Intelligence Enabled SDN Fault Localization via Programmable In-band Network Telemetry

Yongning Tang, Yangxuan Wu
School of Information Technology
Illinois State University
Normal IL, 61790
Email: ytang, ywu@ilstu.edu

Guang Cheng
School of Cyberspace Security
Southeast University
Nanjing Jiangsu, 21189
Email: gcheng@njnet.edu.cn

Zhiwei Xu
Dept. of Computer Science
University of Michigan-Dearborn
Dearborn MI, 48128
Email: zwxu@umich.edu

***Abstract*—Intelligent Fault localization for SDN becomes one of the most critical but difficult tasks. This paper proposes a new approach called *Policy-Aware In-band Network Telemetry* (PAINT) to tackle SDN fault localization. In the *PAINT* system, network operators define and deploy network services using a high-level *Service Provisioning Language* (SPL). Then, *PAINT* automatically parses the service policy to infer the causal relationship between service related network components and (end-to-end) service-level observable symptoms. Based on the causality model, *PAINT* deploys monitoring instruments for the symptoms. *PAINT* utilizes a dynamically created Symptom-Fault-Telemetry model to incorporate *In-band Network Telemetry* (INT) actions systematically into the fault reasoning process to improve the efficiency and accuracy of fault localization for SDN. *PAINT* has been extensively evaluated in a simulation environment for its accuracy and scalability with very positive results.**

***Index Terms*—fault Localization, Policy Aware, INT, SDN**

I. Introduction

Recent studies [3], [4] have forecast that global data center traffic will quadruple by 2021, and the majority of the Internet services will be provisioned from various Software Defined Networking (SDN) enabled datacenters. Despite their importance, datacenter networks still face the risk of performance, availability and reliability issues. For example, on Feb. 15, 2018, a database glitch affecting Google's application development platform. A widespread AWS network outage on Mar. 2, 2018 rendered choppy popular Internet services including Atlassian, Alack, and Twilio.

Many SDN fault reasoning and verification techniques assist operators focus on either analyzing the control plane configuration or checking the data plane network behavior. These solutions are limited in that they cannot correlate network symptoms between the control and the data planes, are expensive in terns of required computation and monitoring intrusiveness, and are harder to generalize across protocols since they have to model complex configuration languages and dynamic protocol behavior.

Recent advances on data plane programmability make *In-band Network Telemetry* (INT) a new method that can perform end-to-end monitoring directly in the data plane, enabling real-time and fine-grained network monitoring. However, how to adopt this new opportunity into a network management system is a non-trivial task, which may bring useful information to a network administrator or cause unnecessary intrusive overhead to a network.

This paper proposes a new approach called *Policy-Aware In-band Network Telemetry* (PAINT) to tackle SDN fault localization. In *PAINT*, network operators define and deploy network services using a high-level *Service Provisioning Language* (SPL). Then, *PAINT* automatically parses the service policy to infer the causal relationship between service related network components and (end-to-end) service-level symptoms. Based on the causality model, *PAINT* deploys monitoring instruments for the symptoms. If any symptoms received, *PAINT* conducts an initial fault reasoning to locate the root causes. However, if the initial result is too coarse-granular (i.e., too many suspected components) to accept, *PAINT* will adopt *INT* to collect relevant information directly from the internal network. *PAINT* utilizes a dynamically created Symptom-Fault-Telemetry model to incorporate *INT* actions systematically into the fault reasoning process to improve the efficiency and accuracy of fault localization for SDN.

In this paper, we make the following contributions:

- We propose a Symptom-Fault-Telemetry model to systematically incorporate an event-driven *INT* mechanism into the SDN fault reasoning process.
- We propose a mechanism to dynamically create service-oriented monitoring tasks via policy inference.
- The prototyped *PAINT* system shows the feasibility and the performance of the Policy-Aware INT mechanism.

This rest of the paper is organized as follows. Section II formalizes the problem in the context of SDN datacenter networks. Section III presents the individual modules of the *PAINT* system. We briefly discuss the implementation in Section IV, and extensively evaluate its performance in Section V. We will summarize other related work in Section VI and wrap it up in Section VII.

II. Motivation and Problem Formalization

Many recent studies show that modern datacenters are rife with hardware and software failures even though they are designed to be robust to large numbers of such faults. The large scale of deployment both ensures a non-trivial fault incidence rate and complicates the localization of these faults resulted from various root causes, including hardware failures, policy configuration and deployment errors.

SDN provides new abstractions and open interfaces to enable network programmability to multiple SDN layers. Network virtualization technologies shift traditional network functions to virtual ones called Virtual Network Functions (VNF), and embed them into commoditized hardware. However, in the SDN and NFV context, highly changing networks impose numerous challenges for fault localization, especially on how to detect continuous policy and provisioning changes and update dependencies among virtual and physical resources. Traditional network fault localization depending on pre-deployed monitoring instruments becomes infeasible for a modern SDN network. For instance, some newly created virtual hardware and/or software components may not even exist initially. Moreover, the SDN software stack itself is a complex distributed system, operating in asynchronous, heterogeneous, and failure-prone environments [3].

We believe service oriented monitoring can effectively address the challenges above. In the service oriented approach, monitoring tasks will be defined and deployed dynamically based on the current provisioned services and how these services are implemented. However, what to monitor and which observation granularity is sufficient remain as challenging questions.

Recently, a programmable data plane has been proposed [2], changing switches from fixed functions to programmable ones, and create many new applications including *In-band Network Telemetry* (INT) [1]. *INT* is a framework designed to allow the collection and reporting of network state, by the data plane, without requiring intervention or work by the control plane. In the *INT* architectural model, packets contain header fields that are interpreted as *telemetry instructions* by network devices. These instructions tell an INT-capable device what state to collect and write into the packet as it traverses the network. *INT* traffic sources (e.g., applications, end-host networking stacks, hypervisors, NICs, send-side ToRs, etc.) can embed the instructions either in normal data packets or in special probe packets. Similarly, *INT* traffic sinks retrieve (and optionally report) the collected results of these instructions, allowing the traffic sinks to monitor the exact data plane state that the packets observed while being forwarded. In the *PAINT* system, *INT* actions are used to monitor Hop latency, Queue occupancy. and link utilization for relevant QoS parameters, which can be easily extended with other functionalities.

III. Policy-aware Fault Reasoning

In this paper, we propose a novel approach called *PAINT* that can dynamically construct and update a Symptom-Fault-Telemetry model to incorporate *INT* actions into a Symptom-Fault model. As shown in Fig. 1, *PAINT* can:

1) understand a high-level policy and its defined network services along with any associated QoS requirements;
2) communicate with the controller to find how each service is deployed over an SDN network;

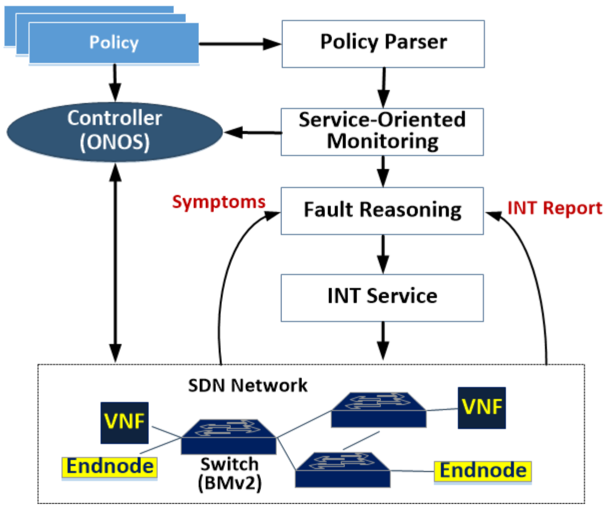


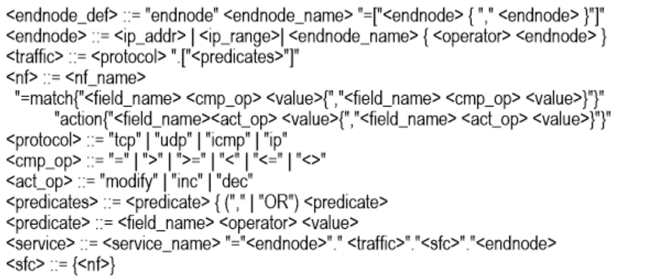
Fig. 1. *PAINT* System Overview

3) dynamically define and deploy service-oriented monitoring instruments to conduct (coarse-grained) end-to-end monitoring for service level symptoms;
4) collect and analyze symptoms to localize root causes;
5) when symptoms are insufficient to pinpoint exact faulty components, the relevant *INT* instructions will be automatically generated to inform the appropriate *INT* sources to conduct in-band network telemetry and collect reports until the satisfiable root causes found.

We use $F = \{f_1, f_2, \cdots, f_n\}$ to denote the fault set, and $S = \{s_1, s_2, \cdots, s_m\}$ to denote the symptom set that can be caused by one or multiple faults in F. Causality matrix $C_{F\times S} = c_{ij}(s_i|f_j)$ is used to define causal certainty between fault $f_i(f_i \in F)$ and symptom $s_i(s_i \in S)$. If $c_{ij} = 1$, it implies that the symptom s_i can be caused or explained by the fault f_j. S_{f_i} is a set of symptoms caused by the fault f_i. F_{s_i} is a set of all faults that might cause symptom s_i.

The task of the fault reasoning is to search for root faults in F based on the observed symptoms S_O. Accordingly, we have to address the following three problems: (1) Given the Fault-Symptom correlation matrix and S_O, construct a set of the most possible hypotheses, $\Phi = \{h_1, h_2, \cdots, h_p\}$, $h_i \subseteq F$, that can explain the current observed symptoms; (2) Given a set of possible hypotheses, find the most credible hypothesis h, that can give the best explanation for the current observed symptoms; (3) If multiple faults are still included in the hypothesis to explain a single symptom and cannot be further distinguished, and thus the selected hypothesis does not satisfy troubleshooting granularity requirement. In this case, *PAINT* will define a set of INT instructions and send to appropriate INT sources. With the received INT reports, *PAINT* can quickly exclude irrelevant faults from the previous hypotheses, and thus pinpoint to the exact ones.

In the following, we first introduce a Service Provisioning Policy Language, which is used in the *PAINT* system. *PAINT* is agnostic to policy abstraction model itself. Any other policy language such as APIC [4], GBP [5], PGA [6] can be easily



```
<endnode_def> ::= "endnode" <endnode_name> "=["<endnode> { "," <endnode> }"]"
<endnode> ::= <ip_addr> | <ip_range>| <endnode_name> { <operator> <endnode> }
<traffic> ::= <protocol> ".["<predicates>"]"
<nf> ::= <nf_name>
 "=match{"<field_name> <cmp_op> <value>{","<field_name> <cmp_op> <value>}"}"
      "action{"<field_name><act_op> <value>{","<field_name> <act_op> <value>}"}"
<protocol> ::= "tcp" | "udp" | "icmp" | "ip"
<cmp_op> ::= "=" | ">" | ">=" | "<" | "<=" | "<>"
<act_op> ::= "modify" | "inc" | "dec"
<predicates> ::= <predicate> { ("," | "OR") <predicate>
<predicate> ::= <field_name> <operator> <value>
<service> ::= <service_name> "="<endnode>"." <traffic>"."<sfc>"."<endnode>
<sfc> ::= {<nf>}
```

Fig. 2. The Syntax of Service Provisioning Policy Language

integrated into the *PAINT* system.

A. Service-oriented Monitoring via Policy Inference

1) Service-oriented QoS-aware policy language: In the following, we briefly discuss a policy language used in *PAINT* that can facilitate service provisioning and fault localization.

Fig. 2 shows the grammar of PAINT policy language in EBNF syntax. There are several important elements in the policy language: endnode, network function, traffic, and service. A service is composed of two logical endnodes and a traffic pattern between them with optionally a set of netowrk functions. Next, we briefly elaborate each language element with examples.

Endnode is a logical unit which contains the network addresses of entities (workstations, servers and network devices) that share the same pattern (e.g., in the same subnet or providing same type of server). The entities in one endhost can come from different physical subnets. An endhost can be defined by a set of IP ranges, or constructed by combining other endhost. The following example defines a endhost hadoopmachines:

$$hadoopmachines = [172.16.1.1172.16.1.100]$$
$$webserver = [10.0.0.110.0.0.2]$$

Traffic is defined as the combination of a protocol name and a set of properties associated with that protocol. Each property of a protocol is a predicate which is defined by the field name in that protocol header, the operator and value of that field as shown in Fig. 2.

The policy language defines high-level users intent. For example, $tcp.[port = 80]$ means http traffic. One service can represent multiple traffic flows in the network as long as those traffic flows can satisfy the conditions defined in the properties set. For example, $tcp.[port > 2045, port < 3078]$ represents all tcp traffics with destination port between 2045 and 3078.We can define the FB instant messaging (FB msg) and Bit Torrent (torrent) service as follow:

$$traffic : FB_{msg} = tcp.[port = 5050];$$
$$torrent = tcp.[port >= 6881, port <= 6999];$$
$$http = tcp.[port = 80];$$

NF (network function) is defined as the combination of a "match" operation and an "action" operation. For example, we can define a commercial L7-aware network function load balancer (LB) as match(dstip = Web.virtIP) modify(dstip=Web.RIPs), where Web.RIPs is a set of real IP addresses of destination web server EPs and Web.virtIP is the exposed virtual address of the web service. We can define the network function Load Balancer as follow:

$$nf \quad LB = match(dstip = 192.168.1.1)$$
$$action(dstip\ modify\ 10.0.0.1,\ 10.0.0.2)$$

Service is a network service provisioning between two end nodes with a specified traffic pattern and a service function chain consisting of a set of various network functions. For example, a web service between "hadoopmachines" and "webserver" through a "LB" via TCP port 80 is simply presented as: $webservice = hadoopmachines.http.LB.webserver$

A policy rule from a stakeholder is essentially a service specifications, and structured and expressed as a conceptual network diagram. In the following, we focus on showing a feasible approach to automating the process of fault localization in a highly flexible, dynamic and large scale SDN network.

B. Distributed Service-oriented Monitoring

Network monitoring is a crucial but challenging task. There always exist several tough trade-offs to make when designing such a system, including:

- Monitoring Coverage versus Resource Overhead: In order to monitor more possible failures or faulty components, the system demands more computing resources from both the controller and network forwarding devices for various monitoring tasks.
- Timely-response verse Network Intrusiveness: In order to timely discover or even predict potential problems on the network, consistent sampling and probing are necessarily used to monitor the network status.
- Finer-granularity versus System Workload: In order to accurately pinpoint the root causes, more specific objects need be monitored and thus may result to high workload on the system.

Many approaches have been studies and devised to tackle the challenges above. In the *PAINT* system, we adopt a policy-aware incremental even-driven distributed monitoring approach. The monitoring system first parses the policy to create service-oriented monitoring schemes, in which for each service, one or multiple end-to-end paths along with their QoS parameters if required are selected for monitoring.

Thanks to the scheme in the SPL policy language, the Policy Parser as shown in Fig. 1 can analyze all defined services to dynamically create a list of monitored objects (MO) for each service, which is essentially corresponding to the set of observable symptoms for that service. This is because if any abnormal status observed for a monitored object, a related symptom will be reported. For each symptom, all components along an end-to-end network path for a service, including hardware and software, physical and virtual, can be used to explain the occurrence of that symptom, which is essential the causality model describing the relationship between faulty components and their related symptoms. Such a service-oriented causality model will be used to define a current network monitoring scheme, including what to monitor and whom to report. This monitoring scheme will be submitted to the controller for deployment.

C. Event-driven Policy-aware In-band Network Telemetry

Recently, a programmable data plane using a high-level language for programming protocol-independent packet processors (P4) [2] has been proposed, With P4, network operators can decide how the data forwarding plane processes packets without worrying about implementation details. Many new network applications, including intelligent traffic engineering, smart congestion control, and In-band Network Telemetry, have been proposed upon dataplane programmability.

In-band Network Telemetry (INT) [1] is a framework designed to allow the collection and reporting of network state, by the data plane, without requiring intervention or work by the control plane. In the INT architectural model, packets contain header fields that are interpreted as *Telemetry Instructions* by network devices. These instructions informs an INT-capable device what state to collect and write into the packet as it traverses the network.

INT traffic can be initiated by various sources, such as applications, end-host networking stacks, hypervisors, NICs, send-side ToRs, etc. INT traffic sources can embed the instructions either in normal data packets or in special probe packets. Similarly, INT traffic sinks retrieve (and optionally report) the collected results of these instructions, allowing the traffic sinks to monitor the exact data plane state that the packets have observed while being forwarded. For the need if monitoring delay, loss rate, bandwidth utilization in this work, we will focus on Hop Latency, (in/out) port utilization and packet count, Queue occupancy. The user can define as other information to collect.

If the observed symptoms that are defined by the Service-Oriented Monitoring module as shown in Fig. 1 can provide sufficient information to pinpoint specific faulty components, the fault reasoning process can stop. Otherwise, the INT service module, as shown in Fig. 1, will create a list of INT actions typically optimized for minimum overhead for the network and provide to various appropriate INT sources.

Algorithm 1 returns either a hypothesis h that contains faulty components jointly explaining (S_O) as shown in lines $1-10$. If the size of the hypothesis is too large, we use minimal number of observed symptoms (i.e., service paths) to cover all faults in h as shown in lines $12-15$, and then create INT instructions and deploy them on selected INT sources.

Algorithm 1 Policy-Aware Intelligent Fault Reasoning

```
Input: $S_O$
Output: fault hypothesis $h$ or $S_{int}$
 1: for all $s \in S_O$ do
 2:     $F \leftarrow F \cup F_s$
 3: end for
 4: while $S_O \neq \emptyset$ do
 5:     for ¡$f \in F$¿ do
 6:         $find\ f\ s.t.\ |S_f|\ is\ maximum$
 7:     end for
 8:     $S_O = S_O - S_f$
 9:     $h \leftarrow h \cup \{f\}$
10: end while
11: if $|h| > threshold$ then
12:     for all $s \in S_O$ do
13:         find $s$ s.t. $|F_s \cap h|$ is the largest
14:         $S_{int} \leftarrow S_{int} \cup \{s\}$
15:     end for
16:     return $S_{int}$
17: else
18:     return $< h >$
19: end if
```

IV. PAINT System

A. System Design

We present *PAINT* system that can dynamically generate an end-to-end monitoring scheme for current provisioned services, decide an optimal set of INT actions/instructions based on the analysis observed symptoms in order to localize the root causes of the problems. The system mainly consists of (i) Policy Parser, (ii) Service oriented Monitoring, (iii) Fault Reasoning, and (iv) INT Service.

PAINT is implemented in Python 2.7 and Erlang 16.2.1 [10]. As an extension to Pyretic [9], we use open source Python packages for mapping provisioned services to the corresponding monitoring schemes. Erlang is used as a backend engine for the Symptom-Fault-Telemetry model construction and the distributed monitoring.

Distributed Service-oriented Monitoring can be essentially optimized as a resource optimization problem. From the result of the policy parsing, PAINT can determine which service paths need be monitored. Since the monitoring instruments need be deployed among endnodes with different available resources, This classical problem has been studied extensively in previous work [11] [12] [13]. PAINT adopts a greedy search to identify and rank available endnodes for the given monitoring workload, and then apply max-min fairness approximation algorithm to fill up the available resources among those selected endnodes.

B. ONOS and INT

ONOS, unlike many central SDN controllers, uses a distributed approach: we have several ONOS controller instances instead of only a central one. Thus, ONOS has high scalability and availability. ONOS also allows creating new services easily through abstraction. From version 1.6 (Goldeneye), ONOS supports the P4 language with a BMv2 [16] switch, a P4 software switch. The BMv2 software switch is just a software simulator for P4 hardware, which requires a P4 configuration file to be fully functional. In this case, the configuration file is the INT switch in json format, *sw.json*, the result after compiling the INT P4 source code. At the beginning, the INT Service module reads *sw.json* and pushes the configuration data to all BMv2 switches. INT packets sent to ONOS are processed at the low-level layer of the ONOS core, Int data processor in BMv2 provider.

The format of INT is detailed in the INT specification. In general, a packet with INT information is the original network packet with two more fields: the INT header, which contains

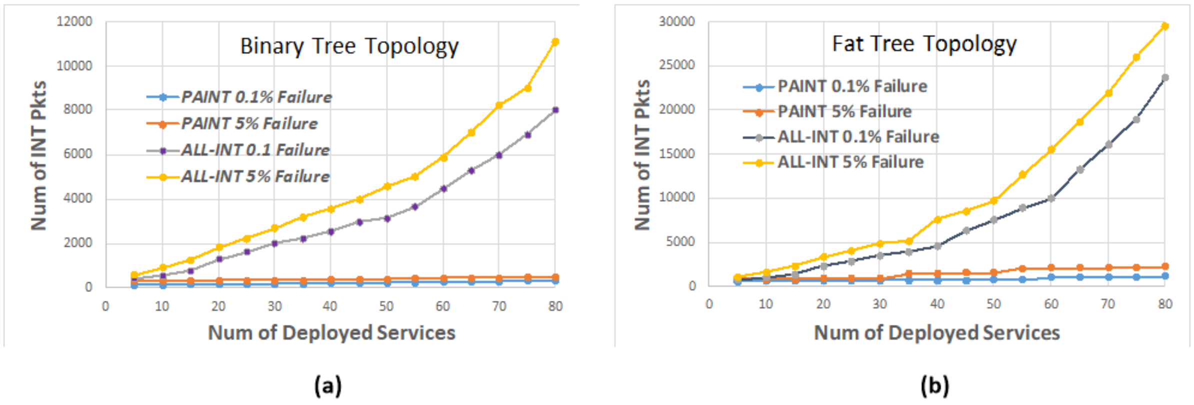


Fig. 3. PAINT Scalability

all fixed INT parameters and information, and the INT data, which contain per-switch information that is added every time an INT packet passes through a switch. These fields must be attached as an option or payload in an encapsulated protocol (e.g., VXLAN), or a shim header in IP, TCP, or UDP packets. In PAINT, we adopt the scheme to integrate INT information into a shim header of UDP packets [8].

V. Evaluation

We evaluate the performance and overhead of the *PAINT* system by comparing its event-driven INT approach with a similar Risk Sharing based approach in both Coarse-grain and Finer-grain modes. We simulate in Mininet for testing with the following two different network topologies:

- A 4-level binary tree topology with 15 switches and 16 servers, which can represent a typical small-to-medium size enterprise network.
- A Google fat-tree topology with $k = 8$ pods, which consists of 16 core, 32 aggregation, and 32 edge switches, connecting to 128 servers and representing a typical medium-to-large scale service provider network,

All the switches are BMv2 switch and connected with ONOS controller through Thrift protocol.

In our evaluation, we focus on two important aspects of *PAINT*: its scalability and its accuracy.

A. Scalability

One of the contribution in *PAINT* is to elegantly balance the efficiency of a fault localization system and its network intrusiveness, In our experiments, we compare the *PAINT* system with a native INT system that always embed INT packets along with service packets.

We have created different scenarios with 5 to 80 provisioned network services. For all identifiable relevant components, including hardware and software, physical and virtual one, we inject 0.1% and 5% faults into the system.

In terms of number of INT packets as a scalability measure, the *PAINT* performs very well as shown in Fig. 3. With the increasing number of services and two different fault scenarios, *PAINT* presents a slow linear increase on the required INT packets. When the number of services increased 16 times, the number of increased INT packets only increased 3 times when the fault rate is 0.1%; increased only 1.5 times when the fault rate is 5%. On the other hand, in the native INT fault localization system, marked as All-INT in Fig. 3. the number of increased INT packets increased almost 20 times for both fault rate scenarios. Even worse, the increasing trend for the All-INT approach implies a possible exponential increase when more services deployed. The similar property has been observed in both medium size binary tree topology and large size fat tree topology.

B. Accuracy

Another important performance metric for any fault localization system is its accuracy. In order to evaluate the accuracy of *PAINT*, we inject different number of faults to the system, varying from 20 t0 300. All the simulated scenarios use Google Fat Tree topology. We use Fault Detection Rate (FDR) and False Positive Ratio (FPR) to evaluate the accuracy. FDR is the ratio of the total detected faults to the total faults. FPR is the ratio of the number of false reported faults to the total number of detected faults.

We compare *PAINT* to commonly used Symptom-Fault bipartite model based approach [14], also referred as the Risk Share (RS) model based approach [15]. For the RS approach, we implement two different versions: coarse-granular RS (i.e., RS-Coarse) and fine-granular RS (i.e., RS-Fine). In RS-Coarse, each logical component may contain multiple independently identifiable components. In RS-Fine, each component is exactly an independently identifiable components.

As shown in Fig. 4, thanks to the event-driven nature of the *PAINT* system, the fault detection rate is very high with the average value 95% for different fault scenarios. We also observed that RS-Coarse can achieve similar performance in FDR as long as the average fault cardinality (e.g., the number of identifiable components in a logical component) close to 10 is acceptable. If keeping the fault cardinality as 1, the same as in the *PAINT* system, RS-Fine performed very poorly with only less than 60% FDR.

One main reason causing higher false positive rate in a fault

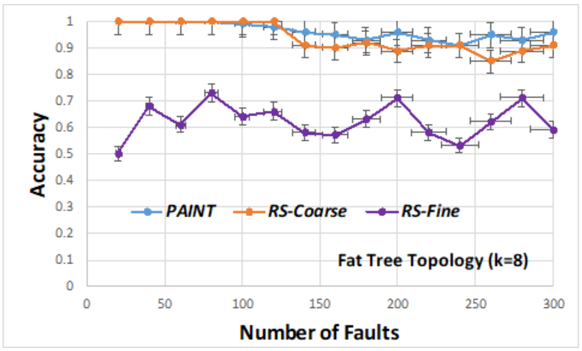


Fig. 4. System Accuracy: Fault Detection Rate

localization system is due to insufficient observation on the network real status. Again, the capability of increasing network visibility based on the need empowers *PAINT* with a very low (3%) FPR. With the higher tolerance on the coarse-grained granularity in RS-Coarse, the FPR remains as low as in the *PAINT* system until the number of faults getting larger (e.g., more than 150). In RS-Fine, the FPR is evidently as worse as $30 - 40\%$.

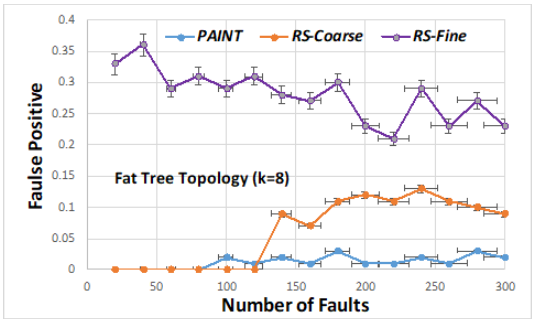


Fig. 5. System Accuracy: False Positive Ratio

VI. Related Work

Many solutions [17] [18] [19] were proposed to address fault localization problem in SDNs. NetSight [19] offers a platform that records packet histories and enables applications to retrieve packet histories of interest. The authors implemented several applications on top of NetSight that automatically retrieve the packet histories leading to specified events, e.g., reachability errors or loops. DPRL [20] implements a requirement checker that automatically generates and injects test packets in order to verify the specified path behavior. Header space analysis [17] is a general and protocol agnostic framework. It allows to statically check network specifications and configurations to identify network requirements such as reachability failures or forwarding loops. Anteater [21] collects the network topology and the forwarding information from the network devices. The operator may then specify an invariant which is verified by Anteater by using a boolean satisfiability problem (SAT) solver.

In contrast to other previous approaches, *PAINT* is a service-oriented and policy-aware, and event-driven fault localization system. PAINT is adaptive to different scenarios with minimal monitoring cost.

VII. Conclusion

PAINT shows a feasible approach to model symptom-fault causality along with INT actions for highly dynamic and large-scale SDNs. The concept of service-oriented fault reasoning provides a new fault diagnosis paradigm that can be used by individual network users or collaboratively used among multiple uses. In our future work, we will explore the possibility of creating a common interface for users from different physical and virtual layers to share their models and observations to jointly troubleshoot SDN faults.

References

[1] The P4.org Applications Working Group, In-band Network Telemetry (INT) Dataplane Specification, Working draft. 2018-08-17.
[2] P. Bosshart, D Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, Amin, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. SIGCOMM Comput. Commun. Rev.July 2014.
[3] R. C. Scott, A. Wundsam, K. Zarifis, and S. Shenker. What, Where, and When: Software Fault Localization for SDN. EECS Department, Univ. California, Berkeley, Tech. Rep. UCB/EECS-2012-178, July 2012.
[4] Cisco Application Centric Infrastructure, https://goo.gl/WQYqnv, 2017
[5] ODL Group Policy, https://wiki.opendaylight.org/view/Group_Policy:Main.
[6] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, PGA: Using Graphs to Express and Automatically Reconcile Network Policies, in SIGCOMM 2015.
[7] Yongning Tang, Guang Cheng, Zhiwei Xu, Feng Chen, Khalid Elmansor, and Yangxuan Wu, Automatic Belief Network Modelling via Policy Inference for SDN Fault Localization. (2016) Journal of Internet Services and Applications. Vol. 7:1.
[8] Jonghwan Hyun; Nguyen Van Tu; James Won-Ki Hong. Towards ONOS-based SDN Monitoring using In-band Network Telemetry. 2018 IEEE/IFIP Network Operations and Management Symposium. Apr. 2018.
[9] Monsanto C, Reich J, Foster N, Rexford J, Walker D (2013) Composing Software Defined Networks. In Proceedings of NSDI 2013
[10] J. Armstrong, R. Virding, and M. Williams. Concurrent Programming in Erlang. Prentice Hall International (UK), 2nd edition, 1996.
[11] Ignacio Bermudez; Stefano Traverso; Maurizio Munaf; and Marco Mellia. A Distributed Architecture for the Monitoring of Clouds and CDNs: Applications to Amazon AWS. IEEE Transactions on Network and Service Management. 2014. Volume: 11 , Issue: 4.
[12] Adam Pavlidis; Giannis Sotiropoulos; Kostas Giotis; Dimitris Kalogeras; and Vasilis Maglaris. NFV-compliant Traffic Monitoring and Anomaly Detection based on Dispersed Vantage Points in Shared Network Infrastructures. 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)
[13] Hongjun Liu; Xiaofeng Hu; Dan Zhao; and Xicheng Lu. Characterizing Enough Vantage Points for Pinpointing Routing Instability. 2012 IEEE 26th International Parallel and Distributed Processing Symposium.
[14] Yongning Tang, Ehab S. Al-Shaer, and Raouf Boutaba, Active Integrated Fault Localization in Communication Networks. IEEE IM 2005
[15] P. Tammana, P. Chandra, R. Kompella, and M. Lee. Fault Localization in Large-Scale Network Policy Deployment.IEEE ICDCS 2018
[16] BMv2, https://github.com/p4lang/behavioral-model.
[17] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In Proceedings of NSDI 2012
[18] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real Time Network Policy Checking Using Header Space Analysis. In Proceedings of NSDI 2013
[19] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres, and N. McKeown. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In Proceedings of NSDI 2014.
[20] Lebrun D, Vissicchio S, Bonaventure O. Towards Test-Driven Software Defined Networking. In IEEE NOMS 2014
[21] Mai H, Khurshid A, Agarwal R, Caesar M, Godfrey PB, King ST. Debugging the Data Plane with Anteater. SIGCOMM 2011