Formal Specification and Verification of Firewall using TLA+

Young-Mi KIM¹, Miyoung KANG², and Jin-Young CHOI³

¹ Department of Computer & Radio Communication Engineering, Korea University, Seoul, Korea
² ITRC Center, Korea University, Seoul, Korea
³ Graduate School of Information Security, Korea University, Seoul, Korea

Abstract - As interest in the Internet of Things (IoT) grows and the traffic from it is expected to increase, the Software-Defined Networking (SDN) technology that can dynamically scale the network is important. The ability to dynamically deploy a network is an advantage of SDN, but it must ensure reliability of the network configuration to be deployed in advance. It is necessary to ensure that the components applied by SDN are adequately deployed and free from errors. In order to solve these problems, there are many studies applying formal methods. In this paper, we show that the firewall rules on a single switch managed by SDN are formally specified using TLA+ and verified using TLC, a model checking tool, so they can be properly configured.

Keywords: TLA+, Formal Methods, Firewall, Software-Defined Networking (SDN)

1 Introduction

As interest in IoT grows today, network scalability is becoming increasingly important. SDN, a programmable networking management scheme, can be one of the technologies that will further advance the development of IoT. The most prominent feature of the IoT era is the massive data transfer, which makes traditional network approaches impossible to dynamically scale computing large data transfers. SDNs are dynamically manageable on the network, which help to respond timely to traffic [1]. That is, centralized and programmable SDN environments can easily adjust to the rapidly changing needs of businesses.

However, SDN applications with errors need verification because they affect the entire networks. Achieving the correctness of dynamically changing SDN applications is a major challenge, and formal methods can help to solve these problems. There are many approaches using the formal specification and verification such as [2]-[5]. Rupak Majumdar et al. [2] formally specify the safety properties of SDN issues using Murphi as an input language of Kuai, a distributed enumerative model checker. Actually, Murphi [6] has a formal verifier, the Murphi tool which is an enumerative (explicit state) model checker, but Kuai is used as a model checker in [2]. Miyoung KANG et al. [3] and M.-K. Shin [4] propose VeriSDN, the framework verifying the firewall application among SDN applications, which uses pACSR [7], an extension of Algebra of Communicating Shared Resources (ACSR) [8]. These works are very meaningful in that they show the techniques ensuring the correctness of the system by formally specifying and verifying it. But, when dealing with large, complex or subtle problems, the formal methods such as Murphi and pACSR used in [2]-[5] may be difficult to be used in industry. To overcome the difficulties of using formal techniques, we use TLA+ [9], Temporal Logic Actions, to specify firewall rules in this paper, inspired by the use cases of Amazon [10][11]. Amazon, which provides web services, has used formal methods from around 2014 to solve the correctness of an ever-growing set of algorithms for replication, consistency, concurrency-control, fault tolerance, auto-scaling, and other coordination activities. After evaluating several methods, Amazon realized that TLA+ could solve their real problems, and leveraged it. Now, Amazon actively encourages the use of TLA+, because it is easy to learn and gives them beneficial results from real problems.

TLA+ is easy to understand and learn, so users who have difficulties using formal methods can easily access and use it like experience of Amazon. And TLA+ is a formal specification language being actually used in industry, and Amazon has gained significant values such as finding bugs and providing a thorough understanding of the service before releasing it as a product by using TLA+. These advantages of TLA+ have motivated to use it for the specification of SDN firewall rules. This paper focuses on specifying and verifying firewall rules that operate on a single switch managed by SDN. The firewall rules based on the pACSR specification of Miyoung KANG et al. [3] are specified with TLA+ and for the verification we use TLC. TLC, a model checking tool, takes a TLA+ specifications and checks the desired properties. This paper contributes to show the ease of using formal methods in managing SDN applications and guide the basic way TLA+ and TLC can be used when formally specifying and verifying the firewall application.

Section 2 introduces TLA+ syntax and semantics helping understanding of the specification in this paper. Section 3 shows the formal specification and verification of firewall operating on a single switch and section 4 describes related works approaching with formal methods efficiently to manage SDN applications, and lastly section 5 ends with conclusion.
2 Formal Specification with TLA+

In the late 1980’s, Leslie Lamport invented TLA, Temporal Logic Actions, providing a mathematical foundation for describing systems by specifying the behavioral properties of a system. He then came up with the language added the ideas for modularizing large specifications to TLA, and called TLA+. TLA+ is quite good for specifying a wide class of systems from program interfaces (APIs) to distributed systems [9]. TLA+ first appeared in 1999. Since then, a toolbox including IDE (Integrated Development Environment) and TLC, model checking tool, has been developed. TLA+ specifications are written with logic and mathematics and used to find errors in a system design. TLC takes a TLA+ specifications and checks the desired properties, tracing all possible execution. TLA+ is simple to learn and apply among formal specification languages.

The next subsection briefly describes TLA+ semantics to help understanding the specification in this paper. The detail about TLA+ is in [9].

2.1 TLA+ semantics

Most TLA+ constructs are for expressing ordinary math, and they use propositional logic, sets, and predicate logic for making formula. The simple form of TLA+ specification consists of Init and Next as below.

\[ \text{Init} \wedge [\text{Next}]_{\text{vars}} \quad \text{(1)} \]

\( \text{Init} \), the initial predicate, is a state formula describing the initial state(s). \( \text{Next} \), the next-state action, is an action formula formalizing the transition relation and usually is expressed in a disjunction \( A_1 \lor \ldots \lor A_n \) of possible actions/events \( A_i \). And \( \text{vars} \) is the tuple of all variables and is used for transitions that don’t change \( \text{vars} \) meaning stuttering transitions. \( \Box F \) is a temporal operator and means that \( F \) is always true.

\( \text{Init} \) and \( \text{Next} \) is expressed using variables and operators. An unprimed variable represents its value in the starting state and primed variable represents its value in the ending state. For example, for any variable, \( v = v \) means that the value of the old state equals the value of the new state and can be expressed with UNCHANGED \( v \).

TLA+ provides the built-in operators such as constant operators representing logic, sets, functions, records, tuples, and strings and numbers, and action and temporal operators. \( (e_1, ..., e_n) \) means the \( n \)-tuple whose \( i^{th} \) component is \( e_i \), and \( e[i] \) means the \( i^{th} \) component of tuple \( e \). IF \( p \) THEN \( e_1 \) ELSE \( e_2 \) construct meaning \( e_1 \) if true \( p \), else \( e_2 \) and LET...IN \( e \) construct meaning \( e \) in the context of the definitions as miscellaneous constructs are provided in TLA+.

In TLA+, \( f[x] \) instead of \( f(x) \) expresses the value a function \( f \) assigns to each element \( x \) of its domain. In \text{EXCEPT} construct for a record, for any function \( f \), the expression \( [f \text{ EXCEPT } ![c] = e] \) can also be written as \( (2) \).

Mathematically, a record is a function whose domain is a set of strings, and \( e.h \) indicating the \( h \)-field of record \( e \) stands for \( e[“h”] \). \([h_1 : S_1, \ldots, h_n : S_n] \) means a set of all records with \( h_i \) field in \( S_i \).

3 Modeling and Verification of Firewall

In this section, we describe how to formalize a firewall based on the pACSR firewall rules used for the formal modeling in [3] with TLA+ and to check the specified firewall rules using TLC.

The reason that a firewall among SDN applications is targeted for applying formal methods in this paper is because it is simple but critical when errors occurred. A SDN controller manages a lot of applications responsible for many policies such as security policies, firewall policies, and Qos policies usually by rules as shown in Fig. 1. The packets at each switch are forwarded and processed by the rules applied by a SDN Controller with firewall policies. At this point, there may occur conflicts among switches under the SDN controller. So, the specification and verification of the firewall before applying the rules to switches are necessary. This paper introduces the easy and formal way for solving the conflicts problem using TLA+.

![Fig. 1 SDN Controller.](image)

3.1 Formal Specification with pACSR

In this paper, the firewall rules based on the pACSR specification of Miyoung KANG et al. [3] are specified with TLA+. TABLE 1 formally specifies \( FW \) using pACSR called process algebra, it means that a firewall checks packets by matching a sequence of rules in order. A process \( T \) is repeatedly called, an action \( Act \) can be called in that process. A packet \( pkt \) arriving at the firewall is performed the corresponding process \( Act(a_i) \) if there is a matched condition \( ci(pkt) \) for the packet. If the \( a_i \) equals “allow”, the packet leaves the firewall, otherwise it returns to the packet receiving state \( FW \) meaning the packet \( pkt \) is dropped and \( FW \) waits a new packet. If there is no matched condition, the rules are checked sequentially and the process is repeated until there are no more rules. If the packet no longer has a rule that matches
the condition, it returns to the packet receiving state \( FW \). The firewall specified with pACSR can be formally written in TLA+ in 3.2 subsection and checked about the desired properties using TLC in 3.3 subsection.

### 3.2 Translation of Firewall Rules to TLA+

The formal specification of TLA+ firewall rules begins with EXTENDS \( \text{Naturals, Sequences} \), which are TLA+ standard modules and define numeric and tuple operators respectively. The next line of EXTENDS statement declares constants and variables as shown in Fig. 2. Constants in the specification must be provided with values when checking the model.

In Fig. 3, the initial state \( \text{Init} \) defines that the state of firewall equals “rdy” indicating the ready state for receiving a packet and an arbitrary packet \( \text{pkt} \) is the element of the set of all received packets. The packet \( \text{pkt} \) is a record whose \( p \) and \( a \) fields are an element of \( \text{Packet} \) and (“allow”, “drop”) respectively. The dropped packets \( \text{pktQ} \) is the empty tuple because there are no dropped packets at the initial state. And the invariant \( \text{TypeInvariant} \) is defined that the state of \( \text{fw} \) equals “rdy” or “busy” representing the state processing a packet and \( \text{pkt} \) is the same as the initial state definition. The next action can be defined as an action that receives a packet at the firewall, \( \text{FWIn}(p) \), or processes the received packet, \( \text{Act} \). To receive a packet is possible only when a firewall is in a ready state, and the new packet is assigned to \( \text{pkt.p} \) meaning the field \( p \) of the packet record \( \text{pkt} \) and also a matching target for the firewall rules condition. The field \( a \) of \( \text{pkt} \) is not decided yet because it reflects a result of firewall rules check. So, the new state of \( \text{pkt} \), \( \text{pkt}' \), means that \( \text{pkt.p} \) only is changed and \( \text{pkt.a} \) is not changed. The expression is \( \text{pkt}' = [\text{pkt} \ EXCEPT \ !.p = p] \). For handling a packet, the firewall becomes “busy” state. Additionally, unlike the description in pACSR this paper describes \( \text{pktQ} \) queueing the dropped packet among the received packets. Actually the use of \( \text{pktQ} \) is a trick of this specification for model checking and at the stage receiving a packet, \( \text{FWIn}(p) \), there is no change in \( \text{pktQ} \) because receiving a packet doesn’t relate to dropping a packet.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>pACSR Firewall Rules in [3].</th>
</tr>
</thead>
<tbody>
<tr>
<td>( FW = {} ) ( \times ) ( \text{in}^2 \text{pkt} ). ( T(r_i, \text{pkt}) ) ( T(r, \text{pkt}) = \text{if } c_i(\text{pkt}) \rightarrow \text{Act}(a_i) ) else ( T(r_{i+1}, \text{pkt}) ) ( \text{for } 0 \leq i \leq n ) ( T(r_{n+1}, \text{pkt}) = FW ) ( \text{Act}(a_0) = \text{if } a_i = \text{&quot;allow&quot;} \rightarrow {} \times \text{out}? . FW ) else ( FW )</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2 The declarations in TLA+.

Fig. 3 The initial predicate and invariant.

In Fig. 3, the initial state \( \text{Init} \) defines that the state of firewall equals “rdy” indicating the ready state for receiving a packet and an arbitrary packet \( \text{pkt} \) is the element of the set of all received packets. The packet \( \text{pkt} \) is a record whose \( p \) and \( a \) fields are an element of \( \text{Packet} \) and (“allow”, “drop”) respectively. The dropped packets \( \text{pktQ} \) is the empty tuple because there are no dropped packets at the initial state. And the invariant \( \text{TypeInvariant} \) is defined that the state of \( \text{fw} \) equals “rdy” or “busy” representing the state processing a packet and \( \text{pkt} \) is the same as the initial state definition. The next action can be defined as an action that receives a packet at the firewall, \( \text{FWIn}(p) \), or processes the received packet, \( \text{Act} \). To receive a packet is possible only when a firewall is in a ready state, and the new packet is assigned to \( \text{pkt.p} \) meaning the field \( p \) of the packet record \( \text{pkt} \) and also a matching target for the firewall rules condition. The field \( a \) of \( \text{pkt} \) is not decided yet because it reflects a result of firewall rules check. So, the new state of \( \text{pkt} \), \( \text{pkt}' \), means that \( \text{pkt.p} \) only is changed and \( \text{pkt.a} \) is not changed. The expression is \( \text{pkt}' = [\text{pkt} \ EXCEPT \ !.p = p] \). For handling a packet, the firewall becomes “busy” state. Additionally, unlike the description in pACSR this paper describes \( \text{pktQ} \) queueing the dropped packet among the received packets. Actually the use of \( \text{pktQ} \) is a trick of this specification for model checking and at the stage receiving a packet, \( \text{FWIn}(p) \), there is no change in \( \text{pktQ} \) because receiving a packet doesn’t relate to dropping a packet.

Fig. 4 The definition of \( \text{Next} \) and \( \text{Spec} \).

As shown in Fig. 4, the specification of a firewall, \( \text{Spec} \), is expressed in a conjunction of \( \text{Init} \) and \( \text{Next} \) defined in a disjunction of \( \text{FWIn}(p) \) and \( \text{Act} \). \( \text{Init} \) and \( \text{FWIn}(p) \) of \( \text{Next} \) is
described in Fig. 3. \textit{Act} is enabled when the firewall is busy, meaning that there is a packet to be processed. \textit{Act}, as defined in pACSR, checks that the packet can be added to the dropped packet \textit{pktQ} which would be shown in model checking later. The other part of \textit{Act} is the same as in pACSR. If there is no matched condition, the rules are checked sequentially and the check is repeated until there are no more rules. If the packet no longer has a rule that matches the condition, then the packet is dropped. After checking the firewall rules, the state of a firewall becomes “rdy” so that a new packet can be received, but \textit{pkt} is not changed.

So far, we have described the specification of TLA+ firewall rules checking. Even though the specification is described mathematically, the symbols in TLA+ consist of very familiar symbols to us so that they are very easy to understand and learn.

The next subsection describes that TLC takes the specification explained in this subsection as input and performs it to check the firewall rules model.

3.3 Model checking of Firewall with TLC

TLC, TLA+ model checker, is performed on a model basis. Therefore the model has to be defined with TLA+ for model checking. In particular, this paper specifies the values of declared constant firewall rules as well as invariants for model. The values of constants and the invariants are expressed in accordance with TLA+ syntax. Fortunately, TLA+ toolbox provides an easy GUI environment to use for creating models.

<table>
<thead>
<tr>
<th>rule</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>\text{scrIP = 1} \rightarrow \text{drop}</td>
</tr>
<tr>
<td>R2</td>
<td>\text{dstIP = 2} \rightarrow \text{allow}</td>
</tr>
<tr>
<td>R3</td>
<td>\ast \rightarrow \text{drop}</td>
</tr>
</tbody>
</table>

This paper uses the example of firewall rules in [2] for model checking and TABLE 2 shows it. The rule R1, R2, and R3 in TABLE 2 are expressed in an ordered tuple in TLA+, and firewall rules checking is performed sequentially as a Rule definition in Fig. 4.

\[
\left(\left[ r \rightarrow \text{scrIP = 1}, a \rightarrow \text{drop}\right], \left[ r \rightarrow \text{dstIP = 2}, a \rightarrow \text{allow}\right], \left[ r \rightarrow \ast, a \rightarrow \text{drop}\right]\right)
\]

\[ \{"dstIP = 2","dstIP = 1","scrIP = 1"\} \quad (4) \]

The constant \textit{FWRule} declared in Fig. 2 is provided with the value defined as (3) expressing R1, R2, and R3. And the value of the constant \textit{Packet} is specified as (4). Since the subsection 3.2 is specified at a higher level, the abstract packet is defined as a set of values so that the value of it can match the rule straightforwardly. For example, “\text{scrIP = 1}” packet matches rule R1 and the packet is dropped because an action of R1 is “drop”. As a result, the value of the rule \( r \) and a packet \( p \) can be easily matched by defining in a same form. But the definition of a packet can be differently specified by the definition of Firewall rules.

For model checking, lastly, this paper describes the invariant limiting the length of \textit{pktQ} expressed as \( \text{Len(pktQ)} \). While checking a model, the dropped packet by rules is added to \textit{pktQ} and the length of \textit{pktQ} is checked. Assuming that the invariant is \( \text{Len(pktQ)} = 0 \), if at least one packet drops, it means that it does not satisfy the safety property and TLC informs the error occurrence to a user. Then a user can check why the packet was dropped by analyzing the packet in Error-Trace. Fig. 5 displays the Error-Trace screen provided in TLA+ toolbox.

![Fig. 5 The Error-Trace screen of TLA+ toolbox.](image)

In Fig. 5, TLC shows the Error-Trace of the firewall specification violating the invariants when being provided with values of the constants as described in this paper. That is, Fig. 5 shows that the invariant is violated because there is a dropped packet which is “\text{dstIP=1}” packet.

4 Related Works

SDN applications with errors need verification because they affect the entire network. There are many approaches for this problem, using the formal specification and verification.

Rupak Majumdar et al. [2], describe handling unboundedly many packets under the unbounded state space on network and unboundedly many control messages affected by them between the SDN controller and switches. The safety property relating to the issues of SDN is verified by Kuai, a distributed enumerative model checker. Murphi is used for describing the network topology and safety properties in SDN controller management. And the specification is taken as input by Kuai.

As another approach, Miyoung KANG et al. [3] and M.-K. Shin [4] propose VeriSDN as a technique for verifying the SDN firewall application. Miyoung KANG et al. are an advanced and concretized research from [4]. These papers
suggest VeriSDN, a technique for verifying firewall applications in SDN applications, and explain that formal modeling and verification techniques can be used at the early stage of OpenFlow design. The proposed VeriSDN uses pACSR to describe OpenFlow rules formally and verifies the described contents with VeriFM to help prevent errors in advance.

Zakharova et al. [5] also describe the formal approach to SDN, especially focusing on packet forwarding and flow tables which are defined as OpenFlow. And they introduce a tentative language for a specification of SDN forwarding policies and discuss model checking problems raised from a BDD-based toolset. The paper formally proves the problems if a controller of SDN is a finite state machine, then the model checking problem for SDN can be reduced to a finite model checking problem for PLTL.

The related works can be similar to this paper in that it performs a formal specification and model checking to improve reliability of SDN and describes the switch firewall among SDN applications. However, this paper suggests a method to formally write TLA+ which can be specified more easily and intuitively than Murphy, so it is meaningful that users who feel difficulty in formal specification can also use it.

5 Conclusions
This paper shows that it is possible to use the formal specification and formal verification as a way to increase the reliability of SDN which is receiving much attention now. Especially, the firewall among SDN applications could be formally written using TLA+ and the invariants of it are verified by using TLA+ specification in TLC, a model checking tool included in TLA+ toolbox. Since TLA+ is easier and more intuitive to understand than other formal specification languages, through the work in this paper it is very meaningful to mitigate the difficulty of users who want to apply the formal methods for their works.

However, we described firewall rules on a single switch. Since the formal modeling may vary depending on the network topology, an approach to the many switches should be presented and it is necessary to formally define a firewall in a state where a lot of switches are connected. In future work, we plan to improve formal modeling to reflect various network topologies.

Acknowledgment
This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2017-2015-0-00445) supervised by the IITP(Institute for Information & communications Technology Promotion.

References