A Comprehensive Study of Bugs in Software Defined Networks

Ayush Bhardwaj
Brown University
ayush_bhardwaj@brown.edu

Zhenyu Zhou
Duke University
zzy@cs.duke.edu

Theophilus A. Benson
Brown University
tab@cs.brown.edu

***Abstract*—Software-defined networking (SDN) enables innovative and impressive solutions in the networking domain by decoupling the control plane from the data plane. In an SDN environment, the network control logic for load balancing, routing, and access control is written in software running on a decoupled control plane. As with any software development cycle, the SDN control plane is prone to bugs that impact the network's performance and availability. Yet, as a community, we lack holistic, in-depth studies of bugs within the SDN ecosystem. A bug taxonomy is one of the most promising ways to lay the foundations required for (1) evaluating and directing emerging research directions on fault detection and recovery, and (2) informing operational practices of network administrators. This paper takes the first step towards laying this foundation by providing a comprehensive study and analysis of over 500 'critical' bugs (including ∼150 with manual analysis) in three of the most widely-used SDN controllers, i.e., FAUCET, ONOS, and CORD. We create a taxonomy of these SDN bugs, analyze their operational impact, and implications for the developers. We use our taxonomy to analyze the effectiveness and coverage of several prominent SDN fault tolerance and diagnosis techniques. This study is the first of its kind in scale and coverage to the best of our knowledge.**

***Index Terms*—SDN, Bugs, Fault-Tolerance, Taxonomy**

I. INTRODUCTION

Software-defined Networking (SDN) has enabled a paradigm shift from legacy networking to programmable networks which has transformed ISP networks [1]–[3], Clouds [4]–[6] and content provider networks [7]–[9]. Adoption of SDN by all major companies has enabled them to: (1) simplify provisioning and management of their networks, (2) better utilize network resources available for disposal, and (3) lower CAPEX (Capital Expenditures) and OPEX (Operating Expenditures).

SDN's key principle is to decouple functionality for routing, security, and performance from the networking hardware, i.e., router and switches. This functionality is rewritten in specialized software and deployed at a centralized location, called the controller. Today, modern SDN controllers are complex pieces of software comprising millions of lines of code. With key networking functionality softwarized and deployed on controllers, it is no surprise that any bugs within the SDN controller can lead to network performance and availability issues.

In fact, recent studies by Google [7] and Facebook [10] have shown that 30% of the outages in their SDN deployments are due to software bugs in SDN control planes. Despite the mounting evidence from industry and analysis of opensource bugs [7], [10]–[12], the community is lacking a systematic and detailed analysis of critical bugs within the SDN ecosystem.

This paper provides an in-depth analysis of over 500 critical bugs across three popular and prominent controllers within the SDN ecosystem. We created a taxonomy of bugs through our analysis, evaluated existing SDN fault-tolerant frameworks, and identified classes of bugs that require more research. Our taxonomy provides the building blocks for designing representative and informed fault-injectors for testing SDN controllers.

Our study is motivated by the following key research questions

- RQ1: What are the characteristics of bugs in SDNs?
- RQ2: What is the operational impact of these bugs?
- RQ3: How are these bugs triggered, and what strategies are used to fix them?
- RQ4: How can network operators benefit from this study?
- RQ5: How effective are emerging research prototypes?

In answering these questions, this work lays the foundation for richer and more advanced bug-tolerant SDN systems.

Our key findings are:

- Contrary to the growing work [13], [14] that effectively tackle non-deterministic bugs, our study shows that there is evidence, to the contrary, that most of the critical bugs are deterministic in nature.
- While there is a growing number of SDNs fault tolerance frameworks, e.g., Ravana [13] or STS [12], these are focused on tackling bugs triggered by network-events. Unfortunately, they fall short in tackling bugs triggered by other types of events, e.g., configuration or OS events, e.g., timers. In Section VII-C, we show that while most existing approaches can detect bugs, recovering from these bugs remains an unsolved question and new tools are necessary to fill this gap.
- SDN controllers are prone to bugs like any large software system. However, the specific subset of bugs and their distributions within SDNs are different from traditional server applications and distributed software. For example, in server applications, most bugs are due to configuration [15], [16], whereas, in SDNs, we found external calls and network events form a major portion of the bugs,

978-1-6654-3572-7/21/$31.00 ©2021 IEEE
DOI 10.1109/DSN48987.2021.00026

which requires a redesign of monitoring techniques to monitor all external interactions in addition to network events.
- One of the critical advantages of SDN over legacy networks is the global visibility [17] and the broader optimizations that it enables. However, we observe that the result of many of these bugs (e.g., bugs triggered by network events (19.8%)) is that this visibility is significantly lowered. In essence, these bugs eliminate a crucial benefit of SDNs.

Our analysis of the SDN bug corpus is largely driven by manual analysis and categorization of the different bugs across controller platforms. To ensure that our results generalize, we employ NLP-based analysis across a larger set of bugs.

Given the questions above, we re-used well-established taxonomies [18], [19] (Table I) and extended them to incorporate networking specific issues. The contributions of our characterization study can be summarized as follows:

- We provide a holistic view of SDN bugs to allow developers and researchers to leverage our conclusions to improve the SDN fault tolerance landscape. (§ IV)
- We extract guidelines and operational hints for managing and operating SDN networks (e.g., guidelines for Controller selection). (§ VII-A)
- We evaluate and analyze the coverage and efficacy of several existing SDN fault tolerant and recovery techniques. (§ VII-C)
- We identified the feasibility and effectiveness of designing NLP-based techniques for root cause diagnosis. (§ VII-B)

RoadMap. The rest of this paper is structured as follows: In section II, we discuss our target systems, our methodology, and our approach for automated analysis. In section III, we analyze bugs by their type. In section IV, we explore the operational impact of these bugs. In Section V, we analyze the events that trigger them. In section VI, we analyze code repositories to understand their software engineering practices. In section VII, we discuss the implications of these bugs. In section VIII, we discuss the limitations of and threats to our study. We conclude in sections IX and X, by describing the related and summarize conclusions.

II. Methodology

In this section, we discuss the controller frameworks that we analyze (§ II-A) and present our analysis techniques (§ II-B).

A. Target Systems

In Figure 1, we present an overview of the SDN ecosystem. The ecosystem comprises of three components: (i) SDN Applications, which provide specific network functionality, e.g., routing [20], load balancing [21] or access control [22]. (ii) SDN controller framework, which manages interactions between the SDN Applications and the underlying network devices (e.g., switches). (iii) the network data plane, which

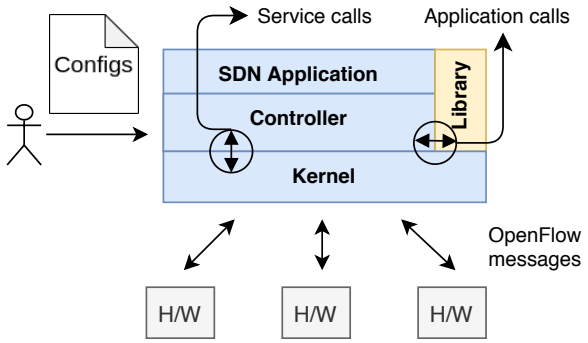

Figure 1: Generic Controller Stack.

consists of the switches and routers running within the network. The interactions between the SDN control plane (applications and controller) and the data plane occur through the exchange of SDN control messages (i.e., OpenFlow messages [23] or XMPP messages [24]). Many SDN controller frameworks build on third-party libraries to provide additional functionality, e.g., state management or packet processing. Thus controllers come bundled with a plethora of additional third-party libraries and services (indicated as a yellow box in Figure 1). SDN controllers are fundamentally event-driven: the arrows in Figure 1 demonstrate the various sources of input events that a controller reacts to (configuration, network events, the kernel through system calls, and application libraries through function calls).

Although there are approximately 32 controllers, we focus our study on three of the four most mature and popular open-source controllers are: ODL, CORD, ONOS, and FAUCET. We selected ONOS and CORD over ODL because they are used by major operators, e.g., Comcast [25], Google [26], etc., in a large scale real-world production environments. Moreover, unlike ONOS or ODL, CORD is specially tailored for emerging technologies (e.g., 5G-MEC [27], [28]) – thus providing a different perspective. We selected FAUCET because it is used at Google [29] and provides a unique perspective from the other controllers because it has a more compact structure and is written in Python. Next, we elaborate on the design of each of the three SDN controller frameworks:

- **FAUCET** [30] boasts a monolithic and compact codebase that migrates existing network functionalities like routing protocols, neighbor discovery, etc., into vendor-independent data planes. FAUCET manages flow decisions by utilizing multiple Access Control Lists(ACL) and multi-table processing [31].
- **ONOS (Open Network Operating System)** [32] builds on four major goals: modularity, configuration flexibility, isolation of subsystems, and protocol agnosticism. ONOS utilizes an intent-based API that captures policy directives for controlling network function. These intent-based APIs are realized through a set of state transition machines. Each subsystem employs a different state machine. This is distinct from FAUCET's monolithic but compact design.
- **Open CORD (Central Office Re-architected as a Datacenter)** [33] is a specialized version of ONOS developed

for Telecom Central Office (CO) to replace purpose-built hardware with cost-effective, agile networks. CORD is composed of four open-source projects, including Openstack, ONOS, Docker, and XOS. CORD provides a unique subsystem, based on XOS [34], to orchestrate coordination across these four code-bases.

B. Data Set and Methodology

The controller frameworks maintain a structured bug tracking and code management system — ONOS and CORD use JIRA for bugs and Gerrit for rolling out fixes, whereas FAUCET uses Github for bug tracking and managing fixes. While Jira includes tags that allow us to analyze bugs based on developer-identified severity levels, for Github, we used a keyword approach [35] to extract severity levels.

Data. As of April 2020, the FAUCET, ONOS, and CORD communities have identified 251, 186, and 358 critical bugs, respectively, which include both open and close bugs. In examining the bugs in ONOS and CORD, we found that: (1) Over time, the number of critical bugs keeps increasing. This motivates a need for more principled analysis. (2) We observe that a burst of bugs occurs around release dates. For example, in the first quarter of 2017, we observed a burst in CORD bugs which coincided with a release [36]. This highlights the need for longitudinal analysis across different releases.

For our study, we randomly selected **50 closed**[1] bugs from each controller for manual analysis. Moreover, we further verified the automatic analysis with an extended data set containing over 500 critical bugs.

C. Bug Autoclassification with NLP

To scale and automate classification, we re-use an NLP technique that prior bug studies have used, i.e., Word2Vec [37], to classify bugs and validate our taxonomy. We summarize the steps as follows:

- First, we pre-process the bug data to extract features. There are three classic approaches for keyword extracting including Latent Dirichlet Allocation (LDA) [38], Hierarchical Dirichlet Process (HDP) [39] and Non-negative Matrix Factorization (NMF) [40] based on Term Frequency Inverse Document Frequency (TF-IDF) [41]. We choose the last approach because previous work [42], [43] has demonstrated its potential to analyze similar data.
- Second, we train a Word2Vec model, which provides a mechanism for automatically determining similar words.

Given a bug description, these two steps allow us to map each bug to a numerical vector in a Euclidean space. After mapping bugs to Euclidean space, we can employ classic Machine Learning (ML) techniques, e.g., Support Vector Machine (SVM) or Decision Tree (DT), to automatically classify the bugs.

[1]49,49,48 from CORD, ONOS, FAUCET — we initially had 50 but removed open bugs to enable classification by fixes.

1) Bug Labeling: We utilize the following dimensions to classify the bugs: bug type, outcome, fix, and trigger. In Table I, we summarize these dimensions. These dimensions align with the recent work to characterize bugs in cloud systems [18], [19] which provide a similar classification as Orthogonal Defect Classification (ODC) [44]. At a high level, we classify bugs based on determinism to understand their reproducibility. For the root-cause and fixes, we classify bugs based on the controller code-base or logic's impact: some problems require changes to logic while others do not– similarly, some bugs are due to existing logic or absence of any logic (e.g., edge cases). To verify the fixes, we manually analyzed the source code patches and fixes. For the triggers, we identify four key events that initiate bugs. These events align with a canonical SDN controller (Figure 1). For the symptoms, we focus on the type of failure triggered by the bug.

Each bug receives at most one tag from each of the dimensions in Table I.

2) Validation: We validated the automated classification techniques with cross-validation by splitting our data set into 2/3 for training and 1/3 for testing. We explored several classic ML techniques, including Support Vector Machine (SVM) and Decision Tree (DT), Principal Component Analysis (PCA), and AdaBoost. In our experiments, we found that SVM model with normalization provided the best accuracy for predicting bug types and symptoms, with accuracies of 96% and 86%, respectively. Unfortunately, we found it hard to find any algorithm to predict bug fixes accurately, and we believe this is because bug descriptions generally provide little data about the fixes.

III. RQ1: Bug Type

We begin by classifying bugs according to determinism. Deterministic bugs are defined as bugs that are clearly reproducible with a fixed set of input actions, whereas non-deterministic bugs are inconsistent and cannot be consistently reproduced by replaying the same set of input events/actions. The key observation is that all frameworks are dominated by deterministic bugs: FAUCET (96%), ONOS (94%), and CORD (94%). One potential reason for this is that many controllers employ standard state-machine-based techniques [13], [14], [45], [46], e.g., Paxos [45] or Raft [46], which tackle and mask most non-deterministic bugs.

Takeaway. Given the dominance of deterministics bugs, we believe that record-and-replay-based recovery techniques [47] will have limited applicability on most SDN controllers. Instead, we recommend failure recovery systems which alter controller input events [12], [48], environments [49], [50], or source code [51]–[55].

IV. RQ2: Operational Impact of SDN Bugs

In this section, we explore the bugs' symptoms and characterize them based on the controller's behavior. The analysis of symptoms and controller behavior provides us with a first step towards understanding each bug's operational impact.

Classification	Categories
Bug Type	Deterministic, Non-deterministic
Root Cause	**Controller Logic-bugs:** Load, Concurrency, Memory, Missing Logic
	Non Controller logic-bugs: Human (misconfiguration), Ecosystem Interaction (Third-Party, Application Libraries or System Calls)
Symptoms	Performance, Fail-stop, Error Message, Byzantine (Wrong Behavior)
Fix	**No Logic Changes:** Rollback Upgrades, Upgrade Packages
	Add New Logic: Add Logic
	Change Existing Logic: Add Synchronization, Fix Configuration, Add Compatibility, Workaround [35]
Trigger	Configuration, External Calls, Network Events (OpenFlow Message), Hardware Reboots

Table I: Bug Taxonomy.

Byzantine Failures (61.33%): A majority of the bugs lead to the following unexpected behavior: (i) gray failures – a partial outage of the controller (52.17%), where some controller functionality is working while others are not. For example, in FAUCET-1623 [56] where the controller continues to manage flows but is unable to manage broadcast packets because of an unhandled edge case, a bug in the mirroring interface (shown in Figure 3). (ii) stalling (20.65%), where the controller temporarily freezes, and (iii) incorrect behavior (27.18%). Unlike stalling or partial outages, incorrect behavior is difficult to detect and diagnose because they do not generate error messages or trigger any normal alerts.

Takeaway. These bugs, in general, highlight the need of formal network verification; however, early works on verification [57]–[59] focus on the datapath or provide limited validation of runtime behavior. Our analysis indicates a need for more runtime verification of controller behavior.

Fail-stop (20%): Bugs that cause fail-stop failures or controller crashes are the most dire bugs as they directly impact the network's availability and lead to production downtime. In Figure 2, we analyze the root cause of these bugs. In FAUCET, these bugs are caused by human mistakes or ecosystem interactions. This implies that crashes are due to the edge cases related to certain external scenarios. In contrast with FAUCET, in ONOS and CORD, a majority of the bugs are due to incorrect controller-logic, e.g., load, memory, and missing code logic. For example, a misconfiguration led to a null pointer exception in CORD's host and multicast handlers (CORD-2470 [60]),

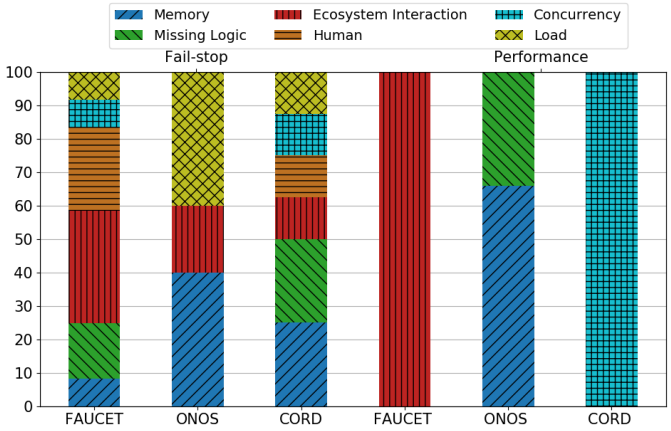

Figure 2: Distribution of Root Causes of the (a) Fail-Stop (first three bars) and (ii) Performance Bugs Across Controllers (last three bars).

```
+ def _build_mask_flood_rules(self, vlan,
    ↪ eth_dst, eth_dst_mask,
    ↪ exclude_unicast, command,
    ↪ flood_priority, mirror_acts):

- def _build_mirrored_flood_rules(self,
    ↪ vlan, eth_dst, eth_dst_mask,
    ↪ exclude_unicast, command,
    ↪ flood_priority):
```

Figure 3: Patch for FAUCET- 1623 [56], where interface mirroring didn't mirror output broadcast packets which was fixed by adding a case for mirrored ports.

which crashed the CORD controller. Despite CORD being based on ONOS, we observe a key difference between ONOS and CORD: in general, CORD has significantly more bugs due to "missing code logic," demonstrating a level of immaturity in the codebase.

Takeaway: Fail-stop bugs are the easiest to detect but have disasterous consequences. Our initial analysis shows that exploring designs to improve memory safety (e.g., memory safe languages like RUST [61] and programming styles [62]) will significantly improve availability.

Error Message (14.7%): In general, we ignore these bugs because they result in warnings that have no direct operational impact. The main observation is that CORD has the best exception handling, which leads to fewer error messages.

Performance (4%): From Figure 2, we observe that most of the bugs that result in slow controller performance can be triaged to one or two root causes. From the Figure, we also observe that different controllers have different root causes. A key surprise is that increased system load is not the main cause of slow performance. Instead, increased-system load leads to other failures, i.e., fail-stop and byzantine failures. We observe that poor performance is due to FAUCET's interactions with the ecosystem, concurrency bugs in ONOS, and memory errors in CORD. Thus broadly speaking, these bugs in FAUCET are due to factors generally beyond the developer's interactions, whereas in ONOS and CORD, they are due to poor programming logic.

Takeaway. Performance bugs [63] can cascade into a variety of dire bugs, e.g., byzantine, crash, etc., that can introduce SDN control plane instability. These bugs require active monitoring and health check system; however, such systems introduce significant overheads. For some of these

ONOS Version	# VD	High
ONOS-1.12.0	3	1
ONOS-1.13.0	28	17
ONOS-1.14.0	35	33
ONOS-2.0.0	50	24
ONOS-2.1.0	59	32
ONOS-2.2.0	62	33
ONOS-2.3.0 [72]	41	24

Table II: Dependency Analysis of ONOS versions. VD: vulnerable dependencies, High: Dependencies with high severity level CVE.

bugs, e.g., Concurrency bugs, we can explore alternative and potentially lighter-weight techniques, e.g., semantic exploration techniques [64]. For example, a CORD concurrency bug (CORD-1734 [65]) where multiple interleaved threads caused performance degradation.

```
+ self.thread_pool = futures.
    ↪ ThreadPoolExecutor(max_workers=1)

- self.thread_pool = futures.
    ↪ ThreadPoolExecutor(max_workers=10)
```

Figure 4: Patch for CORD-1734 [65], where multiple threads were negatively impacting the performance of all API calls. This was attributed to reliance of python on global locks, so as a fix the maximum number of workers were reduced to 1.

New Research Directions: We summarize new research areas based on our observations:

- There are still gaps between the industrial demands and the modern invariant checkers as illustrated with FAUCET-1623 [56] (discussed above). To tackle such bugs with more complex behavior, we need more complex invariant checkers because most existing checkers focus on reachability-based and QoS-based invariants [57], [59].
- We identified a need for more fine-grained failure-indicators and failure-detectors that detect component level availability and correctness. These techniques need to be more expressive than simple heart-beats; they should verify subcomponent correctness. Specifically, for the failures that are due to load and ecosystem interactions, we may predict these crashes by analyzing metrics or existing syslogs. Given this, it would be interesting to evaluate the potential of extending existing log-based failure prediction systems [66]–[68] or metrics-based systems [69], [70] to SDNs.
- We highlighted a need for research into extending fault prediction based on system load to the SDN-domain to address issues with load and cascading errors, e.g., ONOS-4859 [71] that suffers from ineffective use of memory.

V. RQ3: Bug Triggers and Code Fixes

This section analyzes the events that trigger a bug, the code fixes applied to fix the bug(§ V-A), and the time to fix them (§ V-B).

A. Analysis of Bug Triggers

Recall, in Section II-A, we showed that SDN controllers are event-driven and, in general, these controllers only react to the events listed in § II-A. Below we analyze each of these events and discuss the implications for our study.

Configuration (38.8%): We observed that many bugs are triggered when the controller attempts to process system configurations. This fact is astounding because a critical motivation for SDN is to move towards automation and eliminate configuration-based errors [73]–[76].

In Table III, we analyze the type of configurations. We observe that for ONOS and CORD, most of the configuration bugs are due to the configuration of the controller and third-party services.

Interestingly, we observe that only 25% of the configuration-related bugs can be fixed by changing the controller configuration. This implies that research on misconfiguration [77]–[79] It focuses on detecting the impact of an application's configuration on the system and will have limited applicability because third-party code bases' configuration impacts the system.

Takeaway. These observations highlight a need for more research on techniques for diagnosing and debugging the cross-layer impact of configurations. These cross-layer approaches should be coupled with preventive systems such as [80] which detect latent configuration bugs by employing fuzzy-testing.

External Calls (33%): For the external calls, we observed that 41.4% of the code fixes attempt to make the controller more compatible with external libraries by changing function calls or arguments to match the external API or by upgrading the external packages. The use of code patches to fix this interdependence highlights the highly dynamic open-source ecosystem. Interestingly, we also observe that the misconfiguration of the communication between multiple modules is a non-trivial source of these problems. For example, in FAUCET-355 [81] (Figure 5), Guage crashed because of a misconfigured data type between Gauge and InfluxDB [82].

Moreover, as highlighted in prior work [83], a majority of open-source projects utilize outdated dependencies, which often makes the system vulnerable to attacks. SDNs are no exception; for example, in CVE-2018-1000615 [84] we observe that an outdated version of OVSDB [85] lead to a Denial of Service (DoS) attack on ONOS. In Table II, we provide a broader analysis of vulnerabilities in ONOS using dependency-check tool [86] and cross-checking with NVD [87]. Our analysis shows that ONOS' vulnerability increased over time as more dependencies were added with version updates. These vulnerabilities were fixed by changing the libraries, which makes them more critical.

Takeaway. A strong implication of this analysis is a need to design techniques to discover, track, and detect API mismatches. While techniques existing for tracking dependen-

Sub-categories of Configuration Bugs	FAUCET	ONOS	CORD
Controller	52.9%	60%	64.2%
Data Plane	11.7%	15%	14.2%
Third Party	35.4%	25%	21.6%

Table III: Sub-Categories for Configuration Bugs.

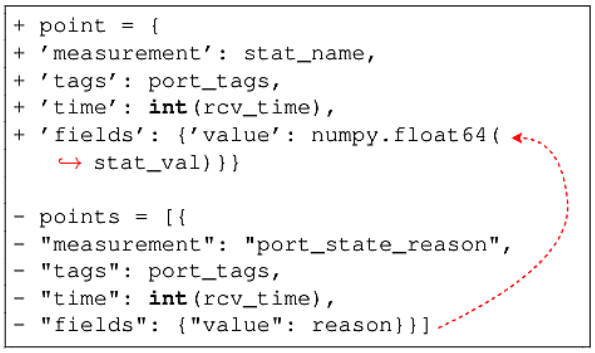

Figure 5: Patch for FAUCET- 355 [81], where InfluxDB [82] only supports one integer type, int64. But initially `OpenFlow` stats were logged as uint64 which were converted to float64 to prevent an overflow.

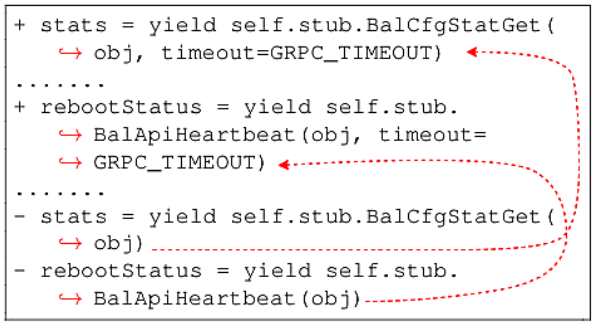

Figure 6: Patch for VOL-549 [91], where timeout was introduced for the GRPC connection to prevent VOLTHA from getting stuck when OLT was rebooted.

cies [88], [89], these techniques do not update the code when dependencies are intentionally updated.

Network Events (19.8%): Despite being designed to handle network events explicitly, the controller contains a non-trivial number of bugs (19.8%) that are triggered by when it processes network events. Specifically, these bugs are tried while the controller is attempting to process OpenFlow messages [2]. These bugs are often addressed by adding additional logic or adding exception handling code, indicating that the existing code is missing crucial logic for handling edge cases.

Takeaways. These observations highlight a need for novel fault tolerance techniques that either automatically rewrite code, or alter properties of the network event such that different code paths and cases are explored.

Hardware Reboots (8.4%): Hardware often reboots for a variety of reasons. Unsurprisingly a non-negligible set of bugs (8.4%) are due to these reboot events. Surprisingly, we observed that hardware reboot-triggered bugs are related to reboots of the optical components (e.g., ONU, OLT etc.), which points to the importance of tracking bindings between hardware configurations and their corresponding components in the abstraction layer (e.g., VOLTHA [90]). For example, in VOL-549 [91] (Figure 6), the VOLTHA core thread gets stuck waiting for the adapter to connect if OLT reboots after initial activation. This bug was fixed by adding a timeout variable.

Takeaways: Anecdotal evidence suggests that such bugs exist because testing environments lack representative failures and equipments [92]. This is a clear sign that emerging approaches to apply Chaos-Monkey style [93] fuzz testing to SDNs are needed, and more work should be done to extend the practicality of such techniques.

Broader Takeaways for Research: A significant set of bugs are due to interactions between the controller and external services (e.g., configuration files, network events, or function calls). These observations suggest that these controllers lack sufficient code for checking for valid inputs. Additionally, these bugs demonstrate a tight-coupling between the controller and the broader environment. As the environment evolves, care must be taken to ensure that the controller's codebase evolves accordingly. We need better tools to track dependencies and highlighting mismatches. Additionally, the developers of the SDN controllers need to introduce better error-guarding logic. Finally, while there is significant work [94]–[97] on addressing system misconfiguration, there is very little work within the SDN space.

[2]In particular, we observe that 44.4% are due to processing link/switch events (i.e., link-up or link-down), 33.3% due to Packet-In, and 22.3% due to `GetStatistics` message (i.e., counter related information).

B. Resolution Time for Triggers

Figure 7 shows the CDF for resolution times for bugs on the basis of the triggers categorised in Table I. In the above analysis, we observed that most bugs are triggered by configuration, but we also found it has the longest tail, which reveals that they are the most severe bug trigger category that could take considerable time to be resolved. It is observed that ONOS has a longer tail as compared to CORD in most of the trigger categories (Configuration, External call, Network event) which could be attributed to its more complex structure (LoC, classes, functionalities). For example, we found a serious ONOS-5992 [98] which impacted multiple versions before it could be fixed, and the fix required addressing multiple bugs: In this bug, killing one ONOS instance resulted in a cluster failure. On the contrary, we observed that bugs triggered by reboot have a longer tail for CORD than ONOS: this was because CORD has specialized code for disaggregated optical equipment, which involves complex configurations, e.g., EPON, GPON [99] and complex logic for tracking the state of these devices.

VI. Analysis of Software Engineering Principles

In this section, we analyze the software engineering practices of the different controllers. We start with an analysis of technical debt [100] (§ VI-A) and how it impacts code fixes. Then we perform a burn analysis (§ VI-B) of FAUCET to understand how changes to the codebase impact FAUCET's bugs and how they are triggered.

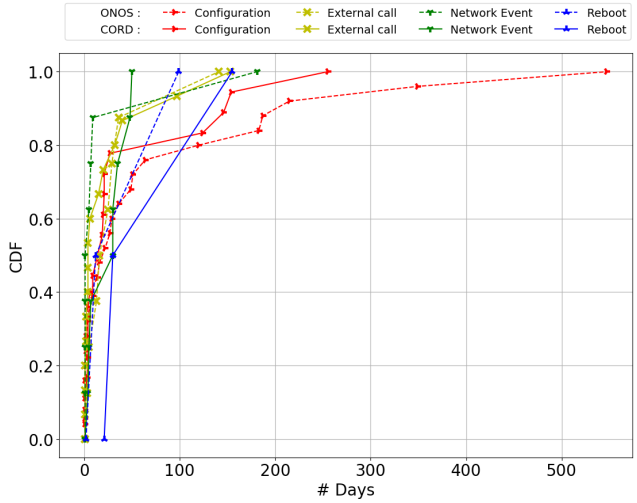

Figure 7: CDF of Resolution Time for Triggers.

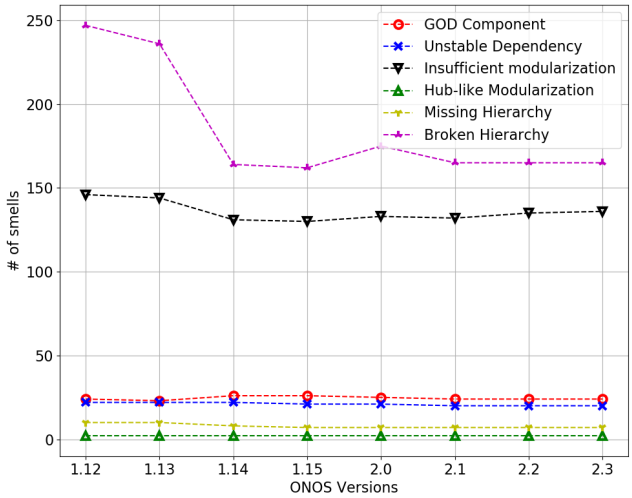

Figure 8: Distrubution of Six Code Smells. A: God Component, B: Unstable Dependency, C: Insufficient Modularization, D: Hub-like Modularization, E: Missing Hierarchy, F: Broken Hierarchy in ONOS Cores Versions.

A. *Smell-Analysis for Code-quality*

SDN controllers are subject to a large number of code changes over time to meet the evolving demands and fix existing bugs; however, such changes eventually lead to software technical debt [100] of software degradation. Code-smells is a popular software engineering technique for analyzing codebases to determine and capture a form of software degradation that is correlated to bugs [101]–[103]. We perform code-smell analysis on several different release versions of ONOS and analyze ONOS' software degradation over time. Additionally, we use the refactoring techniques [104] within the code-smell analysis to co-relate and understand the type of bug fixes, i.e., No Logic Changes, Add New Logic, Change Existing Logic.

We use Designite [105] for our code-smell analysis: Designite utilizes code-quality metrics, and it supports 19 architecture smells along with seven design smells. In Figure 8, we present the code-smell results for various ONOS releases. Next, we describe the smells and focus on those with the most variation across different versions of ONOS.

Broadly, there are two classes of smells: architecture and design. Architecture smells capture system-level impact spanning across multiple components, whereas design smell captures component level impact. Note: while plot Hub-like Modularization [106] and Missing Hierarchy [107], we do not analyze them because their numbers are low and they have slight variation across controller versions.

1) ***Architecture smell [108]:*** We observe that while the number of commits per release decreased or became constant (Figure 10), the architecture smells scores (i.e., God Component, and Unstable dependency smell, in Figure 8) remain constant. This constant architecture smell score, despite a decrease in commits, indicates constant technical debt. We believe this constant debt is potentially due to a gap between developer practices for developing patches and refactoring techniques. Next, we elaborate on the specific scores:

God component [109]. The God component captures the division of functionality across components and indicates code modularity, i.e., modularity of controller design. We observe in Figure 8 that the God component metric is mainly constant. Although the smell metric indicates the level of controller modularity is not growing, we observe that the average number of classes is growing for controllers; this implies that the controller architecture consists of huge classes that impact overall modularity. For example, while the metric remains stable, the package `net.intent.impl` had an increase in the number of classes from 49 to 107 from ONOS 1.12 to 2.3.0. We recommend that developers improve their codebase by making logical changes by decomposing huge classes and potentially changing the controller's Control-Flow graph.

Unstable dependency smells. This smell uses the State Dependency Principle (SDP) [110] to capture the stability of dependencies within the controller codebase. Unlike other smells, these can be difficult to refactor because modifying one dependency can lead to cascading changes to other dependencies. Fortunately, we observe in Figure 8 that the unstable dependency smells have decreased steadily from versions 1.12–2.3: this implies that developers can more freely make changes to dependencies without fear of introducing bugs.

2) ***Design smells [111]:*** As with any software package, ONOS's initial code releases consist of burst in commits due to prototyping new functionality with limited features and potentially unstable codebase: this is reflected in Figure 8 as an initial spike between versions 1.12–1.14 in the

Design smells scores (Insufficient modularization, Hub-like modularization, Missing hierarchy, and Broken Hierarchy). However, after version 1.14, we observed a steady decrease in the number of commits and that the Design smells remained unchanged or largely constant. We note that constant design smells are problematic because design smells have a causal relationship with architecture smells [112]: in short, design smells cause architecture smells, and thus to improving design smells will also improve architecture smells.

Insufficient modularization [106]. This metric captures the modularization of an individual class (Note: this differs from the God component, which captures package-level modularization features). In general, developers can improve this score by changing existing logic and decomposing large and complex classes.

Broken Hierarchy [107]. This smell analyzes the relationships between super-types and sub-types and checks to ensure that sub-types do implement features of their types. This smell is generally an indicator of missing logic. For example, in Figure 9, we present `Run class` which has the ElectorOperations super-type, note that the `Run` class doesn't include methods from its supertype ElectionOperation. After a major upgrade (ONOS-6594 [113]) which addressed severe architecture flaws, the `Run` class (and other related classes) was changed to be a subtype of AsyncLeaderElector – this change fixed the smell.

From Figure 8, we observe an initial spike in broken hierarchy smells (versions 1.12–1.14) demonstrating poor code modularization, and then we observe a reduction (versions 1.14–2.3) which indicates logic changes (add-logic, change existing logic) and restructuring of the existing methods. This conclusion supports the broad set of changes we observe for many of our bug fixes.

```
public static class Run
 extends ElectionOperation {

private String topic;
private NodeId nodeId;

public Run(String topic,
          NodeId nodeId) {
this.topic = topic;
this.nodeId = nodeId;
}
```

Figure 9: Broken Hierarchy in class `run` as it doesn't share an IS-A Relation with it's Super-type.

B. Burn Analysis

This section focuses our burn analysis on FAUCET because of its size (1000's LOC) and highly modular structure. Both properties make FAUCET an ideal candidate for burn analysis. Unfortunately, due to ONOS and CORD's complexity and the interleaving of components within individual source files, we are unable to apply burn analysis.

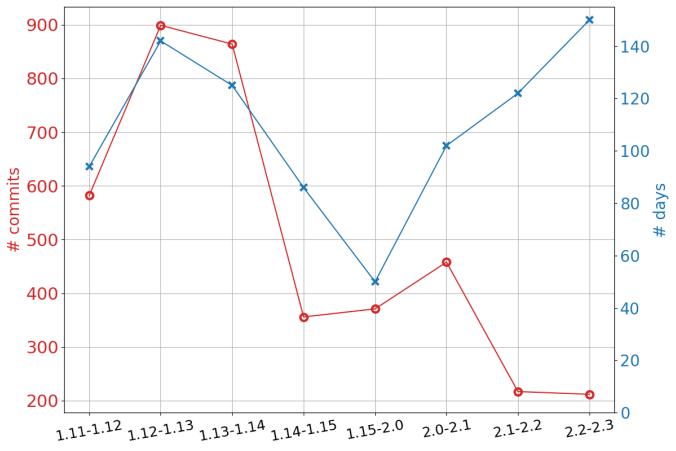

Figure 10: ONOS Github Analysis for number of Commits in each Version Upgrade.

We begin in Figure 11 by characterizing commits and changes to FAUCET's based on the functionality's triggering events: (1) Configuration (38%), (2) Network Functionality (35%), (3) External Abstraction (27%).

Unsurprisingly, we observe that most commits focus on increasing network function, which aligns with an SDN controller's central role, i.e., to provide control over the network. In particular, we observed that most commits are focused on fixing and adding new network functionalities.

The configuration-related commits are the second major category of commits. We believe this can be attributed to complex cross-layer configurations interactions identified in Section V.

Finally, External Libraries' dynamic nature poses a unique challenge for developers who need to make continual modifications to their code to ensure interoperability. To illustrate, In Table IV, we present a list of external dependencies for FAUCET and the number of version changes required. We observe that critical packages, e.g., RYU (network management framework) and chewie (IEEE 802.1x implementation) are subject to most changes and have shorter release cycles than the controller itself. This mismatch implies that the controller will always use outdated versions to introduce correct and security problems (as illustrated in Section V). For example, in FAUCET-2399 [114], an update to chewie prevented the installation of FAUCET. A move towards flexible versioning practices [115] with a balance between agility and predictability in core packages could reduce these bugs.

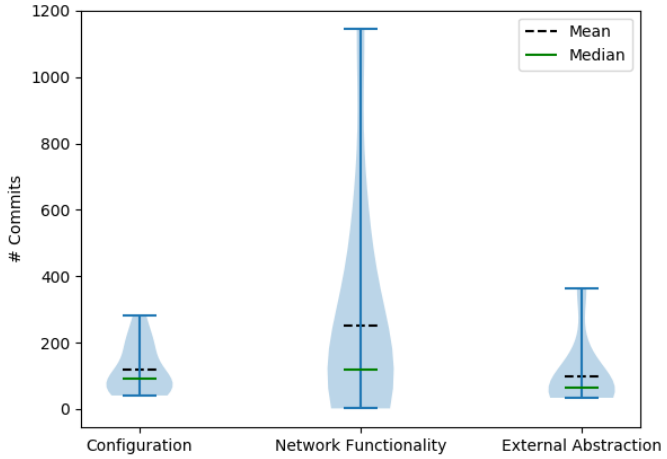

Figure 11: Distribution of Commits in FAUCET Core Across Three Functional Subsystems of a Controller. A: Configuration, B: Network Functionality, C: External Abstraction.

Dependency Name	# version changes	Description
chewie	19	802.1X standard implementation
eventlet	5	networking library
influxdb	1	time series database
msgpack	2	binary serialization
networkx	1	Network Analysis
pbr	1	management of setuptools packaging
prometheus_client	8	Monitoring system
pyyaml	6	YAML Parser
ryu	28	component-based SDN
beka	5	BGP Speaker
pytricia	1	IP Address Lookup

Table IV: Burn-down analysis for FAUCET dependency requirements.

VII. Broader Implications

In this section, we take a step back to understand the broader applicability and implications of our study on network operators. We focus on providing guidelines for (1) selecting controllers (§ VII-A), (2) debugging open issues (§ VII-B), and (3) navigating emerging diagnosis frameworks (§ VII-C).

A. RQ4: Controller Selection Guideline

Inspired by our observations, we provide general guidelines to aid operators in selecting controllers. Our guidelines focus on completeness, functionality, and SDN use cases. We observe that most problems in FAUCET are due to missing logic (specifically 52.5% of bugs), which makes it the least stable of the controllers that we analyzed. Although, CORD and ONOS are based on the same fundamental codebase, we observed that CORD is susceptible to significantly more load-related problems – 30% of bugs in CORD versus 16% in ONOS.

In Table VI, we show two critical use cases that SDN has enabled and the symptoms that affect the core functionality of these use cases. Building on the above observations, we recommend ONOS as the most stable and performant among the analyzed controllers. Unlike CORD, moving towards ONOS will require developers to find appropriate or develop applications due to a lack of rich applications. Moreover, we observed that FAUCET is specialized for a specific use-case, e.g., network slicing [131], [132]. Due to slicing's inherent isolation, we note that using it outside of this narrow use case will often yield missing functionality and logic errors.

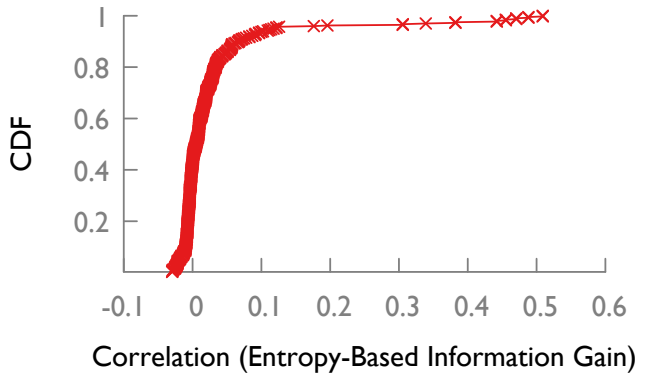

Figure 12: CDF of Bug Category Correlation.

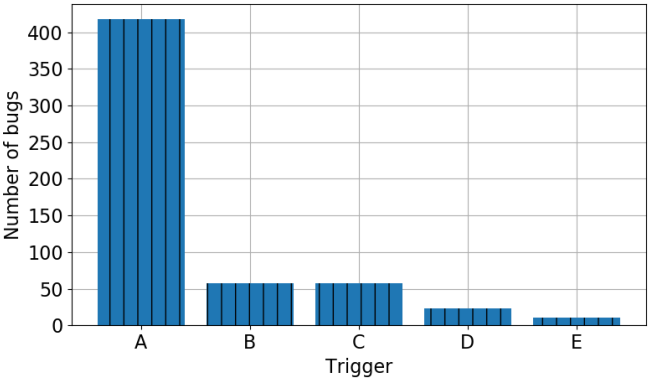

Figure 13: Trigger Distribution among the Whole Dataset. A: Configuration, B: System Calls, C: Third Party Calls, D: Network Events, E: Application Calls. B, C and E belong to External Calls.

B. RQ4: Automating Operators Diagnosis

In the absence of a tool for holistically diagnosing and resolving bugs, we conclude this section by providing guidelines for expediating root-cause diagnosis and resolution. We do this by analyzing the correlations between the bugs and categories (in Table I) and exploring the uniqueness of the keywords (AKA labels) in the bug descriptions.

Correlation Analysis: Figure 12 shows the CDF of correlations between all possible bug and category pairs. The curve illustrates that while most bug-category pairs (93.72% of bug) are fairly correlated, there is a long tail indicating strong-correlated bug categories (6.28% of bugs). For example, we observed that memory bugs are highly deterministic in nature. More interestingly, the bugs triggered by third-party service calls are highly correlated to the fix "add compatibility", which fits with the observations that these bugs could be caused by argument mismatch between library versions. Surprisingly, unlike bugs in the core controller, these third-party bugs are correlated with the outcomes "Error message" and "Byzantine".

	Deterministic		Bug Trigger			
	Yes [116]	**No [117]**	**Configuration [118]**	**Network Event [116], [119]**	**External calls [120]**	**Hardware Reboots [121], [122]**
LegoSDN [48]	✓	✓		✓		✓
Ravana [13]		✓	✓	✓		
SCL [14]		✓	✓	✓		
RoseMary [123]		✓				✓
SCOUT [124]		✓	✓			✓
JURY [125]		✓		✓		✓
DPQoAP [126]		✓		✓		✓

Table V: Effectiveness of Existing Recovery Techniques. C: Configuration, N: Network Event, E: External calls, and H: Hardware Reboots.

SDN use case	**Symptoms**			
	P [71]	**F [60]**	**EM [127]**	**B [56]**
Logically Centralized ([128], [129], [130])	✓	✓		✓
Network Slicing ([131], [132])				✓

Table VI: Symptoms of the bugs affecting SDN use case network operation drastically. EM: Error Message B: Byzantine F: Fails-stop P: Performance.

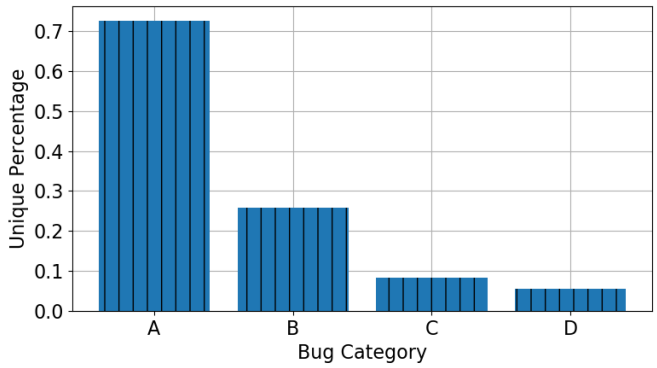

Figure 14: Unique Topic Percentage. A: Deterministic, B: Byzantine, C: Add Synchronization, D: Third Party Calls.

Keyword Analysis: To further understand these correlations, we analyzed the topics extracted by the NLP techniques. We hypothesize that these correlations reflect that specific classes of bugs have unique topics or keywords in the bug description. For example, memory bugs often have a null pointer and other similar exceptions in the bug description. In Figure 14, we listed the top bug categories based on topic uniqueness. We observe that these bug categories are the exact bug categories that have a high correlation discussed earlier. We observe that the uniqueness in topics spreads over all bug classifications. Specifically, bugs with Byzantine symptoms introduce significantly different topics and keywords in the bug description. Similarly, some bug types, e.g., deterministic bugs, have remarkably unique topics.

We also apply our NLP model, which is trained with the manually labeled dataset, onto the whole dataset of critical bugs we get from Jira to demonstrate NLP techniques' potential further. This large dataset contains ∼5X bugs compared to our manually labeled dataset. Figure 13 is the distribution of predicted trigger from the whole dataset. The result indicates that configuration error is the major trigger of SDN controller bugs, and when troubleshooting an SDN controller, the operator should pay more attention to potential configuration glitches. Compared to configuration, the bugs triggered by OpenFlow events only contribute a small part. Given the complexity of capturing, replaying the network events to reproduce a previous scenario, it is more clever to examine the network events after ensuring other more critical potential triggers. We also summarized the results for other aspects, such as the deterministic bug is the dominant bug type. Due to the limit of space, we skip the details in this paper.

Takeaway. These correlations and keyword analysis imply that for a non-trivial amount of bugs, being able to identify outcomes, symptoms, and extract keywords from the bug will allow developers and operators to narrow down the potential root causes and fixes. As part of future work, we anticipate that a decision tree can be developed to help restrict and narrow the developer and operator efforts in diagnosis.

C. RQ5: Selecting Recovery Frameworks

In Table V, we present a survey of existing fault tolerance techniques for SDNs. A key observation here is that no one technique can recover from bugs across all root causes effectively. Unsurprisingly, most techniques [13], [14], [48], [123]–[125] are able to recover from events triggered by OpenFlow messages which is the main focus of most SDN research. Yet, there are very few existing works within the SDN domain for interactions with configuration and external calls. We note that while non-SDN techniques, e.g., Lock-in-Pop [133], can address external events or configuration, these techniques need to be modified to address domain-specific issues.

We observe that most existing systems can easily recover from non-deterministic issues. However, there is very little for deterministic issues that account for most of the problems (as shown in Section III).

	Category	SDN	Cloud [19]	BGP [35]
Symptoms	Fail-stop	20%	59%	39%
	Performance	4%	14%	NA
	Error Message	14.7%	NA	NA
	Byzantine	61.33%	25%	58%

Table VII: Analysis of Bug Symptoms Accross Related Work.

Takeaway. Although we showed a plethora of systems that can diagnose or recover from different types of bugs, in practice, it is not trivial to combine these systems together to form a holistic system for the following reasons:

- Simply layering the systems on each other may introduce inefficiencies or impact accuracy. For example, while SPHINX [134] requires that all input OpenFlow messages to update a "flow graph"-based model, Bouncer [135] proactively filters out some input which may lead to an inconsistent flow graph and, thus, impacts accuracy [3].
- Additionally, their expected inputs and system models are often fundamentally different; thus, integration is a non-trivial task. For example, while SOFT [136] analyzes output generated by different vendor implementations and CHIMP [137] analyzes output from different SDN applications, it is unclear how to compose the results from SOFT and CHIMP to provide a holistic, cross-layer approach to fault detection.

VIII. Threats to validity and discussions

Generalizability. While limited, we believe that our analysis generalizes to future controllers because related work has shown that controllers follow a limited set of design principles that are well represented in the controllers that we studied. Specifically, the three controllers that we analyzed provide coverage over the following design choices: specifically, specialized (CORD) versus generalized (ONOS, FAUCET); monolithic (FAUCET) versus modular (ONOS, CORD); and distributed (ONOS, CORD) versus centralized (FAUCET).

Automated SE Analysis. Our automated code analysis is limited by the constraints of existing software engineering analysis tools, which only support specific languages (JAVA) or specific build systems (maven, gradle). For example, we could not perform smell analysis for FAUCET because it is written in Python, and the smell analysis codebase only supports JAVA-based software. Unfortunately, this limitation limits our ability to perform this analysis on a broader set of controllers.

Different bug management systems. The controllers use different bug management systems, e.g., GitHub (FAUCET), JIRA (ONOS, CORD), which could lead to variation in the type of information available. For example, JIRA provides Gerrit reviews, bug status, timestamps, etc while GitHub provides a different subset of data. These subtle differences impact the set of techniques, tools, and analysis that we could apply. For example, we could not analyze FAUCET's resolution times because their GitHub repository does not provide this information.

[3]The whole point for flow graph is performing anomaly detection, which requires all inputs including the bad ones.

Manual Classification. Our work involves both manual and automated analysis. While the automated analysis is susceptible to noise and bias, we note that we only use the automated analysis to support our manual analysis. In fact, most of our takeaways are based on manual analysis, thus minimizing the impact of learning-based noise on our observations. Our manual analysis's validity is predicated on the fact that the bugs are accurately described and reported.

IX. Related Works

System-Research. In general, bug studies spanning across various domains [18], [19], [35], [138]–[141] lay the foundation for systems research. While prior studies have focused on distributed systems, we lack similar in-depth and comprehensive studies for SDN controllers. Unsurprisingly, we observed that, despite using a similar classification as prior work [18], [19], bugs in SDN controllers differ significantly in their distributions, motivating the need for studies such as ours.

SDN Bug Studies. Prior work on SDN bugs [11], [12], [142]–[145] analyze a smaller spectrum of bugs compared with our study, which provides a holistic and in-depth analysis of 'critical' SDN bugs. While our work focuses on understanding bugs and their implications, others [143]–[145] have developed stochastic models to help quantify the reliability of existing controllers.

X. Conclusion

Bugs are a crucial aspect of any software ecosystem, yet within the software-defined networking (SDN) community, we have a poor understanding of our bugs. Without a thorough understanding of these bugs, it is challenging to: (1) understand the efficacy of existing SDN fault tolerance techniques, (2) design representative fault injectors, or (3) identity key areas that are ripe for research. In this paper, our goal is to provide the knowledge required to fill this crucial gap in the community's understanding of the SDN ecosystem by performing, to date, the largest bug study over three popular controller platforms.

XI. Acknowledgements

We thank the anonymous reviewers and our shepherd, Marco Vieira, for their helpful comments. This work was supported by NSF award CNS-1749785.

References

[1] "At&t sdwan details retrieved from," https://www.business.att.com/products/sd-wan.html, 2019, accessed: 10-6-2019.
[2] "Vodafone sdn details retrieved from," https://www.vodafone.co.uk/business/sdn, 2019, accessed: 13-11-2019.
[3] "How sdn simplifies managing digital experiences," https://www.orange-business.com/en/blogs/connecting-technology/networks/how-sdn-simplifies-managing-digital-experiences, 2019, accessed: 13-11-2019.
[4] "Nsx data center details retrieved from," https://www.vmware.com/in/products/nsx.html, 2019, accessed: 10-6-2019.
[5] "The andromeda cloud platform details retrieved from," 2019, https://www.ngcsoftware.com/landing/ngcandromedacloudplatform/.

[6] J. Son and R. Buyya, "A taxonomy of software-defined networking (sdn)-enabled cloud computing," ACM Comput. Surv., vol. 51, no. 3, pp. 59:1–59:36, May 2018. [Online]. Available: http://doi.acm.org/10.1145/3190617

[7] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure," in Proceedings of the 2016 ACM SIGCOMM Conference, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 58–72. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934891

[8] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486019

[9] T. Lei, Z. Lu, X. Wen, X. Zhao, and L. Wang, "Swan: An sdn based campus wlan framework," in 2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE), May 2014, pp. 1–5.

[10] S. Choi, B. Burkov, A. Eckert, T. Fang, S. Kazemkhani, R. Sherwood, Y. Zhang, and H. Zeng, "Fboss: Building switch software at scale," in Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 342–356. [Online]. Available: https://doi.org/10.1145/3230543.3230546

[11] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE way to test openflow applications," in NSDI. San Jose, CA: USENIX, 2012, pp. 127–140. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/canini

[12] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, H. Acharya, K. Zarifis, and S. Shenker, "Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences," ACM SIGCOMM Computer Communication Review, vol. 44, 08 2014.

[13] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller Fault-tolerance in Software-defined Networking," in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:12. [Online]. Available: http://doi.acm.org/10.1145/2774993.2774996

[14] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: Simplifying Distributed SDN Control Planes," in Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 329–345. [Online]. Available: http://dl.acm.org/citation.cfm?id=3154630.3154657

[15] T. Xu and Y. Zhou, "Systems approaches to tackling configuration errors: A survey," ACM Comput. Surv., vol. 47, no. 4, Jul. 2015. [Online]. Available: https://doi.org/10.1145/2791577

[16] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ser. SOSP '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 159–172. [Online]. Available: https://doi.org/10.1145/2043556.2043572

[17] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, 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, 2015.

[18] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria, "What bugs live in the cloud? a study of 3000+ issues in cloud systems," in Proceedings of the ACM Symposium on Cloud Computing, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 7:1–7:14. [Online]. Available: http://doi.acm.org/10.1145/2670979.2670986

[19] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar, "Why does the cloud stop computing?: Lessons from hundreds of service outages," in Proceedings of the Seventh ACM Symposium on Cloud Computing, ser. SoCC '16. New York, NY, USA: ACM, 2016, pp. 1–16. [Online]. Available: http://doi.acm.org/10.1145/2987550.2987583

[20] S. Li, K. Han, N. Ansari, Q. Bao, D. Hu, J. Liu, S. Yu, and Z. Zhu, "Improving sdn scalability with protocol-oblivious source routing: A system-level study," IEEE Transactions on Network and Service Management, vol. 15, no. 1, pp. 275–288, March 2018.

[21] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, and M. Zhu, "A load balancing strategy of sdn controller based on distributed decision," in 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Sep. 2014, pp. 851–856.

[22] N. Paladi and C. Gehrmann, "SDN access control for the masses," CoRR, vol. abs/1811.08094, 2018. [Online]. Available: http://arxiv.org/abs/1811.08094

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[24] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 3920, Oct. 2004. [Online]. Available: https://rfc-editor.org/rfc/rfc3920.txt

[25] "Comcast leads trellis, an open source data center switching fabric," https://www.sdxcentral.com/articles/news/comcast-leads-trellis-an-open-source-data-center-switching-fabric/2018/12/, 2018, accessed: 27-04-2020.

[26] "Google joins cord, now a separate open source group," https://www.sdxcentral.com/articles/news/google-joins-cord-now-separate-open-source-group/2016/07/, 2016, accessed: 27-04-2020.

[27] "Taking seba to production," https://www.youtube.com/watch?v=IEGuXxwbbnQ&feature=youtu.be&ab_channel=OpenNetworkingFoundation, 2020, accessed: 5-12-2020.

[28] "M-cord," https://opennetworking.org/m-cord/, 2020, accessed: 5-12-2020.

[29] J. Bailey and S. Stuart, "Faucet: Deploying sdn in the enterprise," ACM Queue, vol. 14 Issue 5, pp. 54–68, 2016.

[30] "Details retrieved from," https://github.com/faucetsdn/faucet, 2019, accessed: 6-5-2019.

[31] "Introduction to faucet," https://docs.faucet.nz/en/latest/intro.html#what-is-faucet, 2019, accessed: 10-6-2019.

[32] "Onos project details retrieved from," https://onosproject.org/, 2019, accessed: 10-11-2018.

[33] "Opencord details retrieved from," https://opencord.org/, 2019, accessed: 10-11-2018.

[34] "Xos details retrieved from," https://opennetworking.org/xos/, 2021, accessed: 31-03-2021.

[35] Z. Yin, M. Caesar, and Y. Zhou, "Towards Understanding Bugs in Open Source Router Software," SIGCOMM Comput. Commun. Rev., vol. 40, no. 3, pp. 34–40, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1823844.1823849

[36] L. Peterson, "Mysterious-Decision Release Notes," https://wiki.opencord.org/display/CORD/Mysterious-Decision+Release+Notes, 2017, accessed: 6-2019.

[37] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 3111–3119. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999792.2999959

[38] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," J. Mach. Learn. Res., vol. 3, pp. 993–1022, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944937

[39] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical dirichlet processes," 2006.

[40] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in Advances in Neural Information Processing Systems 13, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 556–562. [Online]. Available: http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf

[41] J. E. Ramos, "Using tf-idf to determine word relevance in document queries," 2003.

[42] M. Serifovic, " Image-to-Recipe Translation with Deep Convolutional Neural Networks ," https://towardsdatascience.com/this-ai-is-hungry-b2a8655528be, 2018.

[43] T. U. of British Columbia, " A Comparison of LDA and NMF for Topic Modeling on Literary Themes ," https://wiki.ubc.ca/Course:CPSC522/A_Comparison_of_LDA_and_NMF_for_Topic_Modeling_on_Literary_Themes, 2018.

[44] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M. . Wong, "Orthogonal defect classification-a concept for in-process measurements," IEEE Transactions on Software Engineering, vol. 18, no. 11, pp. 943–956, 1992.

[45] L. Lamport, "The part-time parliament," ACM Trans. Comput. Syst., vol. 16, no. 2, pp. 133–169, May 1998. [Online]. Available: http://doi.acm.org/10.1145/279227.279229

[46] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 305–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643666

[47] S. M. Srinivasan, S. Kandula, C. R. Andrews, and Y. Zhou, "Flashback: A lightweight extension for rollback and deterministic replay for software debugging," in Proceedings of the Annual Conference on USENIX Annual Technical Conference, ser. ATEC '04. USA: USENIX Association, 2004, p. 3.

[48] B. Chandrasekaran, B. Tschaen, and T. Benson, "Isolating and tolerating sdn application failures with legosdn," in Proceedings of the Symposium on SDN Research, ser. SOSR '16. New York, NY, USA: ACM, 2016, pp. 7:1–7:12. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890965

[49] F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, "Rx: Treating bugs as allergies—a safe method to survive software failures," SIGOPS Oper. Syst. Rev., vol. 39, no. 5, pp. 235–248, Oct. 2005. [Online]. Available: http://doi.acm.org/10.1145/1095809.1095833

[50] M. Rinard, C. Cadar, D. Dumitran, D. M. Roy, T. Leu, and W. S. Beebee, Jr., "Enhancing server availability and security through failure-oblivious computing," in Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251275

[51] Y. Wu, A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo, "Automated bug removal for software-defined networks," in Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 719–733. [Online]. Available: http://dl.acm.org/citation.cfm?id=3154630.3154688

[52] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in Proceedings of the 2013 International Conference on Software Engineering, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 802–811. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486893

[53] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, "Semfix: Program repair via semantic analysis," in Proceedings of the 2013 International Conference on Software Engineering, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 772–781. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486890

[54] J. H. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan, W.-F. Wong, Y. Zibin, M. D. Ernst, and M. Rinard, "Automatically patching errors in deployed software," in Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 87–102. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629585

[55] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in Proceedings of the 34th International Conference on Software Engineering, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 3–13. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337225

[56] "Faucet issue #1623 details retrieved from," https://github.com/faucetsdn/faucet/pull/1623, 2019, accessed: 6-5-2019.

[57] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 15–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482630

[58] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat, "Libra: Divide and conquer to verify forwarding tables in huge networks," in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). Seattle, WA: USENIX Association, Apr. 2014, pp. 87–99. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/zeng

[59] 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 the 10th USENIX Conference on Networked Systems Design and Implementation, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 99–112. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482638

[60] "Cord-2470 details retrieved from," https://jira.opencord.org/browse/CORD-2470, 2019, accessed: 6-5-2019.

[61] R. Jung, Germany, J.-H. Jourdan, R. Krebbers, and D. Dreyer, "Safe systems programming in rust: The promise and the challenge," 2020.

[62] H. Xu, Z. Chen, M. Sun, and Y. Zhou, "Memory-safety challenge considered solved? an empirical study with all rust cves," 2020.

[63] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, "Fail-slow at scale: Evidence of hardware performance faults in large production systems," in 16th USENIX Conference on File and Storage Technologies (FAST 18). Oakland, CA: USENIX Association, Feb. 2018, pp. 1–14. [Online]. Available: https://www.usenix.org/conference/fast18/presentation/gunawi

[64] P. Fonseca, C. Li, and R. Rodrigues, "Finding complex concurrency bugs in large multi-threaded applications," in Proceedings of the Sixth Conference on Computer Systems, ser. EuroSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 215–228. [Online]. Available: https://doi.org/10.1145/1966445.1966465

[65] "Details retrieved from," https://jira.opencord.org/browse/CORD-1734, 2019, accessed: 12-12-2020.

[66] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs," in ACM SIGPLAN Notices, vol. 51, no. 4. ACM, 2016, pp. 489–502.

[67] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017, pp. 1285–1298.

[68] Z. Li, Z. Ge, A. Mahimkar, J. Wang, B. Y. Zhao, H. Zheng, J. Emmons, and L. Ogden, "Predictive Analysis in Network Function Virtualization," in Proceedings of the Internet Measurement Conference 2018. ACM, 2018, pp. 161–167.

[69] C. Streiffer, R. Raghavendra, T. Benson, and M. Srivatsa, "Learning to Simplify Distributed Systems Management," in 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018, pp. 1837–1845.

[70] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," in Proceedings International Conference on Dependable Systems and Networks. IEEE, 2002, pp. 595–604.

[71] "Onos-4859 details retrieved from," https://jira.onosproject.org/browse/ONOS-4859, 2019, accessed: 6-5-2019.

[72] "Onos-2.3.0 details retrieved from," https://github.com/opennetworkinglab/onos/tree/ac329a787311c731aac4b7408e5749590bd816fe, 2019, accessed: 26-01-2020.

[73] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 87–98, 2014.

[74] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A Clean Slate 4D Approach to Network Control and Management," ACM SIGCOMM Computer Communication Review, vol. 35, no. 5, pp. 41–54, 2005.

[75] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in ACM SIGCOMM Computer Communication Review, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[76] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of A Routing Control Platform," in Proceedings of the 2nd conference on Symposium on

Networked Systems Design & Implementation-Volume 2. USENIX Association, 2005, pp. 15–28.
[77] A. Rabkin and R. H. Katz, "How Hadoop Clusters Break," IEEE software, vol. 30, no. 4, pp. 88–94, 2012.
[78] M. Attariyan and J. Flinn, "Automating Configuration Troubleshooting with Dynamic Information Flow Analysis," in OSDI, vol. 10, no. 2010, 2010, pp. 1–14.
[79] ——, "Automating configuration troubleshooting with confaid," login Usenix Mag., vol. 36, no. 1, 2011. [Online]. Available: https://www.usenix.org/publications/login/february-2011-volume-36-number-1/automating-configuration-troubleshooting-confaid
[80] T. Xu, X. Jin, P. Huang, Y. Zhou, S. Lu, L. Jin, and S. Pasupathy, "Early detection of configuration errors to reduce failure damage," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). Savannah, GA: USENIX Association, Nov. 2016, pp. 619–634. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/xu
[81] "Faucet issue #355 details retrieved from," https://github.com/faucetsdn/faucet/pull/355, 2019, accessed: 6-5-2019.
[82] "Influxdb details retrieved from," https://www.influxdata.com/, 2019, accessed: 6-5-2019.
[83] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" Empirical Softw. Engg., vol. 23, no. 1, p. 384–417, Feb. 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9521-5
[84] "Cve-2018-1000615 details retrieved from," https://www.cvedetails.com/cve/CVE-2018-1000615/, 2019, accessed: 1-6-2020.
[85] "Ovsdb details retrieved from," http://docs.openvswitch.org/en/latest/ref/ovsdb.7/, 2019, accessed: 1-6-2020.
[86] "Dependency-check details retrieved from," https://jeremylong.github.io/DependencyCheck/, 2020, accessed: 26-06-2020.
[87] "Details retrieved from," https://nvd.nist.gov/, 2020, accessed: 26-06-2020.
[88] S. Mirhosseini and C. Parnin, "Can Automated Pull Requests Encourage Software Developers to Upgrade Out-of-date Dependencies?" in Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, 2017, pp. 84–94.
[89] M. Lungu, R. Robbes, and M. Lanza, "Recovering Inter-Project Dependencies in Software Ecosystems," in Proceedings of the IEEE/ACM international conference on Automated software engineering. ACM, 2010, pp. 309–312.
[90] "Voltha details retrieved from," https://www.opennetworking.org/voltha/, 2020, accessed: 26-06-2020.
[91] "Vol-549 details retrieved from," https://jira.opencord.org/browse/VOL-549, 2019, accessed: 6-5-2019.
[92] "Chaos monkey details retrieved from," https://github.com/Netflix/chaosmonkey, 2019, accessed: 14-3-2019.
[93] N. Shelly, B. Tschaen, K.-T. Förster, M. Chang, T. Benson, and L. Vanbever, "Destroying Networks for Fun (and Profit)," in Proceedings of the 14th ACM Workshop on Hot Topics in Networks. ACM, 2015, p. 6.
[94] L. Keller, P. Upadhyaya, and G. Candea, "Conferr: A tool for assessing resilience to human configuration errors," in 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), June 2008, pp. 157–166.
[95] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 237–250. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924960
[96] A. Whitaker, R. S. Cox, and S. D. Gribble, "Configuration debugging as search: Finding the needle in the haystack," in Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251260
[97] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma, "Automated known problem diagnosis with event traces," in Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, ser. EuroSys '06. New York, NY, USA: ACM, 2006, pp. 375–388. [Online]. Available: http://doi.acm.org/10.1145/1217935.1217972
[98] "Onos-5992 details retrieved from," https://jira.onosproject.org/browse/ONOS-5992, 2020, accessed: 10-12-2020.
[99] "An analysis of xpon technology development," https://www.zte.com.cn/global/about/magazine/zte-technologies/2007/10/en_120/161896.html, 2021, accessed: 01-04-2021.
[100] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," IEEE Software, vol. 29, no. 6, pp. 18–21, 2012.
[101] I. Ahmed, U. A. Mannan, R. Gopinath, and C. Jensen, "An empirical study of design degradation: How software projects get worse over time," in 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2015, pp. 1–10.
[102] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "The evolution and impact of code smells: A case study of two open source systems," in 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 390–400.
[103] W. Li and R. Shatnawi, "An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution," Journal of Systems and Software, vol. 80, no. 7, pp. 1120 – 1128, 2007, dynamic Resource Management in Distributed Real-Time Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121206002780
[104] "Front matter," in Refactoring for Software Design Smells, G. Suryanarayana, G. Samarthyam, and T. Sharma, Eds. Boston: Morgan Kaufmann, 2015, p. iii. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128013977010018
[105] T. Sharma, P. Mishra, and R. Tiwari, "Designite: A software design quality assessment tool," in Proceedings of the 1st International Workshop on Bringing Architectural Design Thinking into Developers' Daily Activities, ser. BRIDGE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1–4. [Online]. Available: https://doi.org/10.1145/2896935.2896938
[106] G. Suryanarayana, G. Samarthyam, and T. Sharma, "Chapter 5 - modularization smells," in Refactoring for Software Design Smells, G. Suryanarayana, G. Samarthyam, and T. Sharma, Eds. Boston: Morgan Kaufmann, 2015, pp. 93 – 122. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128013977000059
[107] ——, "Chapter 6 - hierarchy smells," in Refactoring for Software Design Smells, G. Suryanarayana, G. Samarthyam, and T. Sharma, Eds. Boston: Morgan Kaufmann, 2015, pp. 123 – 192. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128013977000060
[108] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Toward a catalogue of architectural bad smells," in Architectures for Adaptive Software Systems, R. Mirandola, I. Gorton, and C. Hofmeister, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 146–162.
[109] M. Lippert and S. Roock, "Refactoring in large software projects," 2006.
[110] M. Noback, The Stable Dependencies Principle. Berkeley, CA: Apress, 2018, pp. 217–235. [Online]. Available: https://doi.org/10.1007/978-1-4842-4119-6_10
[111] G. Suryanarayana, G. Samarthyam, and T. Sharma, "Chapter 2 - design smells," in Refactoring for Software Design Smells, G. Suryanarayana, G. Samarthyam, and T. Sharma, Eds. Boston: Morgan Kaufmann, 2015, pp. 9 – 19. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128013977000023
[112] T. Sharma, P. Singh, and D. Spinellis, "An empirical investigation on the relationship between design and architecture smells," Empirical Software Engineering, vol. 25, no. 5, pp. 4020–4068, Sep 2020. [Online]. Available: https://doi.org/10.1007/s10664-020-09847-2
[113] "Onos-6594 details retrieved from," https://jira.onosproject.org/browse/ONOS-6594, 2019, accessed: 26-01-2020.
[114] "Faucet codebase details retrieved from," https://github.com/faucetsdn/faucet/pull/2399, 2019, accessed: 6-5-2019.
[115] "Dependency versioning in the wild," in 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), 2019, pp. 349–359.
[116] "Onos-5309 details retrieved from," https://jira.onosproject.org/browse/ONOS-5309, 2019, accessed: 6-5-2019.
[117] "Vol-1201 details retrieved from," https://jira.opencord.org/browse/VOL-1201, 2019, accessed: 6-5-2019.
[118] "Onos-6893 details retrieved from," https://jira.onosproject.org/browse/ONOS-6893, 2019, accessed: 6-5-2019.

[119] "Faucet issue #489 details retrieved from," https://github.com/faucetsdn/faucet/pull/489, 2019, accessed: 6-5-2019.

[120] "Cord-2687 details retrieved from," https://jira.opencord.org/browse/CORD-2687, 2019, accessed: 6-5-2019.

[121] "Onos-2015 details retrieved from," https://jira.onosproject.org/browse/ONOS-2015, 2019, accessed: 6-5-2019.

[122] "Vol-1122 details retrieved from," https://jira.opencord.org/browse/VOL-1122, 2019, accessed: 6-5-2019.

[123] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A robust, secure, and high-performance network operating system," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 78–89. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660353

[124] P. Tammana, C. Nagarajan, P. Mamillapalli, R. Kompella, and M. Lee, "Fault Localization in Large-Scale Network Policy Deployment," in 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018, pp. 54–64.

[125] K. Mahajan, R. Poddar, M. Dhawan, and V. Mann, "Jury: Validating Controller Actions in Software-Defined Networks," in 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2016, pp. 109–120.

[126] B. Yamansavascilar, A. C. Baktir, A. Ozgovde, and C. Ersoy, "Fault tolerance in sdn data plane considering network and application based metrics," Journal of Network and Computer Applications, vol. 170, p. 102780, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S108480452030254X

[127] "Onos-7758 details retrieved from," https://jira.onosproject.org/browse/ONOS-7758, 2019, accessed: 6-5-2019.

[128] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? state distribution trade-offs in software defined networks," in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1–6. [Online]. Available: https://doi.org/10.1145/2342441.2342443

[129] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference, 2012, pp. 1–8.

[130] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ser. HotSDN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 85–90. [Online]. Available: https://doi.org/10.1145/2620728.2620739

[131] R. Sherwood, G. Gibb, K. kiong Yap, M. Casado, N. Mckeown, and G. Parulkar, "Flowvisor: A network virtualization layer," Tech. Rep., 2009.

[132] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, "Network virtualization in multi-tenant datacenters," in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). Seattle, WA: USENIX Association, Apr. 2014, pp. 203–216. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen

[133] Y. Li, B. Dolan-Gavitt, S. Weber, and J. Cappos, "Lock-in-Pop: Securing Privileged Operating System Kernels by Keeping on the Beaten Path," in ATC), 2017, pp. 1–13.

[134] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting Security Attacks in Software-Defined Networks," 01 2015.

[135] M. Costa, M. Castro, L. Zhou, L. Zhang, and M. Peinado, "Bouncer: Securing Software by Blocking Bad Input," ACM SIGOPS Operating Systems Review, vol. 41, no. 6, pp. 117–130, 2007.

[136] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT Way for OpenFlow Switch Interoperability Testing," in Proceedings of the 8th international conference on Emerging networking experiments and technologies. ACM, 2012, pp. 265–276.

[137] T. Nelson, A. D. Ferguson, and S. Krishnamurthi, "Static differential program analysis for software-defined networks," in FM 2015: Formal Methods, N. Bjørner and F. de Boer, Eds. Cham: Springer International Publishing, 2015, pp. 395–413.

[138] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A Comprehensive Study on Deep Learning Bug Characteristics," arXiv preprint arXiv:1906.01388, 2019.

[139] J. Wang, W. Dou, Y. Gao, C. Gao, F. Qin, K. Yin, and J. Wei, "A Comprehensive Study on Real World Concurrency Bugs in Node.js," in 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2017, pp. 520–531.

[140] H. Liu, S. Lu, M. Musuvathi, and S. Nath, "What Bugs Cause Production Cloud Incidents?" in HotOS, 2019, pp. 155–162.

[141] Y. Gao, W. Dou, F. Qin, C. Gao, D. Wang, J. Wei, R. Huang, L. Zhou, and Y. Wu, "An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems," in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2018, pp. 539–550.

[142] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu, "Attacking the brain: Races in the sdn control plane," in Proceedings of the 26th USENIX Conference on Security Symposium, ser. SEC'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 451–468. [Online]. Available: http://dl.acm.org/citation.cfm?id=3241189.3241225

[143] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, A. Blenk, W. Kellerer, and C. Mas Machuca, "Assessing the maturity of sdn controllers with software reliability growth models," IEEE Transactions on Network and Service Management, vol. 15, no. 3, pp. 1090–1104, 2018.

[144] P. Vizarreta, E. Sakic, W. Kellerer, and C. M. Machuca, "Mining software repositories for predictive modelling of defects in sdn controller," in 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019, pp. 80–88.

[145] P. Vizarreta, K. Trivedi, V. Mendiratta, W. Kellerer, and C. Mas-Machuca, "Dason: Dependability assessment framework for imperfect distributed sdn implementations," IEEE Transactions on Network and Service Management, vol. 17, no. 2, pp. 652–667, 2020.