Currently, he is an M.S candidate in the School of Cybersecurity, Korea University. His current research interests are Threat Modeling, Formal Methods, and Secure Software Engineering.