Formal Modelling and Analysis of Software Defined Networks

H. K. Jnanamurthy and V.Varadharajan
Advanced Cyber Security Engineering Research Centre (ACSRC)
The University of Newcastle, Australia

July 10, 2019

1 Abstract

In cloud computing, software-defined network (SDN) gaining more attention due to its advantages in network configuration to improve network performance and network monitoring. SDN addresses an issue of static architecture in traditional networks by allowing centralised control of a network system. SDN contains centralised network intelligence module which separates a process of forwarding packets (data plane) from packet routing process (control plane). It is essential to ensure the correctness of SDN due to secure data transmitting in it. In this paper. Model checking is chosen to verify an SDN network. The Computation Tree Logic (CTL) and Linear Temporal Logic (LTL) used as a specification to express properties of an SDN. Then complete SDN structure is defined formally along with its Kripke structure. Finally, CTL properties are analysed against the SDN Kripke model to assure the properties of SDN is correct.

2 Introduction

There are many formal verification techniques [1] studied in recent years such as model checking, abstract interpretation and boolean satisfiability. Model checking

is widely used in many fields such as verification of hardware, software and security and safety protocols. In model checking, the system is modelled as a state machine and specifications (properties of the system) expresses in linear temporal logic and computation tree logic.

SPIN [2] and SMV [3] are famous model checkers, LTL specifications [4] are allowed to express in SPIN and CTL specifications [4] are in SMV. It is known that automata based verification can lead to state explosion problem. To address state explosion problem, symbolic model checking (SMC) [5] and abstract model checking (AMC) [6] has been developed with successful results. The combination of SMC and Binary Decision Trees (BDDs) [7] maximise the states in the system, but a bottleneck in manipulating the amount of memory required to store BDDs. Bounded Model Checking (BMC) [8] progresses fastly after SMC, the basic idea of BMC is to find a counterexample in executions whose length k. The BMC problem can quickly reduce to satisfiability problem, which can be verified using SAT solvers [9]. Modern SAT solvers can handle satisfiability problem with thousands of variables.

SDN [10] is gaining more attention due to its advantages in network configuration to improve network performance and network monitoring. So it is essential to ensure the correctness of SDN. In this article, we describe SDN by a Kripke structure along with specifications of the SDN. Then security properties of an SDN are expressed using temporal formulas. Then security properties of an SDN are analysed against a designed Kripke model.

3 Preliminaries

3.1 Kripke Structure

A Kripke structure M [11] is a tuple M=(Init, States, Transition, Labelling), where 'States' is the set of states which are defined by a set of propositions 'A' hold on the states, 'Init' $\subseteq$ 'States' is the set of initial states, 'Transition' $\subseteq$ 'States' $\times$ 'States' and 'Labelling' is a labelling function 'Labelling': 'States'$\rightarrow 2^A$.

3.2 Linear-Time Temporal Logic

Linear-time temporal logic, or LTL for short, is a temporal logic, with connectives that allow us to refer to the future. It models time as a sequence of states, extending infinitely into the future. This sequence of states is sometimes called a computation path, or simply a path. Linear-time temporal logic (LTL) has the following syntax given in Backus Naur form:

$$\phi ::= \top \mid \bot \mid \mathrm{p} \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\mathrm{X}\phi) \mid (\mathrm{F}\phi) \mid (\mathrm{G}\phi) \mid (\phi\ \mathrm{U}\ \phi) \mid (\phi\ \mathrm{R}\ \phi)$$

where p is any propositional atom from some set Atoms. Thus, the symbols $\top$ and $\bot$ are LTL formulas, as are all atoms from Atoms; and $\neg\phi$ is an LTL formula if ϕ is one, etc. The connectives X, F, G, U and R are called temporal connectives. X means 'neXt state,' F means 'some Future state,' and G means 'all future states (Globally).' The next two, U and R are called 'Until' and 'Release' respectively.

Definition 1: In a given model M, a path Σ is defined as a sequence of connected edges which connect nodes, lets say s_0, s_1, s_2. . .s_n in S are the nodes such that $\forall \mathrm{m} \geq 0, s_m \longrightarrow s_{m+1}$. The path $\Sigma = s_0, s_1, s_2 . . .s_n$ represents a sequence of nodes in a system M, we define Σ^i as starting from s_i, for example Σ^5 is s_5, s_6, s_7. . . s_n.

Definition 2: A satisfaction relation $\models$ is defined for the model M, considering paths Σ to check linear temporal logic (LTL) formula satisfies Σ. The satisfaction relation $\models$ over Σ and LTL formula is specified as:

- $\Sigma \models \top$, where $\top$ represents true
- $\Sigma \models \mathrm{q}$, iff $\mathrm{q} \in \mathrm{L(s)}$
- $\Sigma \models \mathrm{X}\psi$ iff $\Sigma^2 \models \psi$
- $\Sigma \models \mathrm{G}\psi$ iff $\forall \mathrm{m}\ \mathrm{m} \geq 1$, $\Sigma^m \models \psi$

- $\Sigma \models \mathrm{F}\psi$ iff $\exists$m m$\geq$1, $\Sigma^m \models \psi$
- $\Sigma \models \psi_1$ U ψ_2 iff $\exists$m m$\geq$1 such that $\Sigma^m \models \psi_2$ and $\forall$n, n=1, 2, 3...m-1 satisfies $\Sigma^n \models \psi_1$

3.3 Computation Tree Logic

Definition 3: Computation Tree Logic is a technique to represent time in a tree-like structure in which future is not determined; there are many paths in which 'actual' path is realised. The Backus Naur form of CTL formulas are defined as:

$\phi ::= \top \mid \bot \mid$ p $\mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\mathrm{AX}\phi) \mid (\mathrm{EX}\phi) \mid (\mathrm{AG}\phi) \mid (\mathrm{EG}\phi) \mid (\mathrm{AF}\phi) \mid (\mathrm{EF}\phi) \mid (\mathrm{A}[\phi$ U $\phi]) \mid (\mathrm{E}[\phi$ U $\phi])$

Definition 4: Let M = (S, $\rightarrow$, L) be a model for CTL, s in S, ϕ a CTL formula. The relation M, s $\models \phi$ is defined by structural induction on ϕ:

- M, s $\models \top$ and M, s $\not\models \bot$
- M, s $\models$ q iff q $\in$ L(s)
- M, s $\models \neg\phi$ iff M, s $\not\models \phi$
- M, s $\models \phi_1 \wedge \phi_2$ iff M, s $\models \phi_1$ and M, s $\models \phi_2$
- M, s $\models \phi_1 \vee \phi_2$ iff M, s $\models \phi_1$ or M, s $\models \phi_2$
- M, s $\models \phi_1 \rightarrow \phi_2$ iff M, s $\not\models \phi_1$ or M, s $\models \phi_2$
- M, s $\models \mathrm{AX}\phi$ iff for all s_1 such that s $\rightarrow s_1$ we have M, $s_1 \models \phi$. Thus, AX says: 'in every next state'.
- M, s $\models \mathrm{EX}\phi$ iff for some s_1 such that s $\rightarrow s_1$ we have M, $s_1 \models \phi$. Thus, EX says: 'in some next state'.
- M, s $\models \mathrm{AG}\phi$ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow$ ..., where s_1 equals s, and all s_i along the path, we have M, $s_i \models \phi$.

- M, s $\models$ AFϕ iff for some paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow ...$, where s_1 equals s, and all s_i along the path, we have M, $s_i \models \phi$.
- M, s $\models$ AFϕ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow ...$, where s_1 equals s, and there is some s_i such that M, $s_i \models \phi$.
- M, s $\models$ EFϕ iff for some paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow ...$, where s_1 equals s, and for some s_i along the path, we have M, $s_i \models \phi$.
- M, s $\models$ A[ϕ_1Uϕ_2] iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow ...$, where s_1 equals s, that path satisfies ϕ_1 U ϕ_2.
- M, s $\models$ E[ϕ_1Uϕ_2] iff for some paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow ...$, where s_1 equals s, that path satisfies ϕ_1 U ϕ_2.

4 Modelling of SDN

In this section, we present a formal definition of SDN and security properties which are necessary for SDN. SDN is a tuple SDN:= (P^t, W, C, T) where,

- A packet is a tuple (h, pLD) consists of a packet header 'h' and payload data 'pLD' information which is transmitted in a network. The packet header <st, dt, (α_1, α_1, . . α_n)> contains source address 'st', destination address 'dt' and packet pattern (α_1, . . α_n) which is used to match with ports of the switch during transmission in a network system.
- 'W' is a set of switches W=$\{w_1, w_2, w_3 ... w_n\}$, each switch is a tuple w:= (P, W^c, FR) where 'P' is a set of ports in a switch represented as P=$\{p_1, p_2, p_3 ... p_n\}$. A port 'p' either a input or a output port represented as p(ip, op), where 'ip' is a set of input ports and 'op' is a set of output ports. Each input and output port consists of port ID, which is used to forward a packet and drop a packet based on forwarding rules which are received from the controller. W^c is switch controlling software which handles matching functions to route

the packets to appropriate network devices. FR is a finite set of information to represents forwarding rules denoted as FR=$\{r_1, r_2, r_3...r_n\}$. 'SwitchTrust' is an assignment function 'SwitchTrust': W $\rightarrow$ Label, which assigns each switch with a label, more the number of labels in a switch is considered as a most trusted switch.

- SDN controller is a tuple C:= $\{M^h$, FC, δ, s_0^c, $S^c\}$ that manages control flow of the network based on Open-flow protocol.
  - S^c is a set of control states.
  - s_0^c is a set of initial control states, $s_0^c \subseteq S^c$.
  - M^h: h$\rightarrow h^{'}$ is a header modification function that modifies the packet header based on controller policies.
  - FC: h $\times$ N^s $\times$ PESA $\rightarrow$ FR is a forward rule calculator function, where h is a packet header, N^s: W^s $\times$ h$\rightarrow$ π is a network status, which is used to find a feasible path of a packet in a set of all possible paths π. Paths π are arranged based on feasibility to reach destination i.e. $\{\pi_0, \pi_1, \pi_2, ..., \pi_n\}\in\pi$, where π_0 ranks high to route a packet, π_1 ranks next of π_0.
  - $\delta \subseteq S^c \times M^h \times$ FC $\times S^c$ is a transition relation.
- Topology 'T' of an SDN is a binary relation T$\subseteq$((W $\times$ OP)$\times$(W $\times$ IP)) on switches and ports of the network system. The nodes of a network are represented by a relation (W $\times$ OP) and (W $\times$ IP). Furthermore, <h, p, w> denotes the packet state in the network, where 'h' is a packet header, and 'p' and 'w' is a port of a switch 'w'.

$$Trans(nodes) = \begin{cases} \bigwedge\limits_{\exists w \in W} ConfiG_w(< h, p >, < h, p >), & \text{if } nodes < 2. \\ \bigwedge\limits_{w \in W} ConfiG_w(< h, p >, < h, p' >), & \text{if } nodes \geq 2. \end{cases} \quad (1)$$

$$Trans(nodes) = \begin{cases} \bigwedge\limits_{\exists w \in W} ConfiG_w(< h, p >, < h, p >), & \text{if } nodes < 2. \\ \bigwedge\limits_{w \in W} ConfiG_w(< h, p >, < h', p' >), & \text{if } nodes \geq 2. \end{cases} \quad (2)$$

Once the topology of the network is ready, then the packet transmission relation among the switches is given in recursive relation. Equation 1 presents a packet transmission relation without modifying packet header 'h' from input port p to p'. Equation 2 presents a packet transmission relation with modifying packet header 'h' from input port p to p'. In both equations 1 and 2, the first condition of the equation presents a packet dropping node, where the number of nodes in a network is not sufficient to send a packet across the network. The second condition of the equations presents a transmission relation across the nodes of a network based on network configuration function ConfiG.

ConfiG is network configuration function, which assigns forwarding rules to switches, ConfiG: W $\rightarrow$ FR. Packets enter into an input port of a switch is configure and forwarded based on forwarding rules. A run of an SDN is a sequence of transmission nodes is present as:

$$run = (ConfiG_0, w_0) \xrightarrow{P_1^t} (ConfiG_1, w_1) \xrightarrow{P_2^t} ... \xrightarrow{P_i^t} (ConfiG_i, w_i) \xrightarrow{P_{i+1}^t} (*) \quad (3)$$

Pair $(ConfiG_i, w_i)$ is considered as routing configuration by a controller to the switch w_i. A run is a sequence of nodes in a network, which allows packet P_i^t is passing through in it.

5 Formal Specification on SDN Properties

Verification of SDN security properties is essential, and it enhances the confidence of the system. In this section, we present an SDN's Kripke structure along with specifications of SDN using LTL and CTL. Kripke structure of an SDN and its specifications are used in the verification process via model checking. Due to the advantages of Bounded model checking, we reduce the model checking problem with a bound, which can be solved by SAT solvers.

The Kripke structure of an SDN is a tuple SDN=(S_0, S, T, L) as shown in figure 1, where:

- S_0 is a set of initial states where, $S_0 \subseteq$ S.

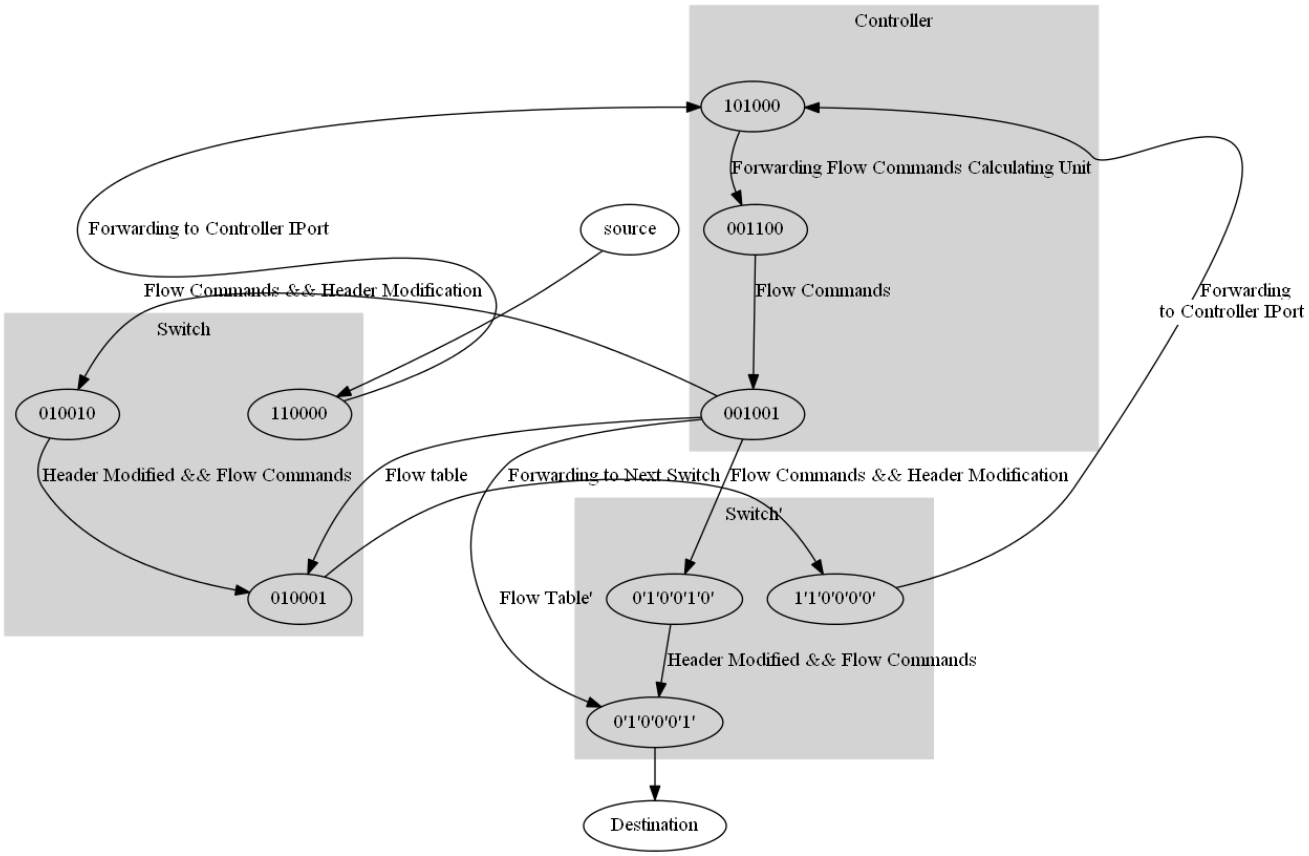

Figure 1: Kripke model for Software Defined Network

- 'S' is a set of states (S^c, S^w $\in$S, where S^c represents states belong to the controller and S^w represents states belong to switch).
- T $\subseteq$ S $\times$ S is a transition relation ({($S^c \times S^c$), ($S^c \times S^w$), ($S^w \times S^c$), ($S^w \times S^w$) } $\in$ T), where S^c and S^w is a set of states in a controller and switch.
- 'L' is a labelling function 'L': 'S'$\rightarrow 2^A$. In our model we used boolean vector <I, W, C , FC, M^h, O > $\in$ A. 'I' and 'O' are the bits to represent the input and output port of a switch and a controller. 'W' is a bit to represent a packet state in a switch and 'C' is a bit to represent the packet state in a controller. FC and M^h are the bits to represent states, where forward rules calculation and header modification happens.

Given a Kripke structure of SDN, the specifications of a system are expressed using an LTL or CTL formula 'f' and a bound 'd', and semantics of BMC can describe a process of constructing a propositional formula [SDN, $X_f]_d$.

Let (s_0, s_1, s_2, . . s_d) be a finite sequence of states in a path π. The description of a formula [SDN, $X_f]_d$ contains two components: SDN is a propositional formula that contains (s_0, s_1, s_2, . . s_d) and X_f is also a propositional formula to validate the constraints given in a formula 'f'. To define specification constraints X_f, we provide a definition of loop condition 'L^j' which is propositional formula that is true if there is a loop in path π. Loop condition is considered as valid if there is a transition from 'j' to previous states.

$$L^j := \bigvee_{i=0}^{j} (S_j, S_i) \tag{4}$$

For a Kripke structure SDN, and d $\geq$ 0,

$$[SDN]_d := Init(S_0) \wedge \bigwedge_{i=1}^{d-1} (S_i, S_{i+1}) \tag{5}$$

The complete formula, which includes an SDN model and specification is presented in a boolean formula [SDN, $X_f]_d$.

$$[SDN, X_f]_d := Init(S_0) \wedge \bigwedge_{i=1}^{d-1} (S_i, S_{i+1}) \ \wedge (\neg L^d \wedge [X_f]_d) \tag{6}$$

Consider a Kripke structure of an SDN in figure 2. Each state of an SDN represented by five-bit variables. We use b[5], b[4], b[3], b[2], b[1] and b[0], where b[5] is a high bit and b[0] be the low bit. The initial state of an SDN is represented as follows,

$$Init(S_0) := b[5] \wedge b[4] \wedge \neg b[3] \wedge \neg b[2] \wedge \neg b[1] \wedge \neg b[0] \tag{7}$$

The transition relation of an SDN is represented as follows,

$$
\begin{aligned}
T(s,s') := \{&(b[5] \wedge b[4] \wedge \neg b[3] \wedge \neg b[2] \wedge \neg b[1] \wedge \neg b[0] \\
&\wedge b'[5] \wedge \neg b'[4] \wedge b'[3] \wedge \neg b'[2] \wedge \neg b'[1] \wedge \neg b'[0]) \\
&\vee \\
&(b[5] \wedge \neg b[4] \wedge b[3] \wedge \neg b[2] \wedge \neg b[1] \wedge \neg b[0] \\
&\wedge \neg b'[5] \wedge \neg b'[4] \wedge b'[3] \wedge b'[2] \wedge \neg b'[1] \wedge \neg b'[0]) \\
&\vee \\
&(\neg b[5] \wedge \neg b[4] \wedge b[3] \wedge b[2] \wedge \neg b[1] \wedge \neg b[0] \\
&\wedge \neg b'[5] \wedge \neg b'[4] \wedge b'[3] \wedge \neg b'[2] \wedge \neg b'[1] \wedge b'[0]) \\
&\vee \\
&(\neg b[5] \wedge \neg b[4] \wedge b[3] \wedge \neg b[2] \wedge \neg b[1] \wedge b[0] \\
&\wedge \neg b'[5] \wedge b'[4] \wedge \neg b'[3] \wedge \neg b'[2] \wedge \neg b'[1] \wedge b'[0]) \\
&\vee \\
&(\neg b[5] \wedge \neg b[4] \wedge b[3] \wedge \neg b[2] \wedge \neg b[1] \wedge b[0] \\
&\wedge \neg b'[5] \wedge b'[4] \wedge \neg b'[3] \wedge \neg b'[2] \wedge b'[1] \wedge \neg b'[0]) \\
&\vee \\
&(\neg b[5] \wedge b[4] \wedge \neg b[3] \wedge \neg b[2] \wedge b[1] \wedge \neg b[0] \\
&\wedge \neg b'[5] \wedge b'[4] \wedge \neg b'[3] \wedge \neg b'[2] \wedge \neg b'[1] \wedge b'[0]) \\
&\vee \\
&(\neg b[5] \wedge b[4] \wedge \neg b[3] \wedge \neg b[2] \wedge \neg b[1] \wedge b[0] \\
&\wedge b'[5] \wedge b'[4] \wedge \neg b'[3] \wedge \neg b'[2] \wedge \neg b'[1] \wedge \neg b'[0])\}
\end{aligned}
\tag{8}
$$

5.1 Analyse with a Faulty Transition

We now add a faulty transition from state 101000 to state 001001 denote by T_f.

$$
\begin{aligned}
T_f(s,s') := T(s,s') \vee (&b[5] \wedge \neg b[4] \wedge b[3] \wedge \neg b[2] \wedge \neg b[1] \wedge \neg b[0] \wedge \neg b'[5] \wedge \neg b'[4] \wedge b'[3] \\
&\wedge \neg b'[2] \wedge \neg b'[1] \wedge b'[0])
\end{aligned}
\tag{9}
$$

Consider the primary property of an SDN, forward rules has to calculate

for every packet which is pass through the controller for routing. The property is represented as Gp, where p is $\neg b[5] \wedge \neg b[4] \wedge b[3] \wedge b[2] \wedge \neg b[1] \wedge \neg b[0]$. Using BMC, we generate a counterexample results witness of F¬p. The absence of such property indicates the SDN property is violated.

Consider a case where the bound d = 2 and unrolling the faulty SDN system transition relation in the following formula:

$$[[SDN]]_2 := Init(S_0) \wedge T_f(S_0, S_1) \wedge T_f(S_1, S_2) \tag{10}$$

The loop condition is represented as:

$$L^2 := \bigvee_{i=0}^{2} (S_2, S_i) \tag{11}$$

The formula on a path without loops:

$$\begin{aligned} [[F(\neg p)]]_2^0 &:= \neg p(S_0) \vee [[F(\neg p)]]_2^1 \\ [[F(\neg p)]]_2^1 &:= \neg p(S_1) \vee [[F(\neg p)]]_2^2 \\ [[F(\neg p)]]_2^2 &:= \neg p(S_2) \vee [[F(\neg p)]]_2^3 \\ [[F(\neg p)]]_2^3 &:= 0 \end{aligned} \tag{12}$$

Finally by substituting all terms and we get:

$$[[F(\neg p)]]_2^0 := \neg p(S_0) \vee \neg p(S_1) \vee \neg p(S_2) \tag{13}$$

By putting SDN transitions, loop condition and property together, we can get a new formula:

$$[[SDN, F(\neg p)]]_2 := [[SDN]]_2 \wedge (\neg L^2 \wedge [[F(\neg p)]]_2^0) \tag{14}$$

Since a missing path is sufficient to prove a violation of SDN property, the loop condition is excluded. This results in the following formula:

$$\begin{aligned} [[SDN, F(\neg p)]]_2 := [[SDN]]_2 \wedge [[F(\neg p)]]_2^0 : - \\ Init(S_0) \wedge T_f(S_0, S_1) \wedge T_f(S_1, S_2) \wedge (\neg p(S_0) \vee \neg p(S_1) \vee \neg p(S_2)) \end{aligned} \tag{15}$$

The assignment 110000, 101000, 001001 satisfies the above formula $[[SDN, F(\neg p)]]_2$; this assignment violates the fundamental property of an SDN.

5.2 Expression of SDN Properties using Temporal Logic

$AG(W_{ip} \rightarrow A(\neg W'_{ip}\mathbf{W}FC)$**:** For any state in SDN, it is not possible to send a packet to next switch without calculating and routed by forwarding rules.

$AG(C_{ip} \rightarrow AX(FC))$**:** For any state in SDN, where a packet enters the controller always calculate forwarding rules before sending it to the switch.

$AG(W_{ip} \rightarrow AX(C_{ip}))$**:** For any state in SDN, where a packet enters the switch always forward to the controller to calculate forwarding rules.

$AG\neg(W_{ip} \rightarrow AX(W_{op}))$**:** For any state in SDN, where a packet enters the switch's input port should not forward to an output port without calculating forwarding rules.

References

[1] M. C. Gaudel. Formal methods for software testing (invited paper). In *2017 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 1–3, Sept 2017.

[2] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[3] S. V. A. Campos. Ieee recommended practice for powering and grounding electronic equipment. (color book series - emerald book). In *Proceedings. XII Symposium on Integrated Circuits and Systems Design (Cat. No.PR00387)*, pages 98–101, 1999.

[4] C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal description logics: A survey. In *2008 15th International Symposium on Temporal Representation and Reasoning*, pages 3–14, June 2008.

[5] S. Ben-David, B. Sterin, J. M. Atlee, and S. Beidu. Symbolic model checking of product-line requirements using sat-based methods. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 189–199, May 2015.

[6] C. Tian and Z. Duan. Detecting spurious counterexamples efficiently in abstract model checking. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 202–211, May 2013.

[7] S. Reda, R. Drechsler, and A. Orailoglu. On the relation between sat and bdds for equivalence checking. In *Proceedings International Symposium on Quality Electronic Design*, pages 394–399, 2002.

[8] L. Giordano, A. Martelli, and D. T. Dupré. Achieving completeness in the verification of action theories by bounded model checking in asp. *Journal of Logic and Computation*, 25(6):1307–1330, Dec 2015.

[9] P. Arcaini, A. Gargantini, and E. Riccobene. How to optimize the use of sat and smt solvers for test generation of boolean expressions. *The Computer Journal*, 58(11):2900–2920, Nov 2015.

[10] A. Prajapati, A. Sakadasariya, and J. Patel. Software defined network: Future of networking. In *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pages 1351–1354, Jan 2018.

[11] Patricia Bouyer, Patrick Gardy, and Nicolas Markey. Quantitative verification of weighted kripke structures. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 64–80, Cham, 2014. Springer International Publishing.