SDN Intent-based conformance checking: application to security policies

Nicolas Herbaut, Camilo Correa, Jacques Robin, Raul Mazo

SDN Intent-based conformance checking: application to security policies

Nicolas Herbaut , Camilo Correa , Jacques Robin
Centre de Recherche en Informatique
Université Paris 1 Panthon-Sorbonne
Paris, France

Raul Mazo
Lab-STICC
ENSTA Bretagne
Brest, France

***Abstract*—With the popularity of software defined networking architectures, the growing complexity of its use cases dictates the need for better auditability especially for security. In this paper, we aim at facilitating high-level management-plane policy configuration conformance auditing and their reflection in the data plane, to detect missing or spurious flow rules with respect to security policies. To this end, we propose an efficient conformance checking approach based on an intentional northbound interface as well as traces of management, control and data plane. Leveraging a proof-of-concept implementation of our approach, we compare its conformance-checking runtime and precision against a direct method on virtual topologies and find that it significantly improves scalability. We conclude by proposing directions for further enhancements extending the techniques presented herein.**

***Index Terms*—Software-defined networking, Intent-based networking, security, conformance checking,**

I. Introduction

Modern technological infrastructure deployments are of a substantively more complex nature than those that preceded them. With systems that can number in the thousands of servers and virtualized instances, new network management paradigms have emerged. Among them one finds software defined networks (SDN) [1], whose aim is centralizing and simplifying the administration of large and potentially heterogeneous infrastructure. However, SDN also bring about novel security challenges causing organizations to adopt strategies to both preempting and responding to cybersecurity incidents notably post-mortem analysis through security policy conformance checking.

To this end, organizations deployed technologies to collect all possible incident data so that new knowledge can be derived from tangible evidence of what has occurred. Since the network is one of the most common attack vector auditing discrepancies between high-level security policies and their concrete implementation in Forwarding Devices (FD) is of paramount importance [2]. However, network traces can be massive in size and quite low-level, making its analysis difficult. In this respect, Intent-based networking offers a very interesting approach by providing a high-level abstraction to specify policies rather than mechanisms [3].

This work was also supported by the project C4IIoT funded by the European Commission under Grant Agreements No. 833828. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

In order to engineer such auditing capabilities in SDN, we must tackle the following research questions:

1) How to verify that the data plane flows conform to management plane decisions without missing or spurious flow rules?
2) How to improve conformance checking performance with management rules expressed in a high-level abstract form?

The rest of the paper is organized as follows. We first give some technical background for our approach in Section II. We then present the key features of our architecture and the technical challenges that it addresses in Section III. We proceed to evaluate and discuss the experiment that we conducted and its result in Section IV before concluding with perspectives of future improvements in Section V.

II. Background/Related work

In this section, we shade light on the research questions, and give the intuitions driving the construction of our approach.

In this paper, we employ the term conformance checking to refer to the verification of high-level rule enforcement through execution traces, as it is commonly done in the business process community. We consider cases where the management plane does not have a deep understanding of the lower-level intricacies of the network and its models are thus too abstract to be directly used in traditional model checking [4]. This is often the case when decisions are taken by artificial intelligence [5]. Our proposal extends the concept of Intent Assurance [6], where the network controller provides functions and interfaces to assess that the network adhere and comply with the intents, to the management layer.

In classical SDN Architectures, the management plane is responsible for faults, monitoring and configuration management operations of the network [7]. Its responsibilities also include orchestrating the full lifecycle of the enterprise Information System, infrastructure resource provisioning, security control, and compatibility with third-party systems.

Once a decision has been passed to the control plane, it is often necessary to monitor the actual implementation of the management configuration change. One possibility is to use data plane centric solutions for collecting measurement data such as NetFlow [8] or OpenSketch [9] for production

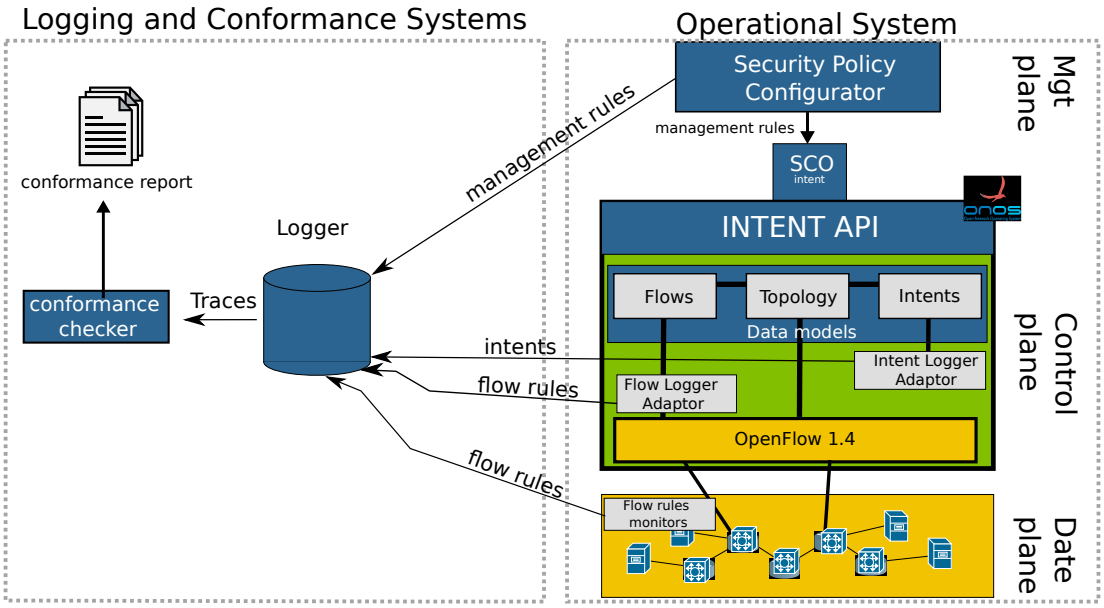


Fig. 1: High-Level Architecture

settings or Open VSwitch monitor for lab settings. In a large-scale scenario, however, the enormous quantity of information collected might be counterproductive to really understand the relationship between the low-level configuration of the FDs and the real business cases motivating each rule. Just as decentralized control plane has been proposed to tackle with the scalability issues [10], distributed monitoring started being integrated in production-ready SDN controllers [11] to this end.

In this paper, we propose an Intent-based approach [3] to collect traces to verify that management plane original intentions are correctly implemented in the data plane. By creating intent primitives that include the management plane business objectives in their design, we can achieve both ambitions of minimizing the quantity of information stored and making its interpretation easier for auditing purposes.

Finally, determining the level of abstraction of our monitoring construct is of paramount importance. Several options can be considered, depending on the targeted domain. For technical domains, such as software-defined wireless connectivity, Coronado et al. [12] adopted programming abstraction allowing administrators to specify how a precise portion of the flow space shall be treated in the wireless access segment were proposed. For regulatory domains, Ujcich et al. [13] proposed a GDPR-aware intent to enforce regulatory requirements for system and network design.

In this paper, we aim at assessing if a security and connectivity policy decided in the management plane is well deployed in the data plane. To this end, we designed a simple host-based ban list. It can be easily implemented by composing built-in intents developed for traffic management provided by the SDN controller. In the following sections, we demonstrate how Intent-based security-policies can be used to achieve conformance checking.

III. Auditable Network Management Model

Figure 1 presents the high-level architecture of our model. It comprises a) an *Operational System* that carries out traffic management, at both the control and forwarding planes and b) the *Logging and Conformance System* that bring the auditability features into the picture.

A. Operational systems

This set of components follow the usual Layers and Architecture Terminology from RFC 7426, and also support Intent-based Networking.

1) 3-Tiered SDN Infrastructure: In this system, the security policy configurator maintains a high-level view of its business objectives, possibly provided through algorithms, artificial intelligence or regulatory constraints. It keeps its models synchronized with the logging and conformance systems by publishing the same management rules used to configure the SDN apps deployed on the SDN Controller.

On the control plane, the SDN Controller is at the intersection of network management and data plane. It hosts SDN applications extending its features from different domains ranging from connectivity to security. Applications rely on the controller northbound interface (e.g., REST API) to manipulate the various data models (Flows Model, Topology Model and Intents Model).

Once an SDN app modifies the controller data models (e.g., submit an intent, deploy a flow), the device drivers translate it into instruction sent in the protocol supported by the FD (e.g., OpenFlow) so that the data plane can successfully forward traffic. Two logger adaptors are deployed in the modular runtime system of the SDN controller. Each adaptor subscribes to changes the SDN controller data models and populates the logger database/ with updates, respectively intents for the intent logger adaptor and Flows for the flow logger adaptor.

In the data plane, monitoring of flow-rule updates of FD can be achieved by subscribing to flow tables. In our model, flow-rule monitors are responsible for sending flow rules updates to the logging and conformance systems.

2) Intent-Aware SDN Controller: Intent-based networking [14] allows defining high-level network specifications that are enforced at the Controller level, without having to manage the flow-level configuration. When an intent is deployed to the SDN controller, it is compiled according to the current topology. If the compilation succeeds, it updates the controller flows models and subsequently triggers the deployment of the computed flow rules on the FDs. In case of topology changes, such as a device or link addition or removal, intents are recompiled and flow rules are redeployed accordingly, reducing the logic needed at the SDN app level.

If the network topology is not able to accommodate the intent anymore, the recompilation fails and the SDN app deployed within the controller can handle the issue and possibly deploy a new intent e.g., to reroute the traffic to another available back-up host. In any case, should the intent fail to deploy, the management plane can be informed to take action.

In our model, a custom Security and Connectivity (SCO) SDN application relies on the controller intent API to update the data models according to the Security Policy Configurator management rules. Intents are particularly useful for conformance checking operations between the management plane and the control place, since they can be expressed in a high-level form with similar semantics.

Now that we have covered the SDN infrastructure, we move on to the components responsible for persisting traces of modifications in the management, control and data plane and establishing the conformity between them.

B. Logging and Conformance Systems

These systems support both the collection and analysis of the data used for network policy conformance reports, highlighting discrepancies between management rules and data plane flows. We use a logging mechanism to store the network data which raise several issues such as scalability, security and performance. Although the design of such a logging mechanism is important, it is beyond the scope of this paper and is left for future work.

1) Intents & Management Rules: Employing intents at the control plane level is applicable to a vast range of use cases. We implemented a proof of concept of a network security policy system providing connectivity of hosts in the data plane using management rules containing white listed subnets and black listed hosts and host-to-host connections.

This kind of simple use case is common in industrial plants to logically segregates a compromised host from the rest of the network, while maintaining connectivity between trusted hosts. Listing 1 shows an example of management rules. As we can see, the intents of the management rules are easily understandable, as they allow any two host on the `s1` subnet to communicate while preventing host `h2` to communicate with any other host. It also precludes hosts `h3` and `h4` exchanging any traffic.

Method Name	Complexity	Comment
Direct method (III-B2a)	$\mathcal{O}(KLN^2)$	K: the average number of simple paths between 2 hosts, L the average size of the K-shortest path, N: the number of nodes
Indirect method (III-B2b+III-B2c)	$\mathcal{O}(N^2+F)$	N: the number of nodes, F: the number of flows

TABLE I: Conformance checking algorithm complexity assuming simple path lookup complexity of $\mathcal{O}(1)$

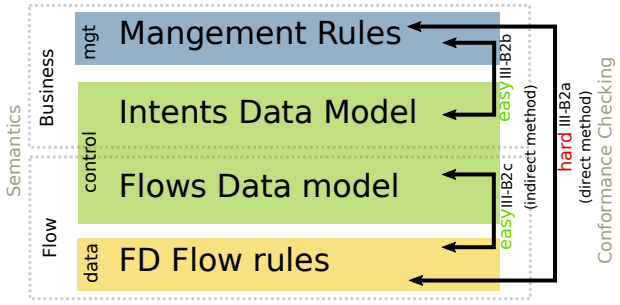


Fig. 2: Conformance checking operations crossing semantic domains is hard, and easy within semantic domains.

```
allow s1
block h2
block-from-to h3 h4
```

Listing 1: SCO (Security and Connectivity) Configuration example

Having shown an example of management rules, we will now demonstrate how one ensures that such rules get indeed implemented down the data plane level.

2) Conformance checking strategies: Due to lack of space, we only provide the intuitions of the conformance checking algorithms[1] and complexity of the main steps in table I.

a) Flow-based conformance of the management rules: (direct method III-B2a) to prove that a blocked host or blocked links are isolated from the rest of the network, we need to make sure that no FD form the data plane allows access to the incriminated hosts. At the other end of the spectrum, proving that connectivity intents are respected requires to actually find a valid flow between the two hosts. This involves finding the k-shortest paths between the hosts and testing them one by one until we find a valid one.

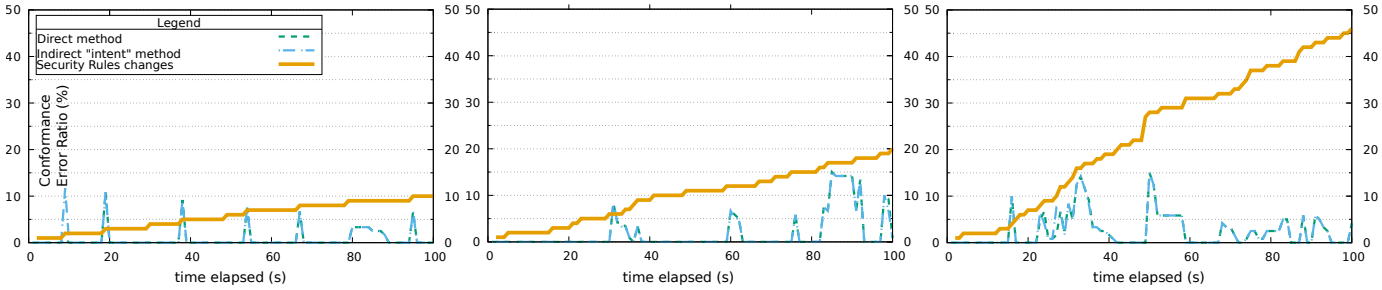
Since management rules and flow rules use different semantics, checking the conformity between these rules is difficult and compute intensive.

b) Intent-based conformance of the management rules: (indirect intent method III-B2b) Conversly, conformance checking between management rules and intent data models is made easy by the fact that they are expressed with the same semantic. In fact, SCO simply states that two hosts should be able to communicate by issuing a Host-to-Host intent

[1]code and details for this work, including conformance checking algorithms available https://github.com/nherbaut/netsoft-2021-paper

		FatTree(3)	FatTree(4)	FatTree(5)	FatTree(6)
Direct method	**III-B2a**	**30**s	**36**s	**1152**s	**>3h**
Indirect method	**III-B2b**	**0.3**s	**0.4**s	**0.4**s	**1.2**s
	III-B2c	**0.1**s	**0.1**s	**0.1**s	**0.2**s

TABLE II: Run times comparison for conformance checking report generation, following scenario IV-B



(a) 10s between reconfigurations (b) 5s between reconfigurations (c) 2s between reconfigurations

Fig. 3: Missing and spurious flow detection error ratio

containing both hosts IP address. When deploying this intent in the intent data models, SCO informs the SDN controller that it should list the k-shortest path between the host, pick one and finally try to deploy the flow rules on the FDs accordingly. If the flow rules are not successfully deployed (e.g., due to a saturated flow table or a FD down along the path), they must be rolled back and another path should be tried next until no path remains.

Intent-based conformance checking involves looking up the intent list composed of host pairs to make sure that no intent grant connectivity to a blocked host. It also involves verifying that for each non-blocked host pair, there exists an intent. Since we used the SDN controller built-in intents to implement SCO, development efforts are kept minimal.

c) Flow-based conformance of the FD flow rules: (III-B2c) is also easy, since flow data models are expressed with the same semantic as flows rules on the FDs. We simply need to check that for each FD, the flows are identical to the SDN Controller's corresponding flow data models.

C. Conformance Checking

To successfully find discrepancies between management rules and flow rules, we can apply the direct conformance checking algorithm III-B2a. It verifies whether the management rules are consistent with the FD flow rules. This task is difficult, as discussed earlier and illustrated in Figure 2, because it involves verification across semantic boundaries.

We propose to use intents to circumvent this issue and use an indirect method III-B2b+III-B2c, e.g., checking a) whether the management rules are consistent with the intent models and b) whether the flow data models conform with the FDs flow rules. Theoretically, direct and indirect methods are equivalent, since the controller intent engine is supposed to make intents models consistent with the flow model at all times (i.e. (1) for every intent, the corresponding flows are also in the data models and (2) no flow is present in the flow data models that have not been created by an intent deployment). If this assumption holds, having intents conforming to the management rules implies that the flow data models also conform the management rules. However, this does not always hold empirically: SDN controllers data models are only eventually consistent, since flow deployments following a management rules update are not done immediately, due to the delay caused by the intent installation.

In the next section, we evaluate if upon configuration changes, the SDN controller data models converge fast enough to actually use intents for conformance checking.

IV. Preliminary Evaluation and Discussion

A. Experimental testbed

We deployed SCO on top of an ONOS 2.4 SDN Controller. We used Mininet 2.2.2 to generate fat tree data center topologies of order K. We used an E-2176G Xeon machine with 32 GB of RAM to run our implementation and the conformance-checking. The flow rules monitors applications were built on top of Open vSwitch monitoring.

B. Test Scenario

The experiment consisted in attaching the virtual topology to the ONOS controller, and letting hosts periodically connect to each other through ICMP. Once the intents were installed and the flows deployed by the SCO app, we simulated management rules updates, to artificially generate both types of conformance checking errors. Each L second ($L \sim \text{Poisson}(\lambda)$), a host was chosen at random in the topology. If the host was not blocked by the security policy, we added it to the list of blocked hosts and tested if the corresponding flows were removed, i.e., that no spurious flow rules remained on the data plane. Conversely, if the chosen host was blocked in the current management rules, we unblocked it and made sure that the flow rules were correctly deployed, i.e., that the set flow

rules providing connectivity between the two hosts were not missing.

Traces were collected by the logger adaptors and transmitted to the logger by the security policy configurator app, the logger adaptors and the flow rules monitors whenever a data model change was detected. After a 100s runtime, we ran a custom Python3 conformance-checker tool implementing the methods described in Section III-C.

Runtime was also recorded to empirically confirm the complexity of the different strategies of conformance checking.

C. Results

1) Conformance checking performances: Table II shows that for typical Datacenter Fat-Tree topologies, flow-based conformance checking (the direct methodIII-B2a) is intractable for size $K > 5$, requiring several hours to run. On the other hand, the intent-based indirect method returns results in a matter of seconds. We conclude that performing conformance checking with intents (III-B2b) is the only viable option for medium or large topologies. Algorithm (III-B2c) runtimes are negligible compared to (III-B2a) and (III-B2b).

2) Direct vs. Indirect Method Precision Comparison: For Intent-based conformance checking to be useful, we need to make sure that it provides similar results than the flow-based ones. To verify this assumption, we run experiments for FatTree Topologies with $K = 4$ and computed the number of conformance rules violation with direct and indirect methods while injecting new management rules in a random fashion with a mean arrival time of 10, 5 and 2 seconds.

Results are presented in Figure 3 where we show the conformance error ratio corresponding to the connectivity breach and spurious flows.

Direct and indirect conformance checking methods lead to very similar error detection: curves are almost overlapping in Figure 3 and the maximum delta between them is 10.8 % with an average of 0.11% over the three experiments. We observed that the cases where the conformance result of both methods differed was always a case of missing connectivity undetected through indirect method. This difference can be explained by the extra processing required by intents to install flows, causing the intent models to be incorrectly labeled as conforming in step III-B2b. In these cases, the discrepancy was always resolved in less than 1s.

This experiment shows that conformance checking can be applied to intents instead of flows with a marginal and transitory impact on precision for the specified topologies on our test bed. It offers a better scalability thanks to the semantic proximity between intents and management rules.

Note that the conformance checking algorithm in this paper performs a complete revalidation for all the applicable intents and flow configuration at any point in time. This approach could be optimized, as proposed by Ujcich et al. [15], by recording the provenance and evolution of intents for each change in the applications requests or the state of the network. Our proposal can also be used to detect Intent drifts [6], caused by changes in intent realization over time.

V. Conclusion and Future Work

In this paper, we presented a new method to improve the auditability of programmable networks. We developed a proof of concept for auditing security rules and we relied on Intent-based networking to make the problem more tractable. We simulated the implementability of our approach on a virtual testbed, and discussed some performance aspects of Intent-based conformance checking.

In *Industrial Internet of Things*, a key motivation to use SDN is productivity optimization and predictability [16]. This why overcoming it is the priority of our current work in H2020 project "*Cybersecurity for the IIoT*". In future work, we plan to investigate how to implement the logger in a distributed fashion with a blockchain to make is more scalable and tamper-proof. Moreover, we plan to extend security policy intents beyond vulnerability minimization to also include mitigation to suspected attacks.

References

[1] D. Kreutz, F. M. V. 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.

[2] NIST, "SP 800-53 Rev.4 - Security and Privacy Controls for Federal Information Systems and Organizations," *National Institute of Standards and Technology*, vol. 800-53, pp. 1–460, 2014.

[3] O. N. F. I. Nbi, P. Onf, and C. Janz, "Intent NBI Definition and Principles," pp. 1–23, 2015.

[4] A. Souri, M. Norouzi, P. Asghari, A. M. Rahmani, and G. Emadi, "A systematic literature review on formal verification of software-defined networks," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 2, p. e3788, 2020.

[5] T. A. Khan, A. Mehmood, J. J. D. Ravera, A. Muhammad, K. Abbas, and W.-C. Song, "Intent-based orchestration of network slices and resource assurance using machine learning," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–2.

[6] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-Based Networking - Concepts and Definitions," Internet Engineering Task Force, Internet-Draft draft-irtf-nmrg-ibn-concepts-definitions-03, Feb. 2021.

[7] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (SDN): Layers and architecture terminology," in *RFC 7426*. IRTF, 2015.

[8] B. Claise, "Cisco Systems NetFlow Services Export Version 9," *Request for Comments*, 2004.

[9] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *NSDI'13*, 2013, pp. 29–42.

[10] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multi-domain SDN controllers," in *IEEE/IFIP NOMS 2014*. IEEE Computer Society, 2014.

[11] W. Kim, J. Li, J. W.-K. Hong, and Y.-J. Suh, "Ofmon: Openflow monitoring system in onos controllers," in *NetSoft*. IEEE, 2016, pp. 397–402.

[12] E. Coronado, R. Riggio, J. Villa1ón, and A. Garrido, "Lasagna: Programming abstractions for end-to-end slicing in software-defined wlans," in *WoWMoM*. IEEE, 2018, pp. 14–15.

[13] B. E. Ujcich and W. H. Sanders, "Data protection intents for software-defined networking," Tech. Rep., 2019.

[14] Y. Han, J. Li, D. Hoang, J. H. Yoo, and J. W. K. Hong, "An intent-based network virtualization platform for SDN," in *CNSM 2016*. IEEE, jan 2017, pp. 353–358.

[15] B. E. Ujcich, A. Bates, and W. H. Sanders, "Provenance for intent-based networking," in *2020 NetSoft*. IEEE, 2020, pp. 195–199.

[16] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-defined industrial internet of things in the context of industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, 2016.