Research Directions in Formal Verification of Network Configurations toward Verification of Mobile Networks

Hideki Sakurada[1,2] and Kouichi Sakurai[2,3]

[1] Nippon Telegraph and Telephone Corporation
me@hidekisakurada.com
[2] Kyushu University
sakurai@inf.kyushu-u.ac.jp
[3] Advanced Telecommunications Research Institute International

Abstract

This paper reviews current research trends in the formal verification of computer network configurations, specifically focusing on formal verification for software-defined networking (SDN). We explore the challenges encountered when applying formal verification, comparing its application to pre-SDN network verification efforts. Additionally, we discuss the potential application of formal verification in mobile networks. We first provide an overview of research on the formal verification of virtual LAN (VLAN) configurations, which predates the emergence of SDN. We next illustrate SDN and existing research applying formal verification to SDN. Finally, we briefly examine potential scenarios for applying formal verification to mobile networks.

Keywords: formal verification, network, configuration, SDN

1 Introduction

When constructing and managing a computer network, multiple network devices are installed, connected, and configured. To ensure that the network operates as expected, it is necessary to consistently configure the network devices. One method to confirm that the network is functioning as expected is to run and test it. However, it is practically impossible to test every possible communication. Furtermore, it requires to monitor the network while testing to detect any security-related issues such as data being received by unintended recipients.

To solve this problem, it is desirable to be able to statically verify the network based on information about the connection and configuration of network devices without actually running the network. While it is feasible to verify rather simple networks manually, it becomes difficult in general as the size of the network increases. Therefore, methods for modeling and verifying networks applying formal methods have been proposed.

In this paper, we survey a part of such research trends, particularly the research on formal verification targeting software-defined networking (SDN), and discuss the challenges when applying formal verification in comparison with work on formal verification of networks before SDN emerged, as well as application of formal verification to mobile networks.

We first outline research on formal verification of virtual LAN (VLAN) configuration, which was proposed before SDN emerged. We next illustrate SDN and existing research applying formal verification to SDN. Finally, we briefly discuss some scenarios for applying formal verification to mobile networks.

The 7th International Conference on Mobile Internet Security (MobiSec'23), Dec. 19-21, 2023, Okinawa, JAPAN, Article No. 64

2 Overview of Formal Verification

Formal verification is a method to rigorously verify that a system satisfies desired properties described in a mathematical or logical language. Such properties are often called formal specifications of the system. The methodology for formal verification and formal specification description, collectively referred to as formal methods, is an important part of software engineering methodologies.

There are various methods for formal verification, depending on the target system and the formal specifications. Some methods can verify any state transition system, and others can verify systems, e.g., implemented as computer programs written in the C programming language. Some methods can verify only simple properties but allow automatic and fast verification, and others may allow complex properties but may require significant efforts from users. Here we will only mention a few major general-purpose methods, although there are a lot of other general-purpose and special-purpose methods.

The verification methods using SAT solvers are basic methods of formal verification. A SAT solver is a tool that decides the satisfiability of logical formulas in propositional logic. State-of-the-art SAT solvers are found on the web page of SAT Competitions[1]. When using it for formal verification, it is necessary to describe the system and its properties as logical formulas in propositional logic. For instance, by describing the system and the specification as logic formulas S and P respectively, you can demonstrate that there is no case where P does not hold in the system's behavior by showing that logic formulas S and its negation $\neg P$ can not both be satisfied. While the satisfiability of propositional logic is a computationally hard problem, methodologies have been proposed that, for simple logical formulas, can achieve the computation within a realistic timeframe. In addition to SAT solvers, SMT solvers[2] are also used for formal verification and can decide the satisfiability of formulae written in some extensions of the propositional logic. Furthermore, SAT solvers and SMT solvers are used as building blocks of formal verification methods for more specific kinds of systems.

To verify systems that can be modeled as state transition systems, the SMV[3] and NuSMV[4][5][6] tools are often used. These tools verify properties written in temporal logics such as LTL and CTL. For example, with these tools, we can verify that a target system never reaches an undesirable state. Version 2 of NuSMV (NuSMV2) implements two verification algorithms, one of which encodes the verification problem into a SAT problem and calls an internal SAT solver to decide the problem.

The methods mentioned above are all automatic verification tools. To verify more complex properties of systems, semi-automatic methods are used. In such methods, general-purpose theorem provers such as Coq[7] and Isabelle[8]. With these tools, more complex systems and properties can be described in higher-order logics than the propositional and first-order logics. However, these tools usually can not automatically verify properties, and users must help the tools to find proofs of the properties. The correctness of the proofs is automatically checked by these tools.

3 Formal Verification of Network Configuration

To explain the formal verification of network configuration, we present a brief overview the Sakurada's work [9] that verifies network configurations using tagged VLANs. In tagged VLANs, in order to virtually (logically) divide a LAN into multiple VLANs, information called tags is added to the packet header for identification. Here, we explain based on the example in Figure 1.

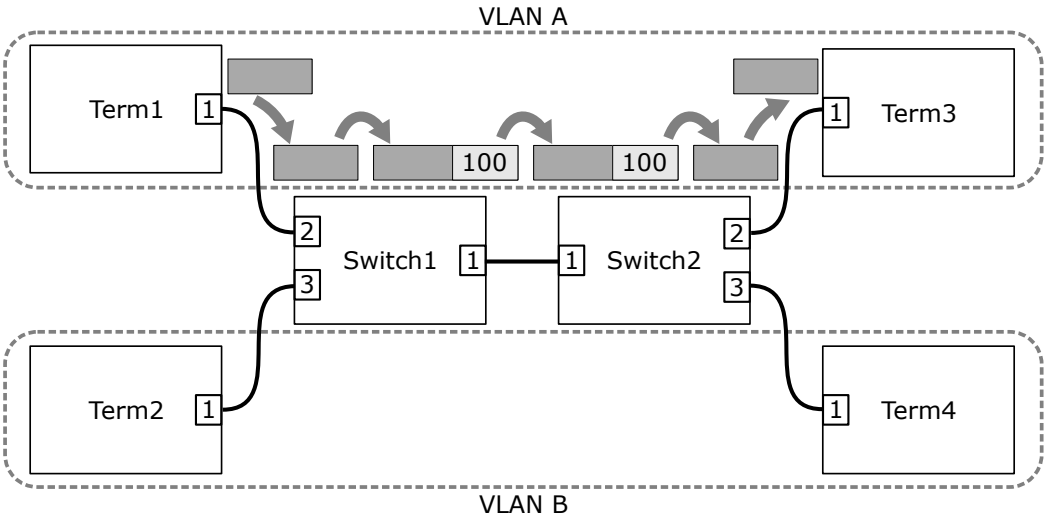

Figure 1: LAN divided into two VLANs

In this example, the LAN is divided into two VLANs, A and B, each of which is given a tag for identification. If the tag for VLAN A is 100, a packet sent from the terminal Term1 belonging to VLAN A will be received at port 2 of the network device Switch1. Switch1 recognizes it as a packet from VLAN A based on the port information and adds tag 100 before sending it to the network device Switch2. Switch2, based on the tag information, identifies it as a packet from VLAN A and forwards it to port 2, where devices belonging to VLAN A are connected. The packet is then received by the terminal Term3 belonging to VLAN A.

However, if there is a misconfiguration in Switch2 and the packet is mistakenly sent from port 3 instead of port 2, not only will Term3 not be able to receive it, but information will also leak to the terminal Term4 belonging to VLAN B. To detect such network issues caused by misconfigurations, one possible method is to send packets and check which port they reach. However, to do this, a means to monitor all network device ports would be necessary. Therefore this research uses the formal verification technique introduced below to perform verification by simulation.

In this research, first, the state of a packet is abstracted and represented by a quadruple of variables (node, port, tag, phase), each representing the network device or terminal that holds the packet, the port number on the device where the packet is being sent or received, the VLAN tag, and the state the packet is in after being received (incoming) or after processing (outgoing).

Next, a state transition model of packets is constructed. The state transitions include the movement of packets between devices and terminals via cables and the transition where a received packet is sent according to the device's configuration. In the latter transition, the tag attached to the packet is rewritten. For example, the state transition model corresponding to the network in Figure 1 is depicted as shown in Figure 2, in which the states of the packets belonging to VLAN B are omitted.

The properties that the network should satisfy are described as properties of this state transition model, using Computation Tree Logic (CTL). For example, if a packet in the state at the bottom left of Figure 2 should not reach the terminal Term4, which does not belong to the same VLAN, we use the CTL formula and "! EF node = Term4," where "EF" means that at

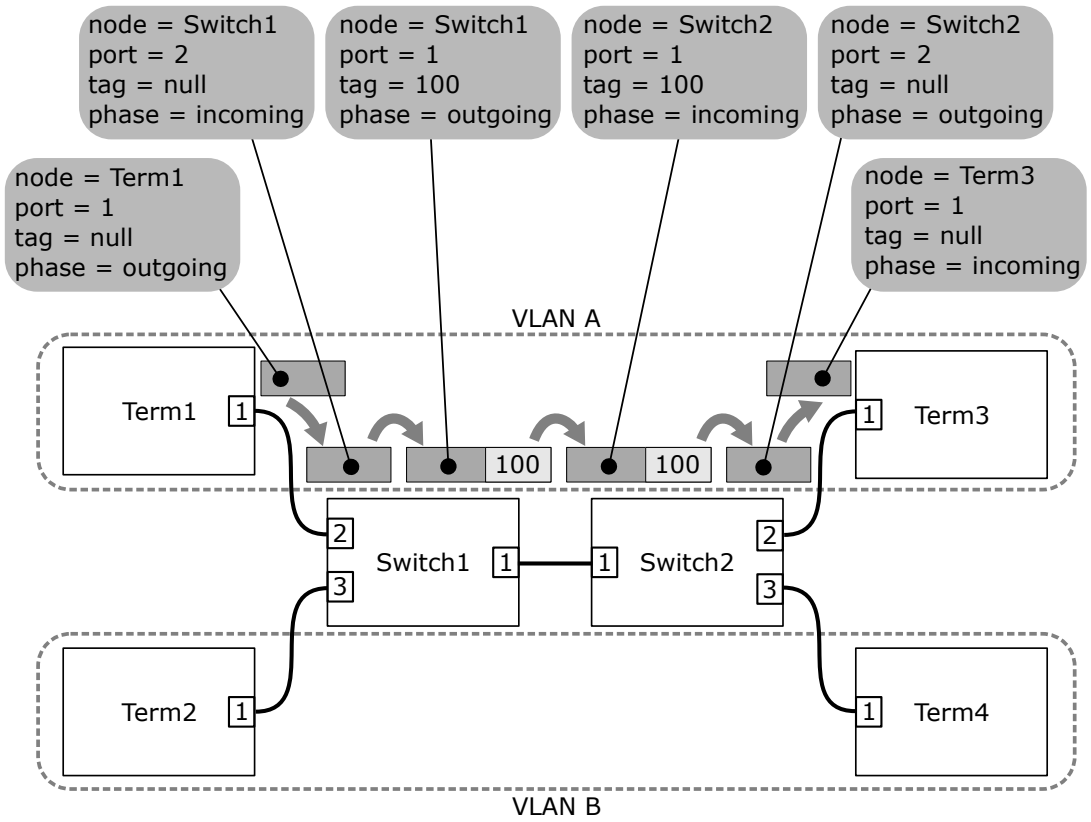

Figure 2: State-transition model of VLANs

some point in a state transition path from the initial state, the equation "node = Term4" holds, and "!" is the negation symbol. It should be noted that other logics such as Linear Temporal Logic (LTL) can also be used to describe properties of the state transition model.

The properties can be verified using a technique called model checking. Model checking tools such as SMV[3][10] and NuSMV[4][5][6] allow us to verify properties described in CTL or LTL automatically. Such tools may also output counterexamples when properties do not hold, providing clues as to where device configuration errors may occur.

Although the work described here focuses on tagged VLANs, the approach can be extended to IP networks.

4 Software-Defined Networking (SDN)

We provide an overview of SDN. Although SDN takes on various forms, we focus on the common elements necessary for formal verification, following the reference [11]. SDN is a network controlled by software that is separated into a control layer and a forwarding (infrastructure) layer, as shown in Figure 3. In SDN, a control device (SDN controller) in the control layer controls multiple network devices (switches) in the forwarding layer. The control layer also operates based on software and operator requests through an API.

The SDN controller controls the forwarding layer by writing packet forwarding rules to the

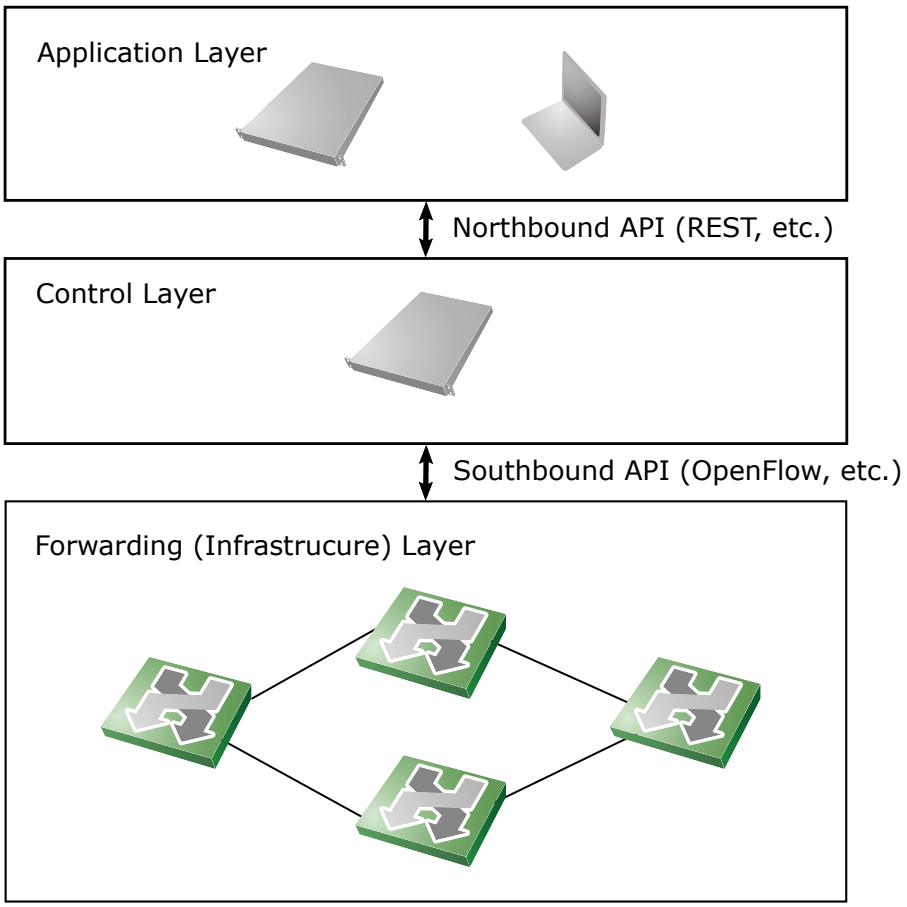

Figure 3: Diagram of SDN (created based on [12])

switches through a southbound API. Protocols like OpenFlow [13] are used as southbound APIs. In addition to the packet forwarding rules, OpenFlow defines messages that allow the SDN controller to send specific packets to the switches and messages that enable the switches to forward received packets to the SDN controller. With these messages, the SDN controller can directly control the switches and send and receive packets, e.g., for topology detection and latency measurement in the forwarding layer. Furthermore, software in the application layer sends requests to the control layer through a northbound API. The northbound API has been discussed by the Open Networking Foundation, which developed the specifications for OpenFlow. However, there is currently no standardized protocol, and it depends on the implementation of the SDN controller.

Compared to traditional network construction without SDN, the configuration of conventional network devices, especially IP routers, was performed by operators using command-line interfaces, and routing protocols were used to aggregate routing information from surrounding networks and calculate the actions (forwarding rules) for packets based on that information. In SDN, only the forwarding process based on the forwarding rules is performed by the devices in the forwarding layer, while the processing that involves the aggregation of routing information

using routing protocols and the calculation of forwarding rules based on that information are offloaded to the SDN controller.

SDN is used in various cases, including network operation and management within data centers [14]. In a data center, where numerous users and services are accommodated on the same physical infrastructure, virtualization of servers and networks is commonly implemented. Since manually changing the settings of physical network devices through the command-line interface every time is inefficient, centralized control using SDN is employed. Moreover, SDN allows us to automatically make necessary changes to network devices for service provisioning in response to requests from the application layer through the control layer.

In addition to its flexibility in network management, SDN also offers cost benefits. SDN controllers can be implemented using open-source controller software. Furthermore, we can use relatively inexpensive network equipment known as white-box (bare-metal) switches, designed and manufactured by combining generic components and not shipped with a switch-specific operating system (switch OS). These switches can then have an open-source switch OS installed for utilization.

5 Application of Formal Verification to SDN

In this section, we discuss several existing studies that apply formal verification to SDN and compare them to the method described in Section 3 for a better understanding. Since there are numerous research works on the application of formal verification to SDN, we defer an exhaustive introduction to other references [15][16][17].

The application of formal verification to SDN can be divided into verification of the control layer and verification of the forwarding layer. Since the latter is more similar to the method described in Section 3, we first see it.

5.1 Verification of Forwarding Layer

In SDN, similar to VLANs, failures such as packet loss can occur due to misconfigurations and inconsistencies in switch settings. The verification of the forwarding layer aims to ensure that such misconfigurations do not occur.

In SDN, it can be assumed that the SDN controller maintains the information about the network topology. Furthermore, the SDN controller can retrieve the forwarding rules configured on the switches. This allows us to easily collect the information required for modeling the network, as discussed in Section 3. However, since SDN may forward packets depending on header information, including VLAN tags, IP addresses, and IP ports, verification methods must handle these types of header information.

FlowChecker [18] is a verification method similar to the approach discussed in Section 3. It uses the information available to the SDN controller to create a state transition model of packets and verify properties such as reachability using symbolic model checking.

Another method proposed earlier is Anteater [19][20]. Anteater uses propositional logic formulas to describe the state transition model of packets and the desired properties (such as loop detection, packet loss, and configuration inconsistencies). It then uses a SAT solver to determine the satisfiability of these formulas and detect any faults. While it is possible to extend the method discussed in Section 3 to IP networks, it requires incorporating the information of the headers as parameters in the model. As a result, when we use header information that has not been used previously, the model construction method needs to be extended accordingly. To address this issue and handle header information more generally, a method called Header

Space Analysis [21] has been proposed. This method represents sets of packets using the bit patterns of their headers. Similar to regular expressions for representing sets of strings, it provides operations such as union, intersection, complement, and difference of the bit patterns. After constructing a state transition model of the bit patterns based on the information of the topology and forwarding rules, it then verifies reachability, detects loops, and examines leakage in virtual divisions (slices) similar to VLANs.

The methods mentioned so far may take longer to perform verification as the size of the network increases. Therefore, it becomes challenging to perform instant verification when making configuration changes. To address this issue, methods such as NetPlumber [22] and VeriFlow [23] define invariant conditions that the network should satisfy and verify that these conditions are maintained based on the differences between new and old configurations. Such verification based on differences cannot be performed using general-purpose model-checking tools. To accomplish this, research on the model-checking methodology itself is necessary.

5.2 Verification of Control Layer

In the verification of the control layer, the SDN controller is validated to configure switches correctly in response to inputs from the application layer and to ensure the correct operation of the forwarding layer. If the correctness of the operation, namely the properties to be verified, can be defined, then verification of the forwarding layer can be conducted.

Kuai [24] models the control and forwarding layers as a single system based on the SDN controller program and performs verification. The SDN controller and switches are modeled as finite-state machines asynchronously exchanging messages, abstracting communication through the southbound API. The sequences of transitions of these finite-state machines are verified to satisfy the given properties. By incorporating the behavior of the SDN controller into the model and allowing for asynchronous communication, this method can also discover potential issues that may arise from the order in which the changes made by the SDN controller to the settings of multiple switches are reflected. This method corresponds to incorporating the operation of the SDN controller into the state transition model in Section 3, but the number of states in the model becomes extremely large. Kuai uses a state reduction technique called partial order reduction, commonly used in the research area of concurrent system verification.

All the methods introduced so far are applicable when the topology is specifically determined, but there are also methods to verify the SDN controller program in a topology-independent manner. Guha et al. [25] propose a method to verify the behavior of an SDN controller that translates inputs from the application layer into OpenFlow commands. The inputs from the application layer are described as a program written in a language called NetCore. This program instructs each switch on the actions to be taken when receiving a packet, such as which port to forward the packet to and its associated conditions. The SDN controller converts the program into a sequence of OpenFlow commands. Then, the verification is performed by ensuring that the forwarding layer operates as instructed by the NetCore language program when the SDN controller's output configures the switch. To achieve this, the formal semantics of the NetCore language, the transformation rules of the SDN controller, and the meaning of the sequence of OpenFlow commands are described as certain logical formulas in the general-purpose theorem-proving system Coq [7], and their relationships are proved in Coq. Since automatically performing verification based on such general definitions is difficult, the verification is carried out by manually describing the proofs in Coq and using Coq's capabilities for partial automation and correctness checking. Furthermore, from the proofs written in Coq, extracting an OCaml [26] program is possible, resulting in an SDN controller software with

rigorous proofs.

5.3 Toward Formal Verification for Mobile Networks

In this section, we briefly discuss the application of formal verification to the configuration of mobile networks.

Similarly to SDN, 4G/5G mobile networks also separate the control plane and the data plane. In 4G/5G mobile networks, they are referred to as C-plane and U-plane, respectively. The C-plane not only controls the U-plane but also has various functions such as authentication and session management of mobile devices such as smartphones. Figure 4 shows a conceptual diagram of a 5G mobile network system. In the U-plane, user equipment (UE) such as a smartphone is connected to the core network through the radio access network (RAN) and communicates with data networks (DN) such as the internet through the user plane function (UPF). In the C-plane, there are various functions controlling the network, such as the access and mobility management function (AMF) performing access authorization and authentication. The significant difference between SDN in Section 4 and 5G systems is that the C-plane has

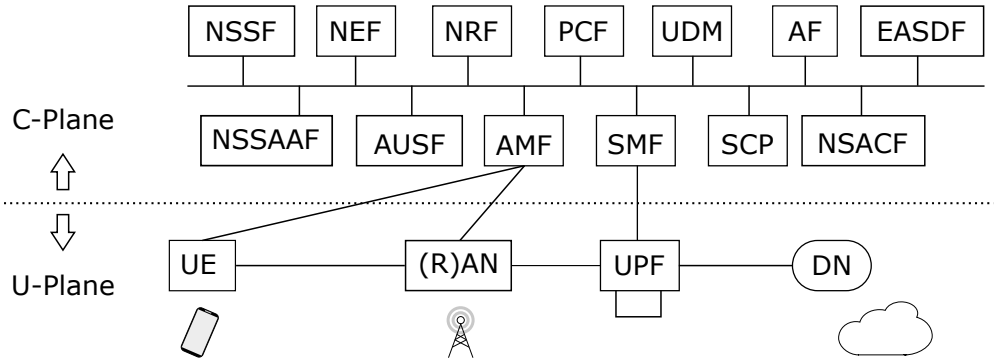

Figure 4: 5G System Architecture [27]

various functions such as authentication, authorization, billing, session management, terminal calling from the network side, and data prioritization control. Furthermore, very low latency as well as high speed is required in 5G systems.

We consider the following scenarios for applying formal verification to mobile networks such as 5G.

The first scenario is similar to the verification of controllers discussed in Section 5.2. We verify behaviors of some network functions instead of controllers in SDN. For example, we verify the consistency of network functions collaboratory controlling the U-plane, considering network functions operating simultaneously for multiple UE. However, unlike in Section 5.2, the impact of U-plane operations by network functions goes beyond packet reachability, so a different approach from the one introduced in Section 5.2 is required.

The next scenario focuses on specific services provided by mobile networks. For example, there is a technology called network slicing, in which a 5G network is virtually divided into multiple networks. These virtual networks provide different service levels such as latency and speed. The security requirements for services using network slicing have been discussed in a document [28] by NSA and CISA, and the requirements stated in the document are properties that should be verified. FlowChecker in Section 5.1 allows us to check the leakage of packets

over slices, although it does not specifically take mobile networks into account. One may apply a similar method for mobile network slicing.

Finally, in cases where low-latency communication is desired within the mobile network, a replacement for UPF called SRv6 MUP has been proposed and implemented. Instead of going through the UPF, which may be located far from UE, it allows UE to communicate through shorter paths. It uses a method called segment routing, which allows flexible path specification through source routing. Since segment routing is a mechanism different from manipulating forwarding rule in Section 4, the formal verification of SDN controlled by segment routing is an interesting research topic.

6 Conclusion

In this paper, we outlined some of the verification methods for SDN from the perspective of whether the VLAN verification method in Section 3 can be extended to SDN. We also briefly discussed the challenges in network configuration verification, particularly for mobile networks.

References

[1] The international SAT competition web page. http://www.satcompetition.org/.

[2] The satisfiability modulo theories library. https://smtlib.cs.uiowa.edu/.

[3] The SMV system. http://www.cs.cmu.edu/~modelcheck/smv.html.

[4] Nusmv: a symbolic model checker. https://nusmv.fbk.eu/.

[5] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model verifier. In N. Halbwachs and D. Peled, editors, *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499, Trento, Italy, July 1999. Springer.

[6] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[7] The Coq Proof Assistant. https://coq.inria.fr/.

[8] The Isabelle web page. https://isabelle.in.tum.de/.

[9] Hideki Sakurada. Model checking configurations for tag VLAN (in Japanese). *IPSJ Journal*, 47(7):2247–2257, jul 2006.

[10] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 1020 states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[11] Larry Peterson and Carmelo Cascone and Brian O'Connor and Thomas Vachuska and Bruce Davie. *Software-Defined Networks: A Systems Approach*. Systems Approach LLC, January 2021.

[12] Diego Kreutz, Fernando M V Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined networking: A comprehensive survey. *Proc. IEEE*, 103(1):14–76, January 2015.

[13] Open Networking Foundation. *OpenFlow Specifications*. https://opennetworking.org/software-defined-standards/specifications/.

[14] Kazuya Suzuki, Hideyuki Shimonishi, Yasunobu Chiba, Yasuhito Takamiya, Kazushi Sugyo, Yoshihiko Kinkai, and Shuji Ishii. OpenFlow and its appliactions (in Japanese). *Computer Software*, 30(2):2_3–2_13, 2013.

[15] Satochi Yamazaki, Yutaka Yukawa, and Toshio Tonouchi. A survey of verification technology for SDN/OpenFlow by applying formal methods (in Japanese). *Computer Software*, 33(2):2_26–2_42, 2016.

[16] Nitin Shukla, Mayank Pandey, and Shashank Srivastava. Formal modeling and verification of software-defined networks: A survey. *International Journal of Network Management*, 29(5):e2082, 2019. e2082 nem.2082.

[17] Alireza Souri, Monire Norouzi, Parvaneh Asghari, Amir Masoud Rahmani, and Ghazaleh Emadi. A systematic literature review on formal verification of software-defined networks. *Transactions on Emerging Telecommunications Technologies*, 31(2):e3788, 2020. e3788 ETT-18-0271.R3.

[18] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*, SafeConfig '10, page 37–44, New York, NY, USA, 2010. Association for Computing Machinery.

[19] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 290–301, New York, NY, USA, 2011. Association for Computing Machinery.

[20] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. *SIGCOMM Comput. Commun. Rev.*, 41(4):290–301, aug 2011.

[21] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 113–126, San Jose, CA, April 2012. USENIX Association.

[22] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 99–111, Lombard, IL, April 2013. USENIX Association.

[23] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. VeriFlow: Verifying Network-Wide invariants in real time. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, Lombard, IL, April 2013. USENIX Association.

[24] Rupak Majumdar, Sai Deep Tetali, and Zilong Wang. Kuai: A model checker for software-defined networks. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*, pages 163–170. ieeexplore.ieee.org, October 2014.

[25] Arjun Guha, Mark Reitblatt, and Nate Foster. Machine-verified network controllers. *SIGPLAN Not.*, 48(6):483–494, June 2013.

[26] The OCaml Programming Language. https://ocaml.org/.

[27] 3GPP. *3GPP TS23.501: System architecture for the 5G System (5GS)* . https://www.3gpp.org/ftp/Specs/archive/23_series/23.501.

[28] The National Security Agency (NSA) and the Cybersecurity and Infrastructure Security Agency (CISA). *5G Network Slicing: Security Considerations for Design, Deployment, and Maintenance*, 2023.