Mining Software Repositories for Predictive Modelling of Defects in SDN Controller

Petra Vizarreta, Ermin Sakic, Wolfgang Kellerer and Carmen Mas Machuca
Technical University of Munich, Chair of Communication Networks
petra.vizarreta@lkn.ei.tum.de, ermin.sakic@{tum.de,siemens.com}, wolfgang.kellerer@tum.de, cmas@tum.de

***Abstract*—In Software Defined Networking (SDN) control plane of forwarding devices is concentrated in the SDN controller, which assumes the role of a network operating system. Big share of today's commercial SDN controllers are based on *OpenDaylight*, an open source SDN controller platform, whose bug repository is publicly available. In this article we provide a first insight into 8k+ bugs reported in the period over five years between March 2013 and September 2018. We first present the functional components in *OpenDaylight* architecture, localize the most vulnerable modules and measure their contribution to the total bug content. We provide high fidelity models that can accurately reproduce the stochastic behaviour of bug manifestation and bug removal rates, and discuss how these can be used to optimize the planning of the test effort, and to improve the software release management. Finally, we study the correlation between the code internals, derived from the Git version control system, and software defect metrics, derived from Jira issue tracker. To the best of our knowledge, this is the first study to provide a comprehensive analysis of bug characteristics in a production grade SDN controller.**

I. Introduction

With Software Defined Networking (SDN), the distributed control plane logic of forwarding devices is moved to a software entity called SDN controller, effectively decoupling the control plane (e.g., path computation) from data plane functions (i.e., switching). The controller provides an integrated interface towards diverse set of forwarding devices, and abstracts the network to the users and applications, which can now program the high level policies without minding the hardware specific implementation details. Present-day production grade SDN controllers additionally provide the support the legacy network protocols and hybrid devices, advanced security features, automated bootstrapping and interworking with virtualization platforms and cloud management systems. The heterogeneity of supported networks and services resulted in the controllers becoming rather complex and presumably buggy software system.

Software bugs[1] are the major issue in modern networks, as shown by the recent studies on Google [1] and Microsoft [2] operational networks, which indicate that bugs caused more than 30% of the documented customer impacting incidents. Moreover, the analysis of post-mortem reports of Google's B4 SDN-based network [1] shows that issues related to network control plane elements prevail. Despite the magnitude and ubiquity of the network control software failures, the state of the art literature is still missing the realistic dependability models of SDN controllers [3]–[11].

The goal of this study is to provide high fidelity models, able to reproduce the stochastic behaviour of bug manifestation and bug removal processes, for different abstraction levels of real SDN controller software. Such models are needed in order to assess and improve the design of a reliable control plane, which often neglects or oversimplifies the nature of controller failures. The models provide useful guidelines for the software developers, helping them optimize the planning of the test effort expenditures by allocating it to the most vulnerable modules, as well as to estimate the residual bug content and expected software failure rates by improving the timing of the release management. The controller in our study is *OpenDaylight*, the biggest open source SDN controller platform, whose bug repository is publicly available. However, the implications of our study are not limited to *OpenDaylight*, given that the big share of commercial controllers by major network equipment vendors, such as Cisco[2], are also based on its distributions.

We follow an approach of the related empirical studies on network incidents [1], [2], [12], [13] and complex systems sharing the similar vulnerabilities [14]–[17] to systematically analyze statistical properties of defects in SDN controller software. Our analysis includes more than 8k bugs reported in the period over five years, between March 2013 and September 2018, in development and deployment phases of *OpenDaylight*. The repository contains the issues related to core controller functions, as well as embedded applications, plug-ins and drivers towards variety of forwarding devices, enabling us to analyze trends and localize the most problematic components. We rely on the standard statistical inference techniques to estimate distributions of times to detect and resolve bugs from the empirical data.

In our previous work we proposed the stochastic models to evaluate and predict the reliability of SDN controller, as well as the interaction between controllers [3]–[6]. In this work we provide the insight into controller software vulnerabilities, which enables us to identify platform bottlenecks and show how the platform dependability evolved over time. We present the strong empirical evidence that the assumptions of the

[1]The terms "software defect" and "bug" are used interchangeably; as well as "software failure" and "bug manifestation"

[2]Cisco Open SDN Controller and Ericsson Cloud SDN

978-3-903176-15-7

previous work on dependability of SDN control plane, such as perfect coordination between replicas and assumed failure rate distributions are not met in practise. We propose more accurate stochastic models of the behaviour of bug manifestation and bug removal processes, and discuss how these can be used by network management community to improve the dependability of control plane designs. We correlate the data from issue tracker and code version system to analyse the relationship between code internals and software defect metrics.

The remainder of the paper is organized as follows. Section II provides an overview of the related work on reliability of SDN controllers, and relevant empirical studies. The architecture of *OpenDaylight* controller platform is described in Section III. In Section IV we present the quantitative analysis and stochastic models of controller software vulnerabilities. Section V summarize the main results of our analysis and discusses directions for the future work.

II. Related work

In this section we present the limitations of the related work on SDN controller reliability, highlighting the assumptions possibly contradict the behaviour of the real production grade controllers.

The first studies on the reliability of SDN control plane consider the controller as perfectly reliable, assuming only control path link failures [18]. More recent studies made different assumptions about the controller reliability [6]–[11], [19]. Some assumptions are over-simplifying the nature of failures, as they were necessary to obtain analytically tractable results, rather than reflecting controller behaviour from real life deployments or testbed measurements. The authors in [19] model controller reliability as deterministic variable. Several other studies [6], [8], [9] assumed that the controller failure to be a Poisson process, which was necessary to obtain analytical solutions of the proposed Markov models. Ros et al. [7] assume that the operational probabilities of network elements, including the controllers, follow different i.i.d. Weibull distributions. Longo et al. [10] discuss the limitations of Markovian models, and assume the reliability of the controller to follow phase-type distribution (generalized hypoexponential distribution), which captures better the changes in operational conditions. In our previous work [5] instantaneous availability of the controller software is modelled by hyperexponential distribution, which describes different failure types (i.e., transient, hang and crash). It also include temporal fluctuations of controller software failure rates, which change due to maturity and state of the controller. The software maturity model was further developed, based on data from ONOS and OpenDaylight [3], [4] bug repositories, which showed the bug manifestation rates can be described as Non-Homogeneous Poisson Process (NHPP).

Another limitation of the previous work is the assumption about the perfect failover between different controller replicas. The study by Gonzalez et al. [11] modelled the synchronization process between controller replicas, with the focus on the trade-off between the consistency and performance. Sakic et al. [6] provide a more realistic model of RAFT consensus algorithm under different failure rates, complementing it with the measurements from a *OpenDaylight* testbed. Our goal in this work is to provide an empirical evidence to support or reject the assumptions made in the previous works, focusing on the assumed distributions of software failure rates.

Previous empirical studies on network incidents by Google [1], Microsoft [12], IP Backbone [13] and data centers [20] provide a valuable data to the industry and the researchers, exposing network vulnerabilities and suggesting preventive measures. However, a comprehensive study on network control software in SDN is still missing. To fill this gap, we follow the approach of the large scale studies on open source software [17], operating systems [15], concurrency issues [16] and on cloud management systems [14] to systematically analyze the nature of controller software failures.

III. OpenDaylight Controller Platform

The *OpenDaylight* controller platform is a collaborative "community-led and industry-supported framework", foreseen from the beginning to be the Linux of the networks. Majority of the *OpenDaylight* key partners are vendors, and the focus at the beginning was on the the applications in data centers and coexistence with network virtualization technologies. The project has grown to have 3,920,556 lines of code, with 1210 developers from industry and research contributing to its code base. Eight releases, each one with several stability releases (SR), have been distributed up to date. The complex code base is organized in 95 projects[3] , ranging from core controller project to advanced embedded controller applications. Due to the space limitations, we include 55 most relevant projects covering more than 98% of the bug content. In order to grasp easily the code organization, we group the projects into 5 categories, and map them to the *OpenDaylight* architecture, as illustrated in Fig. 1a. Descriptions of the projects are adapted from the *OpenDaylight* documentation.

A. Core controller functions

This category consists in core **Controller project**, and two related projects, **topology processing (topoproc)** and **L2-switch**. As the controller project is the largest and the most important of *OpenDaylight* platform, we also present its subcomponents. The role of Service Abstraction Layer (SAL) is to decouple network application interfaces from south-bound protocol plug-ins, e.g., OpenFlow. Initial solution was API-Driven (`AD-SAL`), aiming to provide a collection of direct application interface adaptations, which evolved to a more generic Model-Driven SAL (`MD-SAL`). MD-SAL is providing the supporting functions for other projects. As the part of the controller module, the SAL is connecting the protocol plugins to the Network Function Modules[4], such as Flow Rule Manager (FRM), Topology Manager, Switch Manager, etc. Controller `clustering` enables the load sharing between a group of the controllers, as well as the fault tolerance.

[3]Adapted from OpenDaylight project list

[4]Adapted from documentation of Brocade Vyatta Controller

NETCONF is an XML-based protocol used for configuration and monitoring devices in the network. *OpenDaylight* supports the NETCONF protocol as a northbound server as well as a southbound plugin. RESTCONF allows access to MD-SAL data stores in the controller.

B. Embedded Controller Applications

OpenDaylight provides multitude of embedded applications, related to the original virtualization use case, as well as the applications related to production environment requirements, such as security, monitoring and analytics.

1) Virtualization support: **NetVirt** is a network virtualization solution that includes the support for software and hardware switches, L3VPN (BGPVPN), NAT and Floating IPs, IPv6, Security Groups, MAC and IP learning, etc. **Distributed Overlay Virtual Ethernet (DOVE)** and **VPN service** projects have been deprecated and split into different projects, mainly NetVirt. **Neutron** enables the integration with OpenStack Neutron networking service; while **Service Function Chaining (SFC)** provides ability to define and connect (”chain”) an ordered set of network functions realizing a composite service. **Virtual Tenant Network (VTN)** is an application that provides multi-tenant virtual network on an SDN controller; while **NetIDE** provides the virtualization of SDN networks where users can bring their own controllers.

2) Monitoring and analytics: **Cardinal** enables monitoring of OpenDaylight and underlying network as a service; while **Centinel** provides a framework to collect, aggregate and sink streaming data, leverage **Time Series Data Repository (TSDR)**.

3) Security: The issues related to the security applications, such as **Controller Shield**, **NAT application** and **Unified Secure Channel (USCH)**, are not reported in the *OpenDaylight* bug repository.

4) Miscellaneous: **Generic Network Interface, Utilities and Services (GENIUS)**, allows interference-free coexistence with different applications, while **Energy Management (EMAN)** implements energy measurement and control features. Other representative embedded applications are **Honeycomb** Virtual Bridge Domain (VBD) vector packet processing, **Bit Indexed Explicit Replication (BIER)** architecture for the forwarding of multicast data packets, **Atrium** open source BGP Peering Router and **Armoury** framework to request network function from workload managers.

C. Network abstractions (Policy/Intent)

Network abstractions are provided to users and applications, which can specify high level policies (intents) without minding the low level hardware-specific implementation details. **Group Based Policy (GBP)** projects allows users to express network configuration in a declarative versus imperative way. **Network Modelling (NEMO)** project aims to simplify the usage of network by providing a new intent northbound interface (NBI), enabling network users/applications to describe their demands for network resources, services and logical operations in an intuitive way. **Network Intent Composition (NIC)** project enables the controller to manage and direct network services and network resources based on describing the intent for network behaviours and network policies. **Fabric as a Service (FaaS)** project aims to create a common abstraction layer on top of a physical network, so northbound API or services can be easier to be mapped onto the physical network as concrete device configuration. **Application Layer Traffic Optimization (ALTO)** is an IETF protocol RFC 7285, provide simplified network views and services, e.g., cost maps, to applications.

D. South Bound Interface (SBI) plugins

OpenDaylight supports a variety of southbound protocols, or plugins, adapting to the different types of networks. These plugins represent the drivers for the controller to communicate with the network devices, and represent the largest part of the code base. The SBI plugins are classified into i) native to SDN OpenFlow, ii) interworking with legacy network protocols to ensure the support for hybrid networks, iii) and domain specific, such as support for wireless access points, remote radio heads, packet cable and IoT data manager and iv) security related, such as **Secure Network Bootstrapping Infrastructure (SNBI)** and **Unified Secure Channel (USC)**.

E. Supporting functions

This category comprises the projects that are implicitly related to all previous categories, such as network representation and modelling tools (**MD-SAL** and **YANG tools**); deployment related functions including the standard **Authentication, Authorization and Accounting (AAA)**, release management and integration, as well as documentation; and Graphical User Interface (GUI) **DLUX** and **NEXT**. The remaining 40 projects contribute to approximately to 2% of the bug content, and are grouped together as other supporting functions.

IV. Measuring, analyzing and modelling of defects in the SDN controller

A. Mining the bug repository

The *OpenDaylight* issue tracker relied at the very beginning on the internal mailing list and excel sheets, then used Bugzilla issue tracker in the first 6 releases, and finally migrated to Jira in October, 2017. We have analyzed 8k+ bugs, which were reported in the period over 5 years, beteween March 2013 until September 2018[5]. We were interested the issues labelled as ”bugs”, rather than enhancements and new feature requests. We include the bugs which with status ”done”, ”closed”, ”resolved”, ”verified” or ”confirmed” status, but filter out the issues with status ”open”, ”in review”, ”in progress”, and ”to do”. Each bug entry contains a label to which project it belongs, severity, date of creation and, if applicable, the resolution date and resolution status.

[5]Data retreived on 3rd of September 2018 from OpenDaylight Jira repo

POLICY/INTENT (363)

- **GBP** (275)
- NEMO (8)
- NIC (35)
- FaaS (33)
- ALTO (12)

CORE CONTROLLER (1656)

- **Controller prj. (1485)**
  - **MD-SAL** (462)
  - **AD-SAL** (218)
  - **clustering** (319)
  - config (118)
  - NETCONF(160)
  - RESTCONF(146)
  - *other ctrl.* (62)
- topoproc (85)
- L2 switch (86)

EMBEDDED CONTROLLER APPLICATIONS (2015)

Virtualization support (1765)
- **NetVirt** (1148)
- DOVE (15)
- VPN service (83)
- VTN (156)
- **SFC** (207)
- Neutron (146)
- NetIDE (10)

Monitoring and analytics (86)
- Cardinal (7)
- Centinel (30)
- TSDR (49)

Security related (N/A)
- Controller Shield
- NAT Application
- USCH

Miscellaneous (164)
- GENIUS (125)
- EMAN (4)
- Honeycomb (22)
- BIER (5)
- Atrium (3)
- Armoury (5)

SOUTH BOUND INTERFACE PLUG-INS (2852)

SDN native (1382)
- **OpenFlow** (882)
- OpenFlowJava (64)
- ***OVSDB**** (405)
- OpFlex (1)
- OF-Config (8)
- SNMP4SDN (22)

Interworking with legacy networks (1333)
- **BGP/PCEP** (571)
- LACP (20)
- ***NETCONF**** (439)
- LISP (165)
- SNMP (10)
- SXP (128)

Wireless, cable, IoT (104)
- CapWAP (9)
- OCP (11)
- PCMM/COPS(19)
- IoT-DM (65)

Security related (33)
- SNBI (27)
- USC(6)

SUPPORTING (1615)

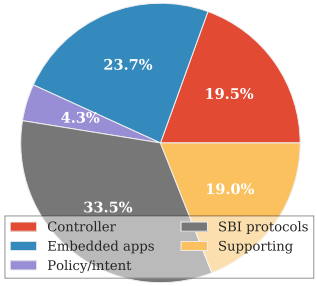
Network representation and modelling tools (828)
- **MD-SAL** (219)
- **YANG Tools** (609)

Deployment related (483)
- AAA (152)
- Integration (127)
- OdlParent (105)
- RelEng (55)
- Docs (44)

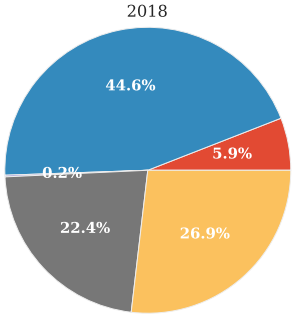
GUI (123)
- DLUX (121)
- NEXT (2)

Other supporting (181)

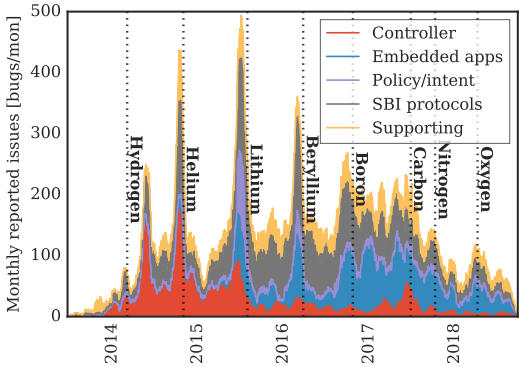
(a) Contributions of different functional blocks and individual projects to the total bug content of the *OpenDaylight* platform. The projects with more than 200 bugs are highlighted. The projects marked with (*) belong to more than one category.



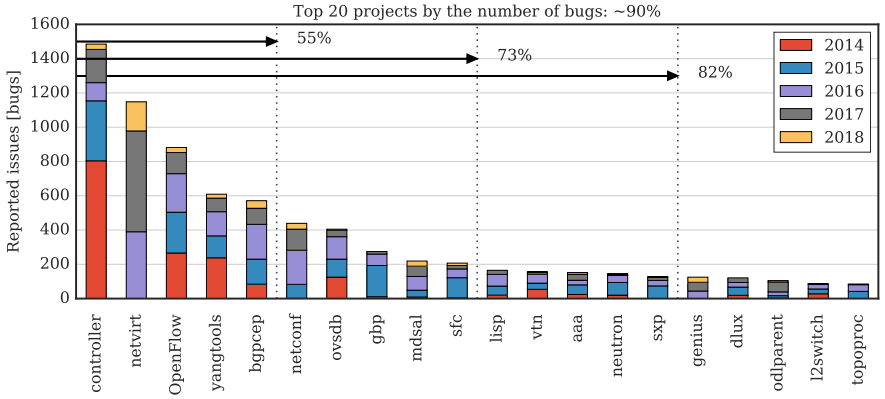
(b) Relative contributions of different functional blocks.



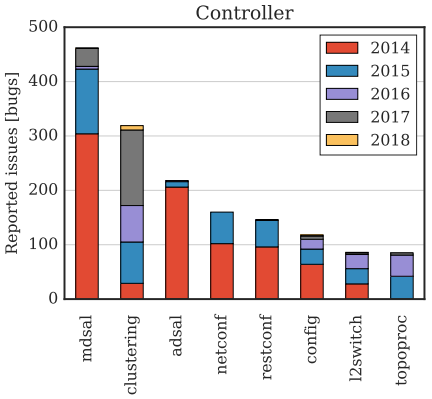
(c) Relative contributions to the bug content reported in 2018.



(d) Changes in the bug reporting rate over time.



(e) The most critical projects by the number of bugs. Top 20 projects contributing to 90% of the total bug content are presented.



(f) Distribution of the bugs in the core controller project.

Fig. 1: Quantitative assessment of vulnerabilities in the *OpenDaylight* SDN controller platform.

B. Quantitative assessment of controller vulnerabilities

The contribution of different functional categories and individual projects to the total bug content of *OpenDaylight* SDN controller platform is illustrated in Fig. 1.

The categorical split of bug content, illustrated in Fig. 1a, allows us to localize the most vulnerable components of the platform. It can be observed that the highest contributions comes from SBI plug-ins, which only underlines the difficulty of implementation of the unified network control system for such a diverse system of networks. However, the comparison of the change in relative contribution of different functional blocks in 2018, illustrated in Fig. 1b and Fig. 1c respectively, shows a clear shift of focus towards embedded applications, mainly those related to network virtualization. Moreover, three phases in can be distinguished in the evolution of the *OpenDaylight*, as illustrated in Fig. 1d, where average monthly bug reporting rate is presented. In the first phase, between Hydrogen and Lithium releases most of the reported issues were related to the core controller project. In the second phase, between Lithium and Boron releases, issues related to SBI plug-ins were predominant, while after Boron release majority of the issues can be attributed to embedded applications. Ten projects with more than 200 reported issues, which are highlighted in Fig. 1a, are evenly spread across all functional blocks.

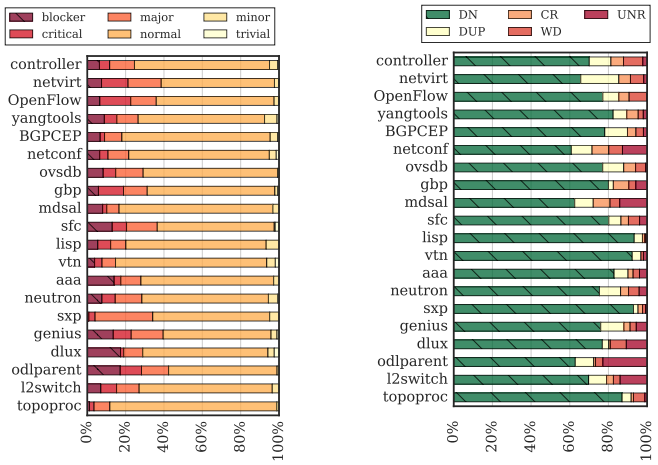
The contribution of the top 20 most critical projects is shown in Fig. 1e. These top 20 projects contribute to more than 90% of the overall number of the reported bugs. We observe that the number of bugs is unevenly distributed across the projects, with top five projects contributing to more than 50% of the overall bug content. The controller project is the most critical, according to the absolute number of reported issues, followed by NetVirt and OpenFlow projects. The issues in these subsystems will have a different impact on the network services. While a critical issue in core controller project may incapacitate the network, the issue in network virtualization subsystem will affect only the users which are deploying this feature in their production environment. A dynamics between different controller failure types and user perceived service reliability needs to be carefully studied, which we leave for the future work.

Since core controller project is the most critical component, we also show the distribution of the bugs across its modules over time in Fig. 1f. We observe that although the transitions from AD-SAL to MD-SAL primitives contributed to the majority of the reported issues, most of them were reported in the first phase of project, before 2016. The most recent issues are related to the clustering subsystem, which is especially troubling, given that virtually all real life deployments rely on distributed control plane implementation, in order to improve control plane scalability and fault tolerance. This critical issue, when the fault tolerance mechanism is in itself faulty, goes against the assumption that the simple replication of the controllers always improves the controller reliability.

C. Modelling the bug discovery and bug removal rates

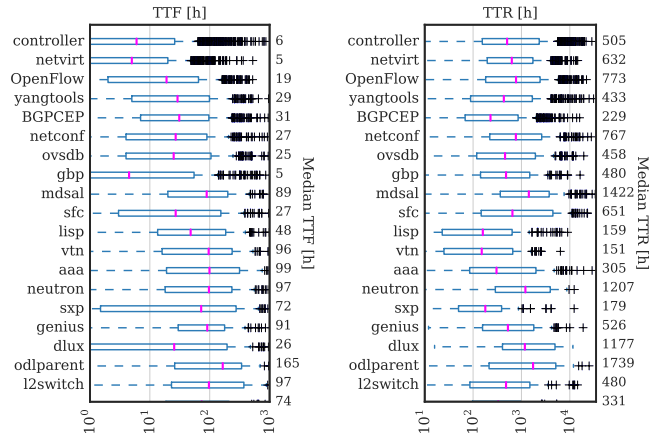
In this section, we study whether the assumptions about controller reliability and failure rates are met in the real life deployments. We study the differences between the projects, as well as the impact of bug severity on the bug manifestation and removal rates. We also analyze how their statistical properties change in over time, and provide the most suitable stochastic models to capture these dependencies. Finally, we discuss the applications of the proposed stochastic models.

Differences between the projects in terms of severity and resolution success are presented in Fig. 2. The bug severity indicates how badly it affects the system functionality, ranging from ”blocker”, i.e., where the functionality becomes unavailable because of the bug, to ”trivial” issues, which do not impact significantly the system performance. The share of severe bugs (”blocker”,”critical”, ”major”), illustrated in Fig. 2a varies between 20% to over 40% across the projects.



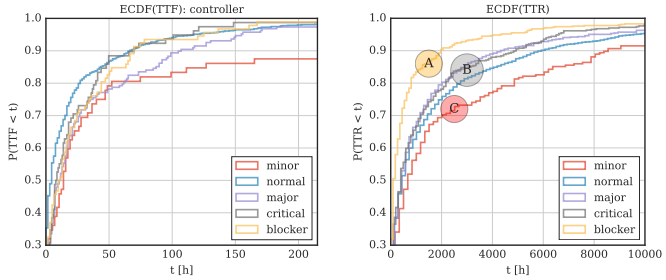
(a) Reported bug severity (b) Resolution status

Fig. 2: Differences in bug severity and bug resolution success.



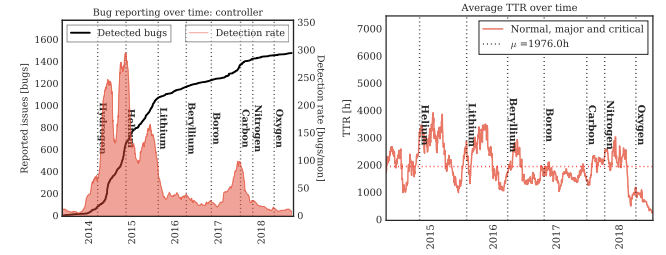
(a) Time to find a bug (TTF) (b) Time to resolve a bug (TTR)

Fig. 3: Differences in bug detection and bug removal rates.



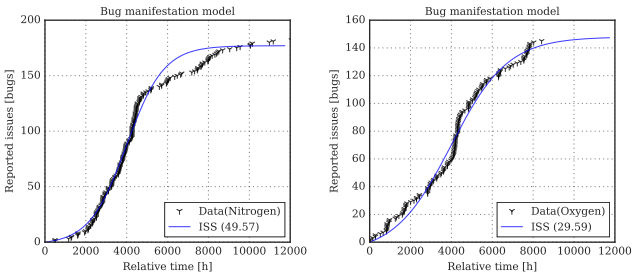
(a) TTF $= f(\text{severity})$ (b) TTR $= f(\text{severity})$

Fig. 4: Impact of bug severity on TTF and TTR distributions.



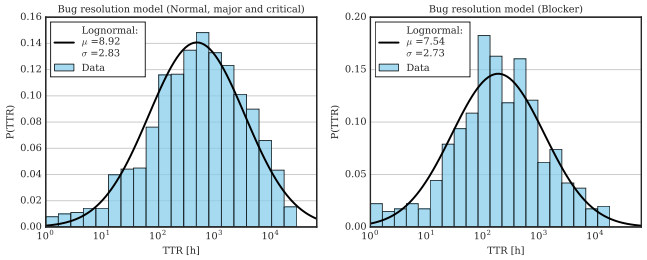
(a) $\text{TTF}^{-1} = f(t)$ (b) TTR $= f(t)$.

Fig. 5: Evolution of bug manifestation and removal rates.



(a) Nitrogen release (b) Oxygen release

Fig. 6: Modelling TTF with Ohba's Inflection S-Shaped model.



(a) Normal, major and critical (b) Blocker

Fig. 7: Modelling TTR with lognormal distribution.

The resolution status of the closed and verified issues can be "done"(DN), "duplicate" (DUP), "can't reproduce" (CR), "won't do" (WD), and "unresolved" (UNR). We consider only the issues with the "done" status to be successfully resolved. It can be observed in Fig. 2b that the resolution success varies between 68% and 75%. Differences in bug detection and bug removal rates between the projects are illustrated in Fig. 3. We observe that the median Time to Find (TTF) the next bug, i.e. the time between successive bug reports, and the median Time to Resolve (TTR), also show big variations across the projects, with order of magnitude difference. It can be observed that these distributions are all long tail distributions, indicating that the Poisson distribution would not be a good fit.

Fig. 4 shows the impact of bug severity on TTF and TTR distributions. Since TTF is relevant on the project level, we illustrate the results for the core controller project in Fig. 4a. We observe that TTR distribution is affected to a greater degree, where three clear classes can identified, illustrated in Fig. 4b.

The changes of TTF and TTR over time, are illustrated in Fig. 5. It can be observed in Fig.5a that TTF exhibits big variations over time, showing peaks shortly before official release dates. For the illustration purpose we show $\text{TTF}^{-1}(t)$ and $\text{N}_{\text{bug}}(t)$ only for the core controller project. With exception of [3], [4], none of the models in the state of the art literature captured this dynamic feature of TTF distributions. On another hand TTR showed a lower degree of variation over time, as illustrated in Fig.5b.

A class of Non-Homogeneous Poisson Process (NHPP) models can explain TTF dynamics, by assuming that the bug manifestation rate is proportional to the residual bug content in the code. Since during testing and early deployment phase (of the particular software release) the bugs are removed from the code, the bug manifestation rate changes gradually [21]. From all NHPP models we found that Ohba's Inflection S-Shaped (ISS) model shows a particularly good fit for the last two *OpenDaylight* releases, Nitrogen and Oxygen, as illustrated in Fig. 6, where Mean Square Error (MSE) of the model is indicated in parenthesis. In short, ISS models assumes that the expected failure rate over time as a three parameter (a, b, ϕ) function of time:

$$\lambda_{iss}(t) = \text{TTF}^{-1}(t) = abe^{-bt}\frac{1+\phi}{(1+\phi e^{-bt})^2} \tag{1}$$

where t is a relative time since the beginning of the integration testing. The corresponding cumulative number of reported bugs, illustrated in Fig. 6a and Fig. 6b is then:

$$\text{N}_{iss}^{\text{bugs}}(t) = \int_0^t \lambda_{iss}(\tau)d\tau = a\frac{1-e^{-bt}}{1+\phi e^{-bt}} \tag{2}$$

Distributions of TTR do not show high variation over time, but exhibit a significant impact of the bug severity. Hence we propose the models for different severity classes. Since the distributions show symmetry in logarithmic scale in Fig.7, we propose a log-normal distribution for TTR, which is often used in related empirical studies:

$$\text{P}[\text{TTR} = t] = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(\ln t-\mu)^2}{2\sigma^2}} \tag{3}$$

The comparison of the fitted model and the data is presented in Fig. 7. The models for TTF and TTR in Fig. 6 and Fig. 7 can be used to improve the accuracy of the existing models of the SDN control plane dependability [3]–[6], and assess the reliability of different control plane design solutions [7], [18].

The stochastic models can also improve release management and software development process. For instance, an official software release T_R may be postponed until the expected failure rate in all the projects has reached a desired level λ_0 [21]:

$$\begin{aligned} &\min\ T_R \\ &\text{s.t.} \max_{p \in \text{Projects}} \lambda_p(T_R) \leq \lambda_0 \end{aligned}$$

The models can be used to optimize the allocation of the test effort. Given the constrained budget T_{budget} for the test effort expenditures, the developers might choose to improve the most defective components first [22]:

$$\begin{aligned} &\max \sum_{p \in \text{Projects}} [N_p^{bugs}(t_0) - N_p^{bugs}(t_0 + t_p)] \\ &\text{s.t.} \sum_{p \in \text{Projects}} t_p \leq T_{\text{budget}} \end{aligned}$$

where t_p represents the effective time effort, e.g., in man-hours, dedicated to a particular project p. Unfortunately, in general case, this optimization problem has to be solved numerically.

D. Correlation between code internals and bug statistics

In this section the relationship between the software defect metrics and different code characteristics is assessed. The code characteristics of interest are:

- *Project size*, expressed as thousand lines of code (kLOC)
- *Project activity level*, reflected in number of commits
- *Community size*, i.e., the number of unique contributors
- *Age of the project*, i.e., the number of internal releases

The software defect metrics related to median TTR (MTTR) and debugging success rate (DS) were defined in Section IV-C. The fault density (FD) is a software reliability metrics defined as the number of bugs per source line of code. Due to the complexity of TTF models, we compare average failure rate (AFR) instead defined as average monthly bug reporting rate.

The summary of the code internals[6]and the relevant bug statistics derived from the issue tracker is presented in Table I, and pairwise relationships are illustrated in Fig. 8. We present the data for the top 20 projects according to the absolute number of bugs, which contribute to approximately 90% of total the *OpenDaylight* code and bug content. The strong correlations, i.e., R-value > 0.5, are indicated in the figure.

1) The project size: The kLOC shows positive correlation with the total bug content and average monthly failure rate (AFR), which is expected. It can be observed that with the exception of the controller project, this relationship is approx. quadratic. This is reflected in fault density (FD), defined as the number of bugs per line of code, which is between 2 and 5 for most projects. Since, the number of code lines depends on the programming language, it is worth noting that most of the *OpenDaylight* projects are written in `Java`, the exception being GUI (`JavaScript`) and VTN projects (`C++`).

2) The project activity level: The number of commits show almost linear relationship with the total bug content, R-value $= 0.93$, and a very strong correlation with AFR, R-value $= 0.78$. It can be observed that relationship between the number of reported bugs to the total number of commits is approx. $1:5$. It would be interesting to analyze the commit messages and assess whether this strong correlation stems from the fact that many commits represent the bug fixes, or the increased bug content of active projects comes from the introduction of the new features and frequent code changes.

3) The community size: The number of unique contributors also shows the strong relationship with the total number of bugs, with R-value $= 0.88$, and AFR, R-value $= 0.84$. We observe that approximately 15 bugs per unique contributor can be expected. This strong correlation project activity level and bug content could potentially be used as a good predictor of the bug content in the future projects.

4) Others: The number of internal project releases did not show strong correlation with other software metrics. However, the calendar age of the project does show a strong correlation with the total bug content, as expected.

An unexpected relationship can be observed between the total bug content and fault density (not shown), as it appears that the larger projects tend to have higher fault density. As the part of the future work we plan to systematically analyze the relationship between the project structural and functional characteristics, including multi-correlation, as well as more complex non-linear relationships.

V. Conclusions and Future Work

In this work we have analyzed the software repositories of *OpenDaylight*, the largest and most widely used production grade SDN controller platform.

We have compared the contributions of the different platform components, and localized the most vulnerable ones. We have demonstrated that the existing abstractions SDN controller do not provide an accurate picture of controller availability, since different projects and functional units have different maturity, and hence different failure rates are expected. Moreover, the failures of different functional units, e.g., controller core v.s. south bound plugins, will have different impact on the users and network applications. As a part of the future work, we plan to study the dynamics between SDN control and data plane, using the controller reliability metrics provided in this paper.

The stochastic models provided for the bug manifestation and removal processes can be used to improve the fidelity of the existing SDN control plane models, such as [5], [6]. The accuracy of our models can be further improved by automatic

[6]Data retrieved on 6th of September 2018 from OpenDaylight Git version control system, available on GitHub.

TABLE I: Summary of code internals and bug related metrics for top 20 projects in *OpenDaylight*

PROJECT	Bugs (all)	Bugs (2017)	Commits	Contributors	Releases	kLOC	FD[$\frac{\text{bugs}}{\text{kLOC}}$]	AFR [$\frac{\text{bugs}}{\text{mon}}$]	MTTR [h]	DS [%]
controller	1485	194	8247	95	59	191.693	7.75	22.76	505	70.11
netvirt	1148	588	5105	91	29	301.175	3.81	37.58	632	68.17
OpenFlow	882	123	3182	65	43	278.393	3.17	15.51	773	70.41
yangtools	609	79	4184	44	65	232.515	2.62	9.91	433	72.18
bgpcep	571	94	3101	33	47	179.892	3.17	9.46	229	72.90
netconf	439	123	2882	56	24	215.3	2.04	9.21	767	71.87
ovsdb	405	37	2590	62	38	70.365	5.76	7.53	458	72.25
gbp	275	15	1057	35	26	147.2	1.87	7.84	480	72.61
mdsal	219	60	3468	51	24	210.933	1.04	4.23	1422	72.25
sfc	207	20	1446	48	35	134.602	1.54	4.47	651	72.51
lisp	165	24	1064	26	107	54.160	3.05	3.42	159	73.05
vtn	156	13	1019	19	36	1263.904	0.12	3.15	151	73.51
aaa	152	35	975	35	30	27.99	5.43	2.9	305	73.72
neutron	146	4	921	44	29	33.247	4.39	2.83	1207	73.75
sxp	128	18	443	29	13	42.142	3.04	3.15	179	74.10
genius	125	51	1590	37	20	89.293	1.40	4.32	526	74.14
dlux	121	27	537	21	35	63.992	1.89	3.60	1177	74.18
odlparent	105	58	1039	28	43	12.905	8.14	2.08	1739	74.02
l2switch	86	3	280	27	34	14.817	5.80	2.03	480	73.97
topoproc	85	4	470	17	23	51.069	1.66	3.56	331	74.12
TOTAL	**7509 (88%)**					**3570.792 (92%)**				

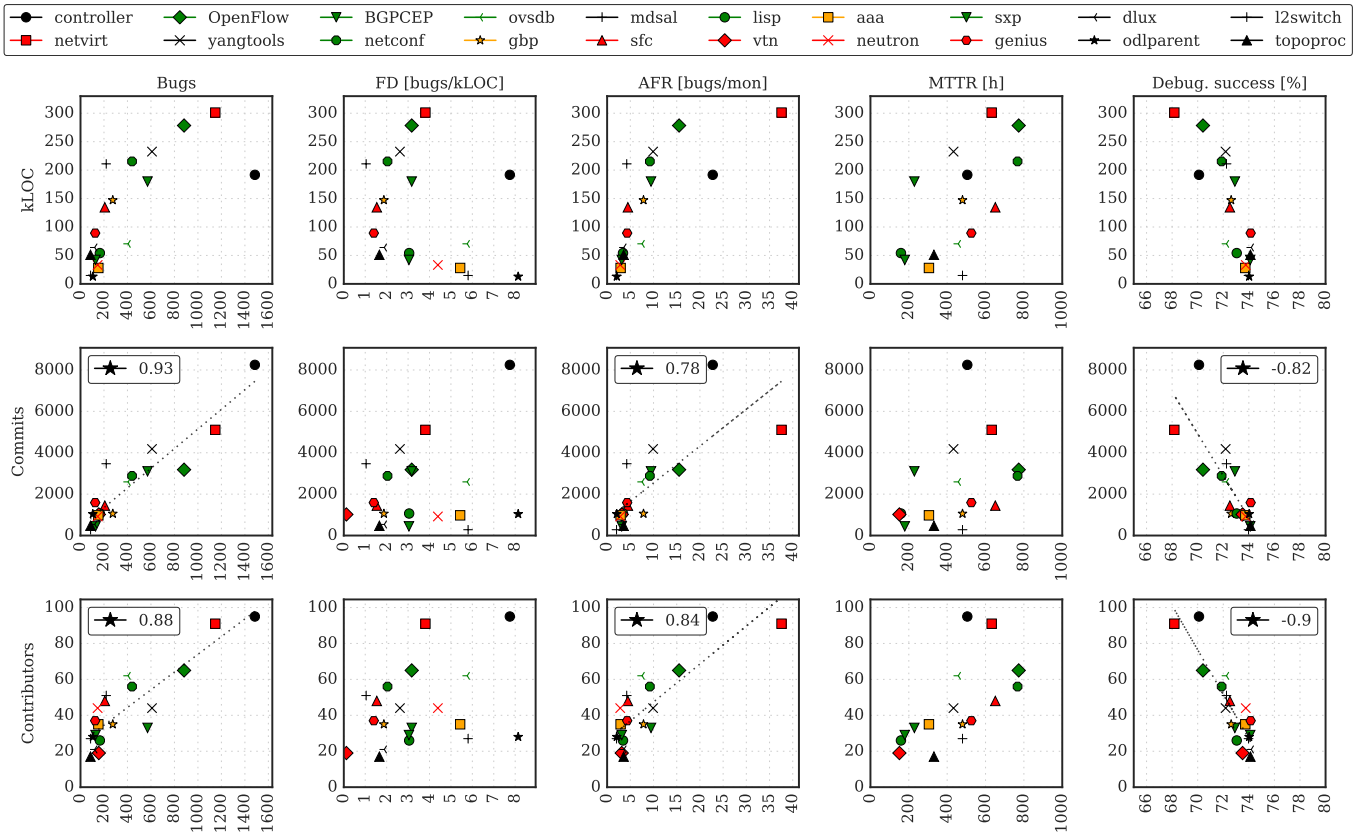


Fig. 8: Pairwise relationships between code internals and bug statistics. Strong correlations (R-value > 0.5), indicated in subfigures, can be observed between project activity level (*commits*) and project community size (*contributors*).

classification of the bugs reported in the issue tracker, e.g., by using Natural Language Processing (NLP) techniques.

We have analyzed the relationship between the project code internals and bug statistics, in attempt to explain where the differences between project come from. Detailed analysis of the correlation between the structural and functional characteristics of the defective code in SDN platforms is another interesting research direction, that we plan to pursue.

ACKNOWLEDGMENT

This work has received funding from CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1) and the German BMBF (Project ID 16KIS0473).

REFERENCES

[1] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or die: High-availability design principles drawn from Google's network infrastructure," in *Proceedings of ACM SIGCOMM Conference*. ACM, 2016, pp. 58–72.

[2] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan, "Crystalnet: Faithfully emulating large production networks," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 599–613.

[3] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, A. Blenk, W. Kellerer, and C. M. Machuca, "Assessing the maturity of sdn controllers with software reliability growth models," *IEEE Transactions on Network and Service Management*, 2018.

[4] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, W. Kellerer, and C. Mas Machuca, "An empirical study of software reliability in SDN controllers," in *13th International Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–9.

[5] P. Vizarreta, P. Heegaard, B. Helvik, W. Kellerer, and C. Mas Machuca, "Characterization of failure dynamics in SDN controllers," in *9th International Workshop on Resilient Networks Design and Modeling (RNDM)*. IEEE, 2017, pp. 1–7.

[6] E. Sakic and W. Kellerer, "Response time and availability study of raft consensus in distributed sdn control plane," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 304–318, 2018.

[7] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 31–36.

[8] T. A. Nguyen, T. Eom, S. An, J. S. Park, J. B. Hong, and D. S. Kim, "Availability modeling and analysis for software defined networks," in *Dependable Computing (PRDC), 2015 IEEE 21st Pacific Rim International Symposium on*. IEEE, 2015, pp. 159–168.

[9] G. Nencioni, B. E. Helvik, A. J. Gonzalez, P. E. Heegaard, and A. Kamisinski, "Availability modelling of software-defined backbone networks," in *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 2016, pp. 105–112.

[10] F. Longo, S. Distefano, D. Bruneo, and M. Scarpa, "Dependability modeling of software defined networking," *Computer Networks*, vol. 83, pp. 280–296, 2015.

[11] A. J. Gonzalez, G. Nencioni, B. E. Helvik, and A. Kamisinski, "A fault-tolerant and consistent sdn controller," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[12] R. Potharaju and N. Jain, "When the network crumbles: An empirical study of cloud network failures and their impact on services," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 15.

[13] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an ip backbone," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2307–2317.

[14] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin *et al.*, "What bugs live in the cloud? a study of 3000+ issues in cloud systems," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

[15] P. Anbalagan and M. Vouk, "An empirical study of security problem reports in linux distributions," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*. IEEE, 2009, pp. 481–484.

[16] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 2, pp. 329–339, 2008.

[17] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai, "Have things changed now?: an empirical study of bug characteristics in modern open source software," in *Proceedings of the 1st workshop on Architectural and system support for improving software dependability*. ACM, 2006, pp. 25–33.

[18] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.

[19] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient sdn control plane," in *Resilient Networks Design and Modeling (RNDM), 2016 8th International Workshop on*. IEEE, 2016, pp. 253–259.

[20] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 350–361.

[21] M. R. Lyu *et al.*, *Handbook of software reliability engineering*. IEEE computer society press CA, 1996.

[22] S. Osaki, *Stochastic models in reliability and maintenance*. Springer, 2002.