Provenance for Intent-Based Networking

Benjamin E. Ujcich*, Adam Bates*, and William H. Sanders*†
*University of Illinois at Urbana-Champaign, Urbana, Illinois, USA
†Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
E-mail: {ujcich2, batesa}@illinois.edu, sanders@cmu.edu

Abstract—**Intent-based networking (IBN) promises to simplify the network management and automated orchestration of high-level policies in future networking architectures such as software-defined networking (SDN). However, such abstraction and automation creates new network visibility challenges. Existing SDN network forensics and diagnostics tools operate at a lower level of network abstraction, which makes intent-level reasoning difficult. We present PROVINTENT, a framework extension for SDN control plane tools that accounts for intent semantics. PROVINTENT records the provenance and evolution of intents as the network's state and apps' requests change over time and enables reasoning at multiple abstractions. We define an intent provenance model, we implement a proof-of-concept tool, and we evaluate the efficacy of PROVINTENT's explanatory capabilities by using a representative intent-driven network application.**

Index Terms—**intent-based networking, IBN, software-defined networking, SDN, data provenance, network visibility, network diagnostics, network troubleshooting, root cause analysis, PROV**

I. INTRODUCTION

Intent-based networking (IBN) has been proposed as a promising approach for automated and policy-aware network management. IBN can extend the programmable functionality found in software-defined networking (SDN) by allowing practitioners to specify *what* policies they want their network to implement rather than *how* their network's underlying mechanisms will implement such policies. This declarative approach—specified through *network intents*—allows for the simplification and abstraction of complex network management [1]. For instance, a practitioner can specify declaratively a network intent with a policy of "allow hosts *A* and *B* to communicate with bandwidth capacity *X*" without having to understand any mechanism details about the underlying device (*i.e.,* switch), topology, or forwarding configurations [2].

Although IBN abstracts away mechanism and configuration details to reduce the management complexity, such abstraction creates new visibility and insight challenges for practitioners who are tasked with the analysis of and explanations about past network activity [3], [4]. First, network intents are designed to allow for practical realization of multiple alternative implementation approaches [1]. A practitioner will want to understand which approach was implemented (*e.g.,* to validate policy optimizations) and why any alternative approaches were not implemented. Second, it may not be possible to install network intents immediately because the necessary resources may not be available [1]. A practitioner will want to understand what the network state looked like at a given time. Finally, network intents may interfere with the current network state or previous network intents [1], and that may induce broader network failures [3]. A practitioner will want to understand the extent to which the network reacted, and verify or validate the enforcement of policies [3].

The aforementioned challenges point to a greater need for explainable network activities that permit practical visibility and insight into past network state. Recent network control plane forensics and diagnostics tools, particularly for SDN [5], [6], are promising because they can track the complex data and process dependencies (or the *provenance*) of network control plane activities. However, from an IBN perspective, such tools that focus on low-level network state and configurations can easily become too complex to understand and, as such, do not provide the appropriate level of abstraction.

In this paper, we present PROVINTENT, a framework extension of SDN-based control plane provenance tools that accounts for IBN semantics and concerns. PROVINTENT annotates SDN control plane dependency and provenance graphs through an overlay of intent state provenance and metadata. Those annotations bridge the semantic gap between high-level intent abstractions and low-level network mechanisms and configurations. As a result, PROVINTENT's intent provenance model and PROVINTENT's implementation alongside SDN tools can be used by practitioners to reason effectively at different levels of abstraction about IBN activities. PROVINTENT simplifies what practitioners need to understand about intents by concisely identifying what low-level network state such intents affected. Our contributions include 1) an intent provenance model that articulates intent states and semantics (Section III); 2) a design and implementation of the intent provenance model that use the ONOS SDN controller's Intent Framework (Section IV); and 3) an evaluation of a representative intent-driven network application that shows how such provenance information can be practically useful (Section V).

II. BACKGROUND & RELATED WORK

A. IBN and SDN

Major SDN controllers, such as the Open Network Operating System (ONOS) [7] and OpenDaylight (ODL) [8], implement IBN capabilities that abstract the details of specific network protocols, mechanisms, and topology [3]. Intents allow network applications or practitioners to specify declaratively *what* the network ought to do at a policy level, rather than *how* the network ought to achieve the desired configuration.

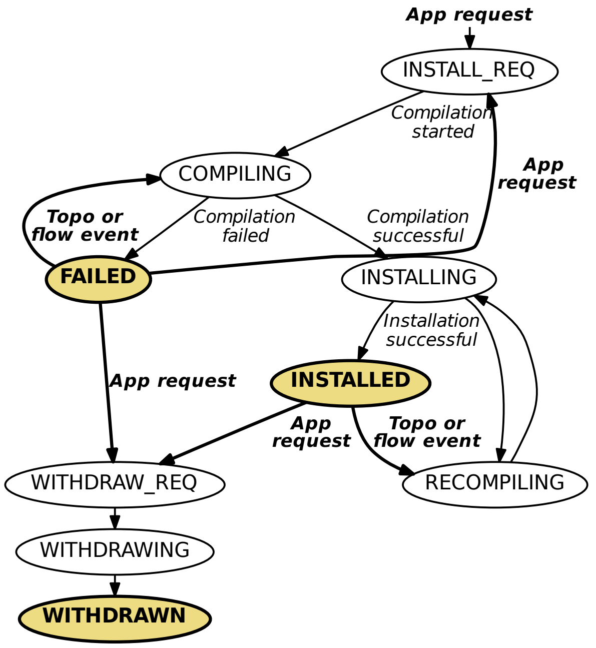

Fig. 1. Intent state machine of ONOS's Intent Framework [3]. Highlighted nodes represent long-lasting states, and bolded edges represent asynchronous state changes instigated from network state changes or app requests.

Given that intents' low-level implementations may react and change depending on network changes or practitioner requests, intents maintain a lifecycle notion and can be modeled succinctly as *intent state machines*. As a representative example, Figure 1 shows the ONOS Intent Framework's intent state machine. Nodes represent *intent states* of the intent lifecycle, and edges represent triggered events or actions that cause intent states to change (*e.g.,* the submission or withdrawal of an intent by a practitioner through an app, or a network topology change). As a result, intent frameworks can respond to any low-level changes that affect high-level intents.

Intent managers coordinate intents by listening to any relevant state changes induced by apps or the network. For instance, a topology change event (*e.g.,* TopologyEvent in ONOS) will be received by the intent manager so that it can check whether any installed intents are affected; if they are, the intents are recompiled and reinstalled. Intent managers also check whether state changes allow for previously failed intents to be successfully installed. Finally, intent managers can install and withdraw intents as needed by practitioners or apps.

B. Existing Network Insight Tools for IBN and SDN

Prior work [9]–[12] proposes unstructured logging for SDN controllers as a way to record past network activities to gain insight and for auditing. However, unstructured logging suffers from two general challenges: 1) the granularity and extent of logging information varies depending on the SDN controller implementation[1]; and 2) subsequent entries (of reported actions) in the log appear causally dependent on all preceding entries in the log, and that can create spurious dependencies.

[1]For instance, ONOS v1.14.0 [13] requires that the practitioner set the logging verbosity level to DEBUG or TRACE in order to expose details that relate to intent state changes or intent metadata.

Graph-based logging approaches mitigate such challenges by structuring past data usage, data generation, and activity dependencies into a *provenance graph* (or a *dependency graph*). PROVSDN [5] and FORENGUARD [6] use such graph structures to track the evolution of SDN control plane activities (*e.g.,* events and API calls) and data structures (*e.g.,* representations of control plane objects). Control plane state and functionality written in Network Datalog can be expressed graphically, as well [14], [15]. Similarly, GitFlow [16] tracks SDN flow rule changes through a version control system with an underlying graph structure. However, all of the aforementioned tools and approaches focus on low-level SDN network configurations and objects. That makes it difficult to use those tools for IBN reasoning because they expose too much complexity and do not explicitly track intents' evolutions.

III. INTENT PROVENANCE MODEL

We propose an intent provenance model. The model captures the essential semantics of intents' states, actions that influence any intent state transitions (*i.e.,* asynchronous network state changes or app requests), and the intent requirements (*e.g.,* flow matching criteria and bandwidth). PROVINTENT uses the intent provenance model to generate an *intent provenance graph*, which allows practitioners to reason about past activities at the intent abstraction and to understand low-level network actions as necessary. We have designed the intent provenance model to be interoperable with existing graph-based network provenance frameworks, such as those used in PROVSDN [5] and FORENGUARD [6].

PROVINTENT's intent provenance model is based on the W3C PROV data model [17]. PROV includes a graphical representation in terms of a *provenance graph*. Such a graph's nodes represent the agents (*i.e.,* system principals), activities (*i.e.,* events or processes), and entities (*i.e.,* data structures) of a given system. Such a graph's edges represent the relations among agents, activities, and entities. For instance, entities are "generated by" activities, activities "use" entities, and activities are "associated with" agents. PROV provenance graphs are directed and acyclic, which enables backward and forward tracing over such graphs to analyze root causes of events or understand information flow [5].

We now define the nodes, edges, and semantic meanings of PROVINTENT's intent provenance model.

1) Nodes: PROV includes the prov:Activity class (or Activity) to specify activities. In the intent provenance model, we extend that class to include the notion of an intent state, denoted by provIntent:IntentState (or IS). Each IS object also includes relevant metadata within application-specific key–value pairs. For instance, such metadata include the state name, the intent key (*i.e.,* persistent identifier), the time stamp of the last state transition, the relevant app that submitted the intent, and any requirements (*e.g.,* bandwidth capacity or routing specifications) that the intent should fulfill.

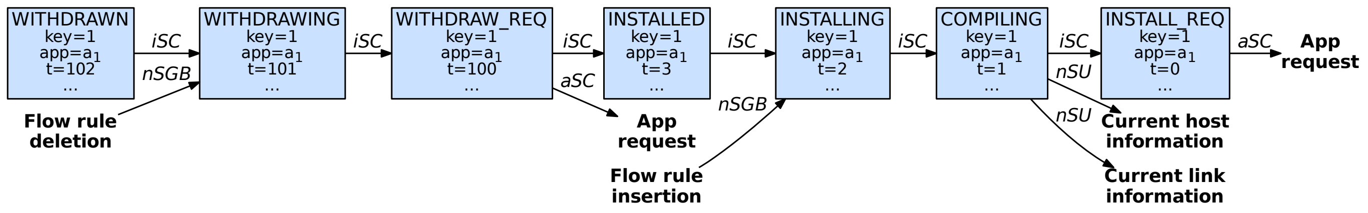
WITHDRAWN key=1 app=a_1 t=102 …
WITHDRAWING key=1 app=a_1 t=101 …
WITHDRAW_REQ key=1 app=a_1 t=100 …
INSTALLED key=1 app=a_1 t=3 …
INSTALLING key=1 app=a_1 t=2 …
COMPILING key=1 app=a_1 t=1 …
INSTALL_REQ key=1 app=a_1 t=0 …
iSC
nSGB
aSC
nSU
Flow rule deletion
App request
Flow rule insertion
Current host information
Current link information

Fig. 2. PROVINTENT intent provenance graph showing a simple intent state evolution. Each blue rectangle represents a provIntent:IntentState object. At time $t = 0$, an app a_1 requests that an intent be installed. The intent is installed at $t = 3$ and remains installed until its withdrawal at $t = 100$. PROVINTENT links any relevant SDN control plane activities (*i.e.,* prov:Activity objects) during each intent state; such activities are represented as bolded text.

2) Edges: PROV includes prov:wasInformedBy relations among Activity objects, which are represented as backward-directed[2] edges. We extend that relation class as follows:

1) provIntent:intentStateChange (or iSC) represents the transitions between intent states (domain: IS, range: IS)
2) provIntent:networkStateUsed (or nSU) represents a network activity that induced a change to the intent's state, such as a topology change (domain: IS, range: Activity)
3) provIntent:networkStateGeneratedBy (or nSGB) represents the effects of an intent's state change on the network state, such as a flow rule installation (domain: Activity, range: IS)
4) provIntent:appStateChange (or aSC) represents an app that induced an intent's state change, such as an intent installation (domain: IS, range Activity)

We note that the intent state machine captures the *possible* states and transitions of intents. Although such states (*i.e.,* IS) and transitions (*i.e.,* iSC) are central to the intent provenance model, we also model how they interact with the underlying SDN control plane API calls (*i.e.,* nSU and nSGB) and app requests (*i.e.,* aSC), too. The resulting graph shows how the intent actually transitioned among states over time.

3) Semantics: Figure 2 shows a representative intent provenance graph that we use to explain the model's semantics.

a) Intent evolution: We can represent an intent's evolution in an intent provenance graph as a path of IS nodes, linked together with iSC edges. That succinctly captures the state transitions that the intent took in the intent state machine. If an app submits an intent that can be realized in the network, the expected evolution would include the intent's installation request (*i.e.,* INSTALL_REQ), its compilation (*i.e.,* COMPILING), its installation (*i.e.,* INSTALLED), and eventually its removal (*i.e.,* WITHDRAW_REQ, WITHDRAWING, and WITHDRAWN). Figure 2 shows that evolution, starting from the most recent state (*i.e.,* WITHDRAWN) and tracing backward in the intent's history.

b) Intent mapping: The nSU, nSGB, and aSC edges serve as the mappings between the high-level intent abstraction and the low-level network state abstraction. These edges link to and from the activities generated by the SDN control plane provenance tool (*e.g.,* PROVSDN). Although the intent provenance model does not specify the exact SDN control plane provenance model to be used, we note that PROVSDN and FORENGUARD are capable of providing the appropriate control plane semantics. For instance, PROVSDN specifies agents (*e.g.,* apps), activities (*e.g.,* API calls), and entities (*e.g.,* control plane data structures). Similarly, FORENGUARD[3] specifies activities (*e.g.,* events and function calls) and entities (*e.g.,* OpenFlow messages and variable fields). Figure 2 shows how an intent provenance graph links to the API calls, represented as bolded text, that PROVSDN collects when such calls get the current network state (*e.g.,* a topology change) or change the network state (*e.g.,* flow rule insertion).

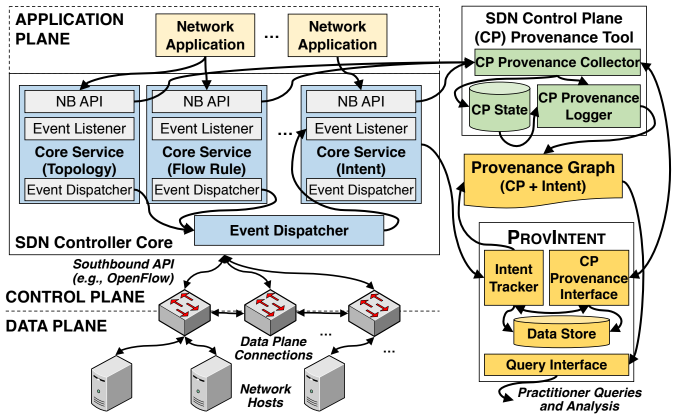

Fig. 3. Overview of PROVINTENT's design with major components and dependencies within the SDN and IBN frameworks.

IV. SYSTEM DESIGN AND IMPLEMENTATION

A. Design and Major Components

PROVINTENT complements and extends existing SDN control plane provenance frameworks. Figure 3 shows the overall design and major system components.

1) SDN controller: The SDN controller manages the intent-based network. The controller exposes a northbound API (or NB API) by which apps can query core services (*e.g.,* topology, flow rule, and intent services). The controller dispatches core services' events to apps interested in such events.

[2]In PROV, an edge represents a past causal relation. Thus, edges should be interpreted as backward-directed in time and logical ordering.

[3]FORENGUARD graphs are specified with forward-directed edges, as opposed to backward-directed edges as in PROV. To implement PROVINTENT with FORENGUARD, one would need to reverse the edge directions.

2) SDN control plane provenance tool: The SDN control plane provenance tool collects provenance metadata for all control plane activities in the SDN controller.

3) Intent tracker and data store: The intent tracker manages intents' states and uses the intent data store to keep track of ProvIntent's internal system state.

4) Control plane provenance interface: The control plane provenance interface interacts with the control plane provenance tool to coordinate any provenance relations and state information. For instance, if the controller's intent service calls the flow rule service's NB API to add a flow rule, the control plane provenance interface will be able to link the control plane provenance with ProvIntent's intent provenance.

5) Provenance graph: The provenance graph is the combined control plane and intent provenance output. If no SDN control plane provenance tool is enabled or available, ProvIntent tracks intent evolution only.

6) Query interface: The query interface allows a practitioner to query and return the relevant provenance graph. For instance, a practitioner can use the graph as input to a separate (automated) reasoning tool for root cause analysis.

B. Implementation

We implemented ProvIntent within ONOS v1.14.0 [13] and used a modified version of ProvSDN [5] as the SDN control plane provenance tool. We note that ProvSDN is incorporated into ONOS but that any control plane provenance tool is applicable because ProvIntent is modularized. ProvIntent can still function without a control plane provenance collector, in which case it would simply track the evolution of the intent as it goes through different states.

The ONOS Intent Framework includes an intent manager, intent compilers, intent installers, and the aforementioned intent state machine shown in Figure 1. ONOS is event-driven, which allows core controller components to dispatch relevant events (*e.g.,* IntentEvent events) for any apps or components listening for them. ONOS includes an Intent class that describes the intent, as well as an IntentData class that includes additional metadata, such as the intent state. As Intents can be compiled down to other (installable) Intents, we link to the highest-level intent that an app submitted.

For the intent state transitions, we added hooks in the intent phase classes. These hooks associate an intent's current state with an intent provenance graph's relevant nodes and edges. For the intent state changes that were made as a result of network state changes (*e.g.,* topology updates), we added hooks in the intent manager's event listener trackers. The hooks use the nSU relation to associate intent state changes with the relevant calls. For the network state changes brought about by intent state changes (*e.g.,* flow rule insertions), we added hooks in the intent installer classes. The hooks associate ProvSDN's provenance information about flow rule service API calls with ProvIntent's intent information by using the nSGB relation. For any other network information referenced by the intent in its compilation, we added hooks in the intent compiler classes and use the nSU relation.

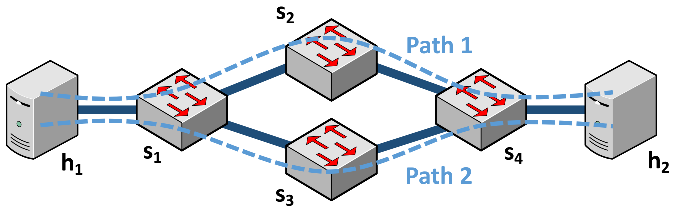

Fig. 4. Evaluation topology using 2 hosts and 4 switches.

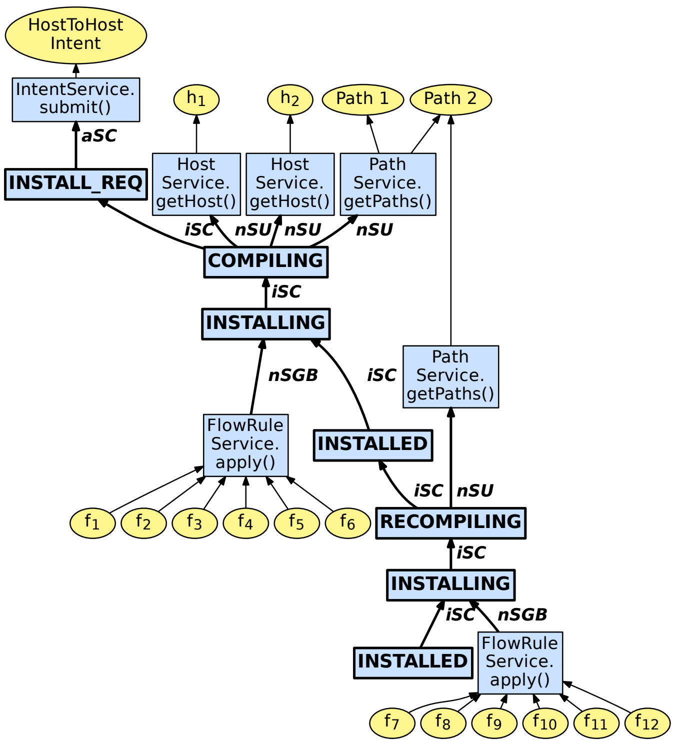

Fig. 5. Partial provenance graph showing ifwd's intent installation request, flow rule installation, topology change, and flow rule reinstallation. Bolded rectangles and edges represent the added provenance that conforms to ProvIntent's intent provenance model. Ellipses represent SDN control plane data structures that conform to ProvSDN's control plane provenance model. A practitioner would begin an analysis at an INSTALLING state and perform backward or forward tracing to understand causes or effects, respectively.

V. Evaluation

We evaluated the efficacy of ProvIntent by using ONOS's ifwd app [18], which implements reactive forwarding with intents and flow objectives. We used a 2-host, 4-switch Mininet [19] topology as shown in Figure 4. As our topology allows for multiple paths between hosts, we administratively shut down links to force a topology change. That allowed us to show how intents change states after they have been submitted.

Scenario: To begin, the ifwd app requests a HostToHostIntent between hosts h_1 and h_2. The HostToHostIntent compiler (*i.e.,* HostToHostIntentCompiler) gets information about the available hosts on the network (via the HostService) and the available shortest paths between the hosts (via the PathService). HostToHostIntents compile to LinkCollectionIntents and (installable) FlowRuleIntents. The installer selects path 1 and installs flow rules f_1 through f_6, which implement the bidirectional communication between hosts h_1 and h_2 across the 3 switches s_1, s_2, and s_4.

To force a reinstallation, we administratively bring down links in path 1 between switches s_1 and s_4. That leaves path 2 as the only viable path between hosts h_1 and h_2. The tracker service tracks the resulting network state changes and triggers a recompilation. As a result, the installer installs flow rules f_7 through f_{12} across the 3 switches s_1, s_3, and s_4.

Analysis: Figure 5 shows the salient features of the resulting provenance graph, which combines intent and control plane provenance from PROVINTENT and PROVSDN, respectively. Following the most recently installed intent state, a practitioner backtraces through the graph's iSC edges to understand the transitions that the intent took in the intent state machine. He or she determines that the intent was reinstalled once.

The practitioner notices that different sets of paths were returned during each compilation. He or she starts to investigate why different paths were chosen, a question that one can answer by backtracing through PROVSDN's control plane provenance. Had PROVINTENT's intent provenance not been included in the provenance graph, the practitioner's task of analyzing the reasons would have been more complicated.

Now suppose that the practitioner wants to validate the extent to which the intent's policy (*e.g.,* "allow communication between hosts h_1 and h_2") was followed in practice. The practitioner performs a backward trace starting at the most recently installed phase, and then performs a forward trace to find all control plane data structures (*i.e.,* entities) relevant to the intent. The result returns flow rules f_1 through f_{12}, and the practitioner can analyze these flows to validate the policy.

VI. Discussion and Future Work

IBN abstracts away unnecessary information when reasoning about policy. However, such abstraction creates visibility challenges when such information is needed to answer questions about what caused network failures, whether policies were implemented in practice, or whether policies could be improved [4]. Those challenges will only become more difficult with increased automation and the rise of machine-learning-based decisions on changing network intents and policies. PROVINTENT balances the need to provide the right amount of information and the ability to track additional details about the network's state as necessary.

The intent provenance model's concepts are applicable to other IBN frameworks and their intent state machines (*e.g.,* OpenDaylight's Network Intent Composition [20]). Intent state machine standardization is an open research problem.

Given that intents are extensible and can be tailored to a particular domain (*e.g.,* data protection [21]), we anticipate that the intent provenance model can fit into a domain-specific, provenance-aware system that can be queried at various abstraction levels. We imagine that a practitioner will be able to issue queries about past high-level activities (*e.g.,* business policies or workflows) that are mapped into lower-level activities (*e.g.,* intents and control plane configurations).

VII. Conclusion

We presented PROVINTENT, an IBN provenance framework that extends SDN control plane provenance to account for intent semantics. PROVINTENT provides capabilities to record and understand past intent and network activities. We defined an intent provenance model and designed a tool to capture and understand the resulting provenance. We used ONOS to implement PROVINTENT to show the practicality and utility of intent provenance for practitioner queries.

Acknowledgment

The authors would like to thank Jenny Applequist for her editorial assistance. This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-1657534 and CNS-1750024.

References

[1] K. Sivakumar and M. Chandramouli, "Concepts of Network Intent," Internet Research Task Force, Internet Draft, Oct. 2017. [Online]. Available: https://tools.ietf.org/id/draft-moulchan-nmrg-network-intent-concepts-00.html
[2] D. Sanvito, D. Moro, M. Gull, I. Filippini, A. Capone, and A. Campanella, "ONOS intent monitor and reroute service: Enabling plug & play routing logic," in *IEEE NetSoft '18.*
[3] A. Campanella, "Intent based network operations," in *IEEE OFC '19.*
[4] J. Smith. (2018, Nov.) Why SDN and IBN demand better network visibility. Network Computing (Informa PLC). [Online]. Available: https://www.networkcomputing.com/networking/why-sdn-and-ibn-demand-better-network-visibility
[5] B. E. Ujcich *et al.*, "Cross-app poisoning in software-defined networking," in *ACM CCS '18.*
[6] H. Wang, G. Yang, P. Chinprutthiwong, L. Xu, Y. Zhang, and G. Gu, "Towards fine-grained network security forensics and diagnosis in the SDN era," in *ACM CCS '18.*
[7] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *ACM HotSDN '14.*
[8] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *IEEE WoWMoM '14.*
[9] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling record and replay troubleshooting for networks," in *USENIX ATC '11.*
[10] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the software defined network control layer." in *NDSS '15.*
[11] C. Scott *et al.*, "Troubleshooting blackbox SDN control software with minimal causal sequences," in *SIGCOMM '14.*
[12] B. E. Ujcich, A. Miller, A. Bates, and W. H. Sanders, "Towards an accountable software-defined networking architecture," in *IEEE NetSoft '17.*
[13] ONF. (2019, Dec.) Github – opennetworkinglab/onos at 1.14.0. [Online]. Available: https://github.com/opennetworkinglab/onos/tree/onos-1.14
[14] Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "Answering why-not queries in software-defined networks with negative provenance," in *ACM HotNets '13.*
[15] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "Differential provenance: Better network diagnostics with reference events," in *ACM HotNets '15.*
[16] A. Dwaraki, S. Seetharaman, S. Natarajan, and T. Wolf, "GitFlow: Flow revision management for software-defined networks," in *ACM SOSR '15.*
[17] P. Missier, K. Belhajjame, and J. Cheney, "The W3C PROV family of specifications for modelling provenance metadata," in *ACM EDBT '13.*
[18] ONF. (2019, Dec.) Github – opennetworkinglab/onos-app-samples/ifwd. [Online]. Available: https://github.com/opennetworkinglab/onos-app-samples/tree/master/ifwd
[19] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *ACM HotNets '10.*
[20] OpenDaylight. (2019, Dec.) NIC:Main. [Online]. Available: https://wiki.opendaylight.org/view/Network_Intent_Composition
[21] B. E. Ujcich and W. H. Sanders, "Data protection intents for software-defined networking," in *IEEE NetSoft '19.*