DASON: Dependability Assessment Framework for Imperfect Distributed SDN Implementations

Petra Vizarreta, Kishor Trivedi, Veena Mendiratta, *Senior Member, IEEE*,
Wolfgang Kellerer, *Senior Member, IEEE*, and Carmen Mas-Machuca, *Senior Member, IEEE*

***Abstract*—In Software Defined Networking (SDN), network programmability is enabled through a logically centralized control plane. Production networks deploy multiple controllers for scalability and reliability reasons, which in turn rely on distributed consensus protocols to operate in a logically centralized manner. However, bugs in distributed control plane can have disastrous effects on the data plane, e.g., losing traffic by installing paths containing blackholes. In this paper we study the prevalence of issues in state-of-the-art distributed frameworks in SDN, by analyzing 500+ issues reported in two of the largest open source SDN controller platforms: Open Network Operating System (ONOS) and OpenDaylight (ODL), during the period between 2014-2019. We identify system vulnerabilities, localize dependability bottlenecks, and provide stochastic models for a holistic assessment of system dependability.**

***Index Terms*—Software Defined Networking, SDN controller, ONOS, OpenDaylight, distributed system, distributed consensus, high availability, fault tolerance, software reliability.**

I. Introduction

A. Problem Definition and Research Challenges

In SOFTWARE Defined Networking (SDN), the control plane logic of forwarding devices is offloaded to an SDN controller, which assumes the role of a network operating system. Logically centralized network control enables fine-grained resource management, dynamic per-flow QoS control and simplified enforcement of traffic engineering policies, spanning a diverse set of network devices. Present-day production grade SDN controllers additionally provide support for 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 has resulted in the controllers becoming rather complex software systems, and recent studies [1]–[3] on large scale operational networks have reported that software bugs caused more than 30% of documented customer impacting incidents.

Production networks deploy multiple controllers to ensure scalability, high availability and high performance. In such distributed architectures, the benefits of logically centralized network control are maintained by means of distributed protocols such as Gossip and Raft [4]. However, correct and stable implementation of distributed network control plane is not trivial, as confirmed by Google's report on critical network outages [1], which showed that control plane issues prevail in their B4 WAN.[1] Their analysis showed that under *control plane software failures*, *maintaining globally consistent network state* is a difficult, and the *cascade of control-plane element failures* is a common culprit of critical customer impacting failures.

Despite the magnitude and ubiquity of network control software failures, the state of the art literature is still missing realistic dependability models of SDN controllers. The goal of this study is to provide high fidelity models that can reproduce the stochastic behaviour of real-life distributed SDN platforms. Such models are needed in order to identify dependability bottlenecks, and reliably assess whether SDN solutions are ready to be deployed in a particular use-case scenario, such as industrial networks [6]. The controllers in our study are Open Network Operating System (ONOS) [7] and OpenDaylight (ODL) [8], two of the largest production-grade open source SDN orchestration platforms, whose code internals and bug repository are publicly available, allowing us to perform an in-depth dependability assessment.

B. Towards Data-Driven Dependability Assurance

We propose DASON, a *data-driven* dependability assessment framework, for a holistic assessment of dependability, as illustrated in Fig. 1. The framework implements a general *analyse-model-evaluate* meta-workflow for dependability assessment, applied the use-cases of open-source distributed SDN orchestration platforms.

1) Mining Software Repositories: In the analysis step, the system architecture and failure modes are extracted by mining

Manuscript received April 5, 2019; revised November 21, 2019; accepted January 23, 2020. Date of publication February 14, 2020; date of current version June 10, 2020. This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets, and has received funding from the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1) and the German BMBF (Project ID 16KIS0473). Kishor Trivedi's research was supported in part by the National Natural Science Foundation of China under Grant number 61872169. The associate editor coordinating the review of this article and approving it for publication was C. Avin. *(Corresponding author: Petra Vizarreta.)*

Petra Vizarreta, Wolfgang Kellerer, and Carmen Mas-Machuca are with the Chair of Communication Networks, Technical University of Munich, 80333 Munich, Germany (e-mail: petra.vizarreta@lkn.ei.tum.de).

Kishor Trivedi is the with Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA.

Veena Mendiratta is with the Distributed Control System Research, Nokia Bell Labs, Naperville, IL 60563 USA.

Digital Object Identifier 10.1109/TNSM.2020.2973925

[1] B4 [5] Google's internal Wide Area Network (WAN), carrying the traffic between data center clusters, is arguably the biggest live SDN network, both in geographical scale and the volume of traffic it serves.

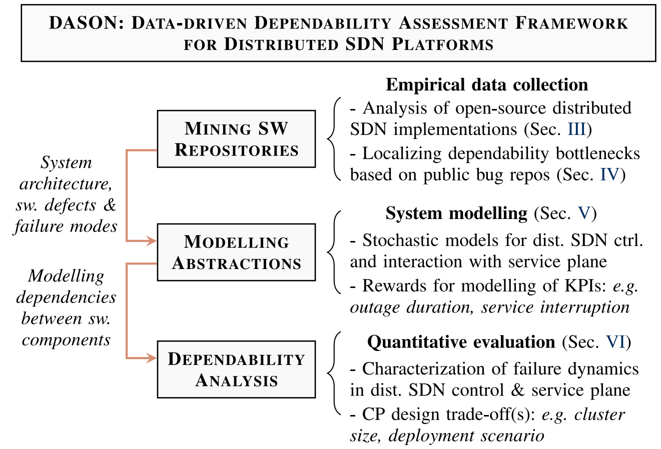

DASON: DATA-DRIVEN DEPENDABILITY ASSESSMENT FRAMEWORK FOR DISTRIBUTED SDN PLATFORMS
System architecture, sw. defects & failure modes
Modelling dependencies between sw. components
MINING SW REPOSITORIES
Empirical data collection
- Analysis of open-source distributed SDN implementations (Sec. III)
- Localizing dependability bottlenecks based on public bug repos (Sec. IV)
MODELLING ABSTRACTIONS
System modelling (Sec. V)
- Stochastic models for dist. SDN ctrl. and interaction with service plane
- Rewards for modelling of KPIs: e.g. outage duration, service interruption
DEPENDABILITY ANALYSIS
Quantitative evaluation (Sec. VI)
- Characterization of failure dynamics in dist. SDN control & service plane
- CP design trade-off(s): e.g. cluster size, deployment scenario

Fig. 1. An overview of DASON: Data-driven dependability assurance framework based on Stochastic Reward Nets (SRN).

software repositories of two distributed SDN platforms, ONOS and ODL. We leverage the fact that the code repositories and issue trackers are open to the public, enabling us to perform detailed analysis of system vulnerabilities. The outcome of this analysis, i.e., prevalent failure modes, has been used to guide the construction of stochastic models proposed in the next step.

2) Modelling Abstractions: The modelling abstractions are provided by the formalism of Stochastic Reward Nets (SRN) [9]. SRNs can be directly mapped to Markov chains, and are widely used in modelling complex systems consisting of large number of dependent components. We model separately single controller nodes, their interaction using distributed system protocols, as well as the services that run in such architectures. Dependability KPIs of interest, e.g., downtime distribution and outage frequency, are computed by assigning reward rates at the SRN level. Model input parameters are based on real-life controllers and systems.

3) Quantitative Dependability Evaluation: Once the stochastic models are defined and parametrized, they can be used as a tool for forecasting the control plane outages and dependability benchmark platform. For instance, the models can be used to characterize the failure dynamics in a distributed SDN control plane, as well as their impact on user-perceived service availability. Different control plane designs, e.g., cluster size and deployment scenarios, can be compared easily, by modifying the parameters of the stochastic models.

C. Our Contribution

The work in this article is strongly motivated by the results presented in our previous two papers [10], [11], although it does not presents their direct extension. The modelling abstractions proposed in [10] focused on independent controller failures, but has neglected the complex interaction between different replicas in a cluster replicas (assuming perfect handover, and no synchronization overhead). The empirical study in [11], which analyzed the large ODL bug repository, showed that these assumptions do not hold in practice, since the clustering component is one of the most buggy modules in ODL core subsystem. In this article we provide more accurate modelling abstractions for imperfect distributed SDN implementations. Moreover, we extend [11] with taxonomy of defects in distributed SDN control plane, as well as service-control plane dependencies. The contributions of this article can be summarized as:

i. We analyse real-life distributed SDN implementations, and localize software defects and common failure modes.

ii. We propose modelling abstractions for imperfect distributed control plane, and interaction with service plane.

iii. We characterize the failure dynamics in realistic scenarios, including not only pure control plane dependability metrics, but also user-perceived service availability.

The remainder of the paper is organized as follows. Section II provides an overview of the related work on distributed SDN controller frameworks, and key empirical and model-based studies relevant for our methodology. Section III presents an overview of distributed SDN control planes, while Section IV discusses defects of real life distributed SDN implementations. In Section V modelling abstractions based on SRN for imperfect distributed SDN plane are presented, and are used for the quantitative analysis of control plane and service availability in Section VI. Section VII concludes the paper with a summary and discussion of the results.

II. RELATED WORK

The following sections provide an overview of the related work on distributed SDN controller frameworks (Section II-A), and relevant empirical and model-based dependability studies (Section II-B).

A. High-Availability in Distributed SDN Implementations

A good overview of distributed SDN control platforms is presented in [12]. The survey compared different architectural designs and their approaches to address scalability and high-availability issues. However, most of the presented controllers have not made it into production environments, such as Onix, HyperFlow, DISCO and Kandoo [13]–[16], or are closed proprietary solutions, such as Google's B4 [5] and Espresso [2]. Hence, we choose to focus on ONOS [7] and ODL [8], two production-grade open source controllers, which form the code-basis for many other commercial vendor products.

The *software maturity* of these two platforms, in terms of the reliability growth, was compared in [17]. The further mining of the software repositories [11] identified the clustering module (distributed control plane implementation) as the culprit in many of the ODL controller failures, but did not further investigate the nature of such issues.

Stability issues under high load of distributed control plane implementation with ONOS was analyzed in [18]. The authors have shown that consensus protocols, such as Raft, misbehave in overload conditions, due to increases in the delay of heartbeat messages and time-threshold based failure detectors. Such behaviour triggers the frequent leader re-elections, leading to a crash of the entire control plane. The same effect of performance degradation under load causing a *node flapping, repeated leader elections, and a cascade of control plane failures* was also observed in [19], which noted that the problem was already reported in the bug repository. Sakic *et al.* proposed ODL control plane enhancements, such as adaptive consistency [20], [21] addressing the issue of chattyness of consensus protocols, and Byzantine Fault Tolerance (BFT) protocols [22] addressing the *security and reliability issues of misbehaving controllers*. Our mining of ONOS and ODL bug repositories, discussed in Section VII, exposed many more issues of practical distributed control plane implementations.

Two informal measurement standards on SDN control plane benchmarking, by the IETF [23] and the ONF [24], specify cluster performance and stability tests. The performance of ODL clustering, in terms of synchronization overhead, failure detection and failover time, was analyzed in [25], while ONOS inter-controller traffic in different scenarios was measured and modelled in [26]. An ONOS report on SDN control plane performance [27] discusses distributed design solutions considered by developers, as well as the final implementation, and demonstrates the improvements compared to an older release. These standard performance and cluster stability tests have already been incorporated in the ONOS and ODL test suites.

Despite the extensive testing many of the bugs go unnoticed during testing, manifesting only in the production environment. One of the reasons why many bugs escape the testing phase is *non-determinism*, such as *racing and concurrency issues*, which makes them extremely hard to reproduce, since triggering requires precise timing between input events and internal procedures [28]. In [29], [30] the authors showed a huge number of concurrency violations in SDN controller applications. In the follow-up work [31], the concurrency violations were clustered and filtered, facilitating fault localization of the root causes analysis for the developers, demonstrating its efficiency on the Floodlight controller. Indeed, our analysis of production-grade controllers showed that concurrency issues are the root cause in many of the reported issues related to distributed protocols.

Google's report on critical network outages [1], showed that control plane issues prevail in their SDN-based B4 WAN. Their analysis showed that *maintaining globally consistent network state* is a challenge, due to the *control plane convergence delays*, *inconsistency between control plane elements*, as well as *synchronization between data and control plane*. A number of partial and complete *failures of control plane elements and the control plane network*, including the *cascade of control-plane element failures* were observed. Noteworthy are also the *operational issues* due to the *buggy control plane software update push*.

Another empirical study on defects in well-known distributed systems, such as Cassandra and HDFS [32], showed that *faulty/error handling* was the cause of 95% of catastrophic failures. In most of the cases the error handling code was either empty or incomplete, ignoring the local failure which then propagated to entire system, or was overreacting, allowing a minor failure to crash the entire system. The authors also noticed *resource leaks* and *incorrect performance* issues, which have not been analyzed before in the context of SDN.

New vulnerabilities, due to cyclic dependencies between the control and data planes in distributed SDN, are discussed in [33]. The authors demonstrate how *control plane network failures* may render the cluster down, even in the absence of partitions. Illustrative examples of the problems of *oscillating leaders* and *lost leadership* were also presented. Alternative adaptive consistency models for ONOS have been discussed in [34], while relaxation of strong consistency models used in ODL was proposed in [35].

Large scale empirical studies on real-life incidents in Google and Microsoft networks [36], IP Backbone [37] and data center networks [38], [39] provide valuable data to the industry and to 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 systematically analyze two of the largest open source repositories (10k+ bugs) to locate the vulnerabilities in production-grade distributed controller platforms.

B. Model-Based Studies on SDN Control Plane Dependability

Despite the diversity and complexity of SDN control plane failures, most of the studies on SDN control plane dependability reduce the controller to a single failure mode, i.e., assuming it is either operational or non-operational. Dynamic models often assume that software failure and repair time are exponentially distributed, which is an assumption thought necessary to obtain analytically tractable results, rather than reflecting controller behaviour from real life deployments or testbed measurements.

The first studies on the reliability of SDN control plane consider the controller as perfectly reliable, assuming only control path link failures [40], and distributing the controllers only for latency reasons. More recent studies [41]–[43] also accounted for the software failures. The authors in [41] model controller availability as a deterministic variable, while in [42] the assumption was that the operational times of network elements, including the controllers, have different i.i.d. Weibull distributions. The temporal variations in software failure rates due to reliability growth are modelled as Non-Homogeneous Poisson Process (NHPP) in [17], [44]. Longo *et al.* [43] 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, when some of the controller instances fail.

More complex dependencies and interactions between the elements of a complex systems use the Stochastic Reward Nets (SRN). The models described using the SRN modelling formalism can be directly translated to large Continuous Time Markov Chains (CTMC). The SRN models for the interaction between SDN control and data plane have been proposed in [20], [45]–[48]. In our previous work [10] we proposed a dynamic controller model based on SRN. The model included five failure modes (e.g., transient and stop-fail software failures), as well as the temporal fluctuations of controller software failure rates, which change in the long term due to maturity and in the short-term due to resource leaks. However, the model did not address the interaction between controllers.

Overall, an important limitation of the previous models is the assumption about the perfect failover between identical controller replicas. Our analysis shows that simple controller replication is ineffective, because of i) shared failures, e.g., semantic bug in path computation, ii) faulty error handling mechanisms, which may lead to a erroneous failover and cause a cascade of controller failures and iii) failures specific to distributed control plane implementations, such as a software bug in distributed consensus protocols. These inefficiencies are modelled as the common mode failure (i), and the coverage factor (ii) in system dependability literature, while the failures specific to distributed systems (iii) are typically neglected.

Indeed, the complexity of interaction between SDN controller replicas has been widely overlooked in the literature. The failure correlation due to control plane misconfiguration was discussed in [46], while Mendiratta *et al.* [47] also discussed the imperfect failover. The study by Gonzalez *et al.* [48] modelled the synchronization process between controller replicas, with the focus on the trade-off between consistency and performance. Sakic and Kellerer [20] provided a realistic response time model of the Raft consensus algorithm under different failure rates, complementing it with the measurements from an ODL testbed. SRN models proposed in this paper, aim to combine all failure modes of distributed SDN implementations, for a holistic assessment of system dependability.

Model-based dependability assurance based on SRN has been successfully applied to various communication systems, such as software defined backbone network [46], NFV-based virtualized core [49], VoIP system [50], IaaS cloud [51], as well as distributed consensus protocols, such Raft [20], Paxos BFT [52], and their application in permissioned block chain systems [53]. We follow a similar approach to provide high fidelity models that account for all failure modes encountered in the issue repositories of distributed SDN platforms.

III. Analysis of Distributed SDN Implementations With ONOS and ODL

Next, we present the basic concepts of distributed systems, with the focus on distributed SDN control plane, implementations with ONOS and ODL. Our analysis is based primarily on the official code documentation and the presentations by ONOS[2] and ODL[3] distributed system engineering teams.

A. A Primer on Distributed Control Plane in SDN

In practice, a cluster of multiple SDN controllers is deployed in order to provide high performance, scalability and high availability. Appearance of the logically centralized control plane is possible due to distributed protocols, which take care of the coordination, knowledge dissemination and seamless failover between different controller replicas. Services provided by the SDN control plane to network devices and applications should be unaware of the distributed control plane implementation. Fig. 2(a) illustrates how the separation of concerns and location transparency are implemented.

In order to manage large-scale networks, network state is partitioned into smaller chunks, called *shards*. Provisioning of a fault-tolerant system requires shards to be replicated on several nodes. Shard replicas can be updated in a *strongly consistent* or *eventually consistent* manner. Distributed systems use different replication styles (Fig. 2(b)) depending on the application requirements and access patterns. i) Consensus based protocols like Raft [4] provide strong consistency, requiring the majority of the replicas to acknowledge the update before it can be committed by the leader, and used to create the response to the client. ii) Gossip protocols provide eventual consistency, using the epidemic style of propagation, where random pairs of neighbours compare their version of the data, known as *anti-entropy*, and agree on an appropriate final state if concurrent updates have occurred, which is known as *reconciliation*. iii) Another style of replication is *primary-backup*. The performance-consistency trade-off is balanced by choosing the number of replicas and replication flavour.

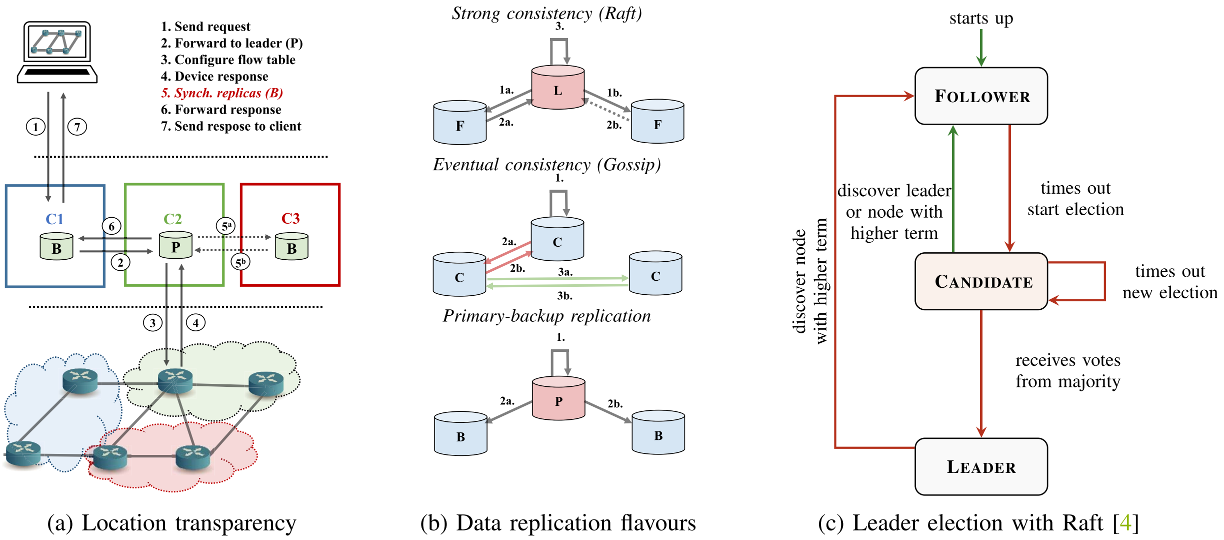
Timestamps and version vectors are used for ordering of the events. Distributed systems are inherently asynchronous and typically there is no global clock. Local clocks skew and drift, and even the NTP protocol can provide a limited accuracy. *Event ordering* is necessary to enforce causal relationships between the events. Hence, the vector clocks, also called *version vectors*, are often used instead. With Raft, the leader is responsible for the correct ordering of the updates.

Cluster membership and role management with Raft is illustrated in Fig. 2(c). The controller (re-)joining the cluster starts as the *follower*. If the leader heartbeat is not received within a given threshold, it becomes a candidate, increases the election *term*, votes for itself and requests the votes from other members. Three outcomes are possible: if the candidate receives the majority of the votes before the *election timeout* it becomes the leader; if it discovers a candidate with higher term or a current leader it becomes the follower; otherwise, it increases the term count and starts the new election round. If a leader discovers a node with a higher term, it gives up the leadership. Raft uses randomized election timeout to reduce the probability of split votes. The leader election service can be used without using Raft's strongly consistent data replication, e.g., for the assignment of the primary-backup roles.

After leader failure, the follower with the largest term and the longest log will win the election. The leader proves its liveliness by sending periodic *heartbeats* to its followers. However, the independent controller nodes communicate over an unreliable network, which typically does not provide bounded delay guarantees, and it is practically impossible to distinguish between network and controller node failures. The messages can be delayed (network congestion or high load on the controller node), or lost (partitioned network or node

[2] Thomas Vachushka: ONOS Distributed Core and Jordan Halterman: Distributed Systems in ONOS with Atomix 3: Architecture and Implementation.

[3] Colin Dixon: Clustering in OpenDaylight, Robert Varga, Jan Medved: OpenDaylight Clustering: What's new in Boron, Moiz Raja, Tom Pantelis: MD-SAL Clustering Internals.



(a) Location transparency (b) Data replication flavours (c) Leader election with Raft [4]

Fig. 2. A primer on distributed SDN control plane implementations with ONOS and ODL.

crash), which can result in temporary inconsistencies between the network state seen by replicas. Failure detection is based on time thresholds, which have to be carefully tuned balancing the trade-off between stability and failure detection efficiency.

The *φ-accrual failure detector* [54], is widely used in distributed systems, including in the SDN controllers implementations addressed here. The detector accounts for a suspicion level, $\Phi = -\log_{10}(1 - F(t))$, where $F(t)$ represents a distribution of previous heartbeat inter-arrival times, implicitly assuming a normal distribution. Raft requires the majority of the controllers to be available, hence, it requires *$2f + 1$* controllers to tolerate f failures. In the case when network partitioning split the cluster into two parts that cannot communicate, either both partitions continue operating independently (fav. availability) or one of the partition freezes (fav. consistency), as consequence of the CAP theorem [55].

After the crash, a node re-joining the cluster has to synchronize with the rest of the cluster. The changes to the data store are kept in a *log*, or a *journal*. *Log compaction*, or *state compression*, is the process of removing the entries from the log that no longer affect the current state. It is performed periodically to prevent the uncontrollable growth of the log. The replicas may request logs from another replicas after order to fill in missing transactions. *Snapshots* of data store state are saved, as a checkpoint in case the node crashes. Journals and snapshots are stored on disk for persistence.

The implementation of distributed systems requires fine-tuning of configuration parameters. The controller nodes have finite resources, such as CPU and memory, which can easily exhaust if not dimensioned and managed properly. The nodes may slow down during high load, or computationally expensive operations, such as serialization of large messages. Distributed systems rely on 3rd-party libraries, which may introduce interoperability issues.

B. ONOS Implementation

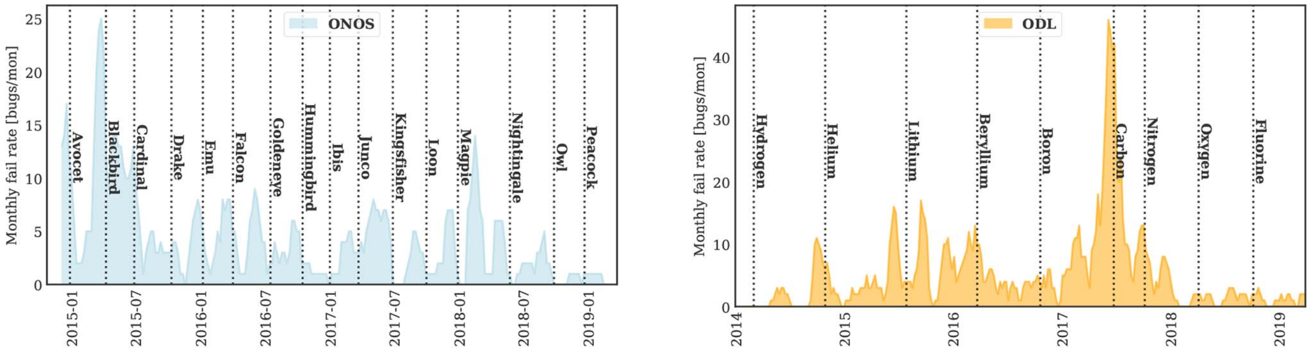
The focus of ONOS, since its inception has been on providing scalability, high availability and carrier-grade performance fulfilling the requirements of large operator networks. The project is supported by the key partners from the telecom operators and network equipment vendors. Distributed core was introduced from the beginning, and has evolved together with the application ecosystem.

ONOS core provides low level distributed primitives, such as `EventuallyConsistentMap` and `ConsistentMap`, offering different consistency models and replication styles. Distributed primitives provide interfaces similar to standard Java classes, implementing the data structures and synchronization operations upon which data stores are built. Developer guidelines suggest that control plane data, such as resource reservation and other network configuration data, use strong consistency. Data originating from the environment, such as network topology (read-intensive), should use eventual consistency to provide faster reaction to the network events. A primary-backup replication is used for the partitioned `FlowRuleStore`, while device mastership uses `LeadershipService`. Journals and snapshots are stored on disk for persistence.

C. ODL Implementation

ODL is a much larger and older project, foreseen from the beginning to be the Linux of the networks, supporting a variety of southbound protocols to ensure the smooth transition from legacy networks. The majority of ODL key partners are vendors, and the focus at the beginning was on the applications in data centers and coexistence with network virtualization technologies. Development of major clustering project features started only after the fifth release (Boron).

ODL provides essentially two data stores, *configurational* to store a desired state, e.g., configuration of the flows, and *operational* store, storing the actual network state. All data is stored in the data tree, which is broken into shards. There are *module-based shards*, e.g., inventory, topology, while the rest of data goes to default shard. The shards are replicated to followers for high availability. Data replication uses the Raft consensus protocol, providing only a strong consistency model



(a) Open Networking Systems (ONOS)

(b) OpenDaylight (ODL)

Fig. 3. The number of software defects related to distributed implementations reported over time for ONOS and ODL. The dates of major releases for both distributed controller platforms are indicated in the figure.

for all network primitives. The `EntityOwnership` service takes care of the leader election, handles failover, as well as co-locating tasks and data. Data change notifications and Remote Procedure Calls (RPCs) operating on a given shard are directed to the entity owner, i.e., the leader.

In ODL, the Akka [56] framework encapsulates the complexity of the distributed protocols. *Akka actors* implement the data tree shards, so interacting with the remote data shard is done by sending the messages to actors. *Akka clustering* implements Raft, and is responsible for the discovery of the nodes, their IP address, as well as the liveliness and reachability of the member. Cluster messaging relies on *Akka remoting*, while *Akka persistence* is responsible for durability.

IV. Localizing Dependability Bottlenecks in Distributed SDN Implementations

Next, we provide the insights into defects reported in different functional areas of distributed control plane. We localize the most vulnerable components, as well as identify prevalent failure modes and their manifestation patterns.

Problems in ONOS and ODL controllers are reported in their public Jira issue trackers.[4] Such bug repositories are a valuable source of information, as they contain the detailed fault reports from test and production environments. In our analysis we consider the issues labelled as “bugs” rather than new feature requests or enhancements. We filter the issues related to defects in distributed implementations. In the case of ODL we select the issues tagged as part of the clustering project, while in the case of ONOS we use manual inspection.

The number of bugs over time for both controllers is presented in Fig. 3. The monthly failure rate for ONOS peaks right before Blackbird (2$^{\text{nd}}$ release), while in the case of ODL the number of defects peaks before Carbon (6$^{\text{th}}$ release), which is consistent with these controllers’ evolution.

In total 500+ issues related to the distributed implementation were reported. We divide these issues into the following four categories: i) defects in the implementation of distributed protocols (DP), ii) scalability and performance (SP), iii) high availability (HA), and iv) operational (OP) issues. In case of ambiguity, we assign a bug to the primary trigger, a necessary condition, which serves as a precursor for the manifestation of a bug as a user perceived failure.

A. Defects in the Implementation of Distributed Protocols (DP)

In a multi-controller architecture, all controllers must have a consistent view of the network state in order to provide correct logically centralized operation, which is ensured by means of distributed protocols, Raft and Gossip. We identify 216 (40%) issues in this category, related to state inconsistency, leader election process, and cluster messaging implementations.

1) State Inconsistency: State inconsistency between control plane and data plane elements has already been identified as one of the leading causes of critical outages in operational SDN networks [1]. Our analysis affirms this finding, discovering 52 (10%) of defects reported in this category.

One reason of the state inconsistency are the *missing data change notifications*. The notifications are missing for a particular data stores [dp1, dp2], update event types [dp3, dp4] and occasionally under particular conditions, e.g., master handover and load balancing [dp5, dp6].

The second root cause of the state inconsistency are *cluster synchronization issues*. It was noted that in the relaxed consistency mode it is possible to be out-of-sync indefinitely [dp7], while [dp8] reports that the node re-joining as follower could not synchronize, and the lagging follower must be forced by the leader to install a snapshot. Another common synchronization issue are *event ordering* problems. This happened, e.g., when last applied index in Raft state machine moves backwards, leading to violation of transaction ordering [dp9,dp10]. For instance, in [dp10] time moves backwards due to the Daylight Saving Time, suggesting that instead of calendar, the vector time should be used for versioning.

2) Leader Election Issues: Leadership assignment and hand-off are essential for load balancing, scale-in/out and failure mitigation operations. Our findings show that a stable leader/master is hard to implement, given that 58 (10%) issues were reported in this category.

`EntityOwnership` least load policy not working as expected [dp11], or not balancing the load properly [dp12],

[4] Data retrieved on March 3, 2019 from ONOS and ODL bug repositories.

have been reported. Sometimes, the controller role change messages are not being delivered and the devices intermittently loose their master [dp12,dp13,dp14].

3) Cluster Messaging System: Another challenge lies in the implementation of the reliable cluster messaging system, which relies on 3rd-party data serialization (Kryo) and messaging (Netty) libraries. Serialization is an expensive operation which can significantly slow down the controller, degrading the performance and eventually leading to the crash of other operations. BGP router crashing during Kryo serialization was reported in [dp15], while in [dp16] the processing of a large message incorrectly triggered `UnreachableMember`.

B. Scalability and Performance (SP) Issues

Increasing the scalability of the control plane should not affect the system performance [27], which should remain stable over the long hours of operation. However, we identified 91 (17%) of issues belonging to this category.

1) Scalability Issues: Providing a performant SDN control plane is non-trivial for large service provider networks, which induces high load on the controllers, both, in terms of topology size and the volume of network events they must handle.

A recurring issue in both controllers is seen when processing a large number of events slows down a node, *delaying the heartbeats*. Several issues related to the unexpected `UnreachableMember` when the cluster is under load have been reported, under umbrella bug [sp1]. Delayed heartbeats have severe consequences on the cluster operation, leading to frequent leader re-election, control plane instability and eventual crash, as was discussed in [18] and [19]. Indeed, Raft requirements for the correct leader election and stable operation requires the following constraints to be satisfied [4]:

$$\mathit{BroadcastTime} \ll \texttt{electionTimeout} \ll \mathit{MTBF}$$

where *BroadcastTime* represents the time to send and receive responses from to all the cluster nodes in parallel (including network propagation delay and node processing time), *MTBF* is the mean time between failures of a single server and `electionTimeout` (Fig. 2(c)).

Slow controller under load can also cause other operations to timeout and misbehave, e.g., installation of large number of flows [sp2], load balancing on large topologies [sp3, sp4], or loose data during scale-out operation [sp5].

2) Performance Regression: Maintaining the same performance at scale is presumably an even harder challenge, due to the overhead introduced by distributed protocols (e.g., leader election, consensus-based replication), as well as the resource leaks which can degrade the performance over time.

Several issues related to *performance degradation* in multinode setup have been reported. For example maximum number of installed intents being lower in the cluster than standalone mode, resource reservation taking more time and higher reaction time to network events [sp6, sp7]. The performance overhead in a cluster setup when using strongly consistent Raft replication style was discussed in [20], and in [21] the authors proposed adaptive consistency models to balance response time and reliability.

Resource leaks, such as unclosed transactions and memory leaks, can cause the performance to degrade over time and lead to the controller crash, due to the resource exhaustion. For example a bug in Atomix log compaction timer [sp8] caused the nodes to eventually run out of disk space. Moreover, a number of memory leaks have been reported, in particular data stores [sp9, sp10, sp11], as well as 3rd-party libraries, such as Netty messaging manager [sp12], and Kryo serialization [sp13]. The increase in resource consumption does not happen only due to the bugs. For instance, expired flows remain in the ODL *configurational* data store [sp14], while in ONOS `EventuallyConsistentMap` naturally grows due to the usage of placeholders replacing dead objects [sp15].

C. High Availability (HA) Issues

HA is a key enabler for mission critical operations, and in many use cases the main reason to adopt a distributed SDN design. The principles to ensure HA are reliable failure detection, failure contention, and fast recovery. Nevertheless, our analysis exposes 118 (21%) defects in HA subsystem.

1) Failure Detection: Failure detection in ONOS and ODL is based on the φ-accrual failure detector [54], which detects when the heartbeat intervals have exceeded given suspicion level. The parameters of the failure detector should be carefully tuned to avoid false positives, triggering unnecessary leader re-elections. Previously, the `heartbeatInterval` and `phiFailureThreshold` could not be configured [ha1]. In addition to false positives caused by slow-performing nodes, some configuration changes [ha2, ha3] can lead to unnecessary state changes.

2) Failure Mitigation: When failures occur, it is important that the nodes fail gracefully, recover quickly and synchronize with the rest of the cluster.

Failure contention is not trivial, due to a tight interaction of the cluster members. Failure contention mechanisms that should be in place to avoid that a failure of one instance propagating entire cluster can be faulty [ha4, ha5, ha6, ha7]. In the last example, a Raft client continually retried a failed operation as long as it could maintain its session, increasing CPU/memory usage in already overloaded partitions, causing the cluster to spiral out of control.

Failing fast and gracefully is another desirable property of highly-available systems, especially given that many other subsystems have transitive dependency to this module [ha9]. Sometimes the nodes hang in a non-recoverable state, instead of crashing hard [ha8]. In case of failures, the leadership handover should happen quickly and without a data loss [ha10, ha11, ha12, ha13].

An Efficient Recovery After Failures: Snapshots of data stores and transaction journals are occasionally persisted for durability, to ensure quick recovery after failures. The state persistence is not perfect, as reported in [ha14, ha15], for flow and intent stores in ONOS. Nevertheless, the most prevalent issue is faulty recovery, with 53 (10%) reported bugs. Typical issues are a node failing to join and sync with the rest of the cluster [ha16, ha17], and a data loss upon restoration [ha18, ha19], leading to the state inconsistency between controller replicas.

D. Operational (OP) Issues

Operational issues include supporting functions, not necessary related to the buggy controller code, but rather to practical deployment scenarios. This category includes 76 (14%) issues, related to documentation and test automation/coverage, cluster configuration and bootstrapping, interworking with virtualization platforms, upgrades and updates of the 3rd-party libraries.

1) Documentation and Testing: An adequate documentation should be provided to facilitate correct usage and configuration of the multi-node cluster [op1]. The execution of the test suites should be automated [op2], and occasionally extended with the new test cases [op3, op4], covering new failure modes, which were previously unaccounted for.

2) Cluster Configuration and Bootstrapping: The controllers in a cluster have to be correctly configured and able to automatically discover the peers. The issues, such as serialization of cluster configuration change, cluster configuration issues [op5, op6], are typically discovered before the deployment.

3) Deployment & Orchestration Issues: The controller software requires a host operating system, and the multi-instance setup is often deployed in virtualization environment, with dockers or virtual machines. The interactions with virtualization layers have to be carefully tested [op7, op8, op9].

4) Upgrades & Updates: The regression tests must be in place to efficiently detect 3rd-party vulnerabilities and backward compatibility issues, for 3rd-party libraries, as well as internal modules [op10].

E. Prevalent Failure Modes

The presented categories of defects significantly differ in their impact on the network services. Most of the bugs in the implementation of distributed protocols lead to soft failures, e.g., transient state inconsistencies [1], while memory leaks, slowing down the controller node are more likely to lead to a hard crash [26], leading to a delay of the heartbeats, which consequently may trigger a fatal cascade of control plane failures [1], [19]. Faulty failure contention mechanisms are the most critical, while they allow a single instance failure to propagate to entire cluster. In the following section we propose the modelling abstractions that capture these effects and can replicate all failure modes in imperfect SDN controller platforms.

V. Modelling Abstractions for Imperfect Distributed SDN Implementations

The modelling abstractions for imperfect distributed SDN implementations are provided in the formalism of Stochastic Reward Nets (SRN), a stochastic extension of Petri Nets [57]. We explain the key SRN modelling ideas via the examples of cluster (Fig. 4), service (Fig. 5) and rejuvenation (Fig. 6) models. In the SRN framework, the combination of markings in the *places* (circles) represents model states. The system state is changed upon the firing of the *activities*, which can be instantaneous ($\texttt{inconsistent\_state} \rightarrow \texttt{sw\_ok}$), deterministic ($\texttt{sw\_prone} \rightarrow \texttt{planned\_restart}$), or follow an n.e.d distribution. An SRN model can be automatically translated to equivalent Continuous Time Markov Chains (CTMC). he states and activities are associated with the corresponding rewards, which allows straightforward evaluation of system performance metrics, such as the expected number of operational controllers.

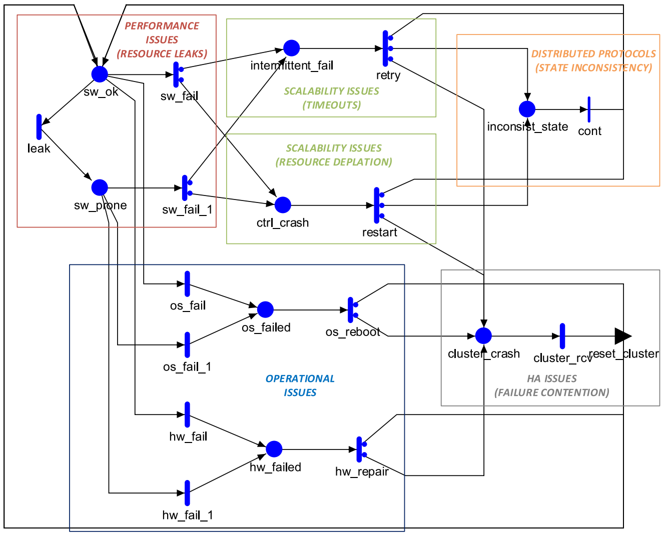


Fig. 4. Modelling abstraction for imperfect SDN cluster.

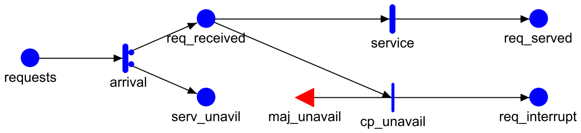


Fig. 5. SRN for service request dynamics.

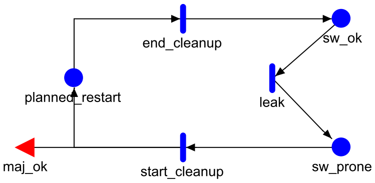


Fig. 6. SRN extension for preventive maintenance.

A. Modelling Abstraction for Imperfect SDN Cluster

Next, we elaborate how the failure patterns discussed in Section IV, are incorporated in the proposed SRN model in Fig. 4.

1) Resource Leaks: When the controller is initiated or reloaded after a crash, it starts from the clean state $\texttt{sw\_ok}$. The baseline software failure rate in this state is λ_0. During the continuous operation the resource leaks are accumulated and the controller performance starts to degrade. A common way to model this effect is to assume that the risk of failure significantly increases after a certain utilization threshold is exceeded [58], as seen in practice [sp15]. For instance, [19] note that the controller throughput and response time degrade significantly when available memory is below 4 GB. We model this effect by introducing the state sw_prone. The time to reach this utilization threshold depends on the controller load and the type of network applications serves. We account for the randomness in the resource leaks by modelling it as Poisson process with the rate λ_{leak}.

2) Soft and Hard Software Failures: Our analysis in Section IV showed two distinct types of failures, soft failures, resolved by a simple retry of the operation, and hard failures leading to a controller crash, requiring a restart. *Soft failures* are short interruptions in the controller operation, due to failed or timed-out transactions, concurrency and data race issues, leader movement and load balancing. They are typically resolved by retrying the operation, which occurs at the rate μ_{retry}. *Hard failures*, i.e., software crash, can happen when the controller node runs out of resources, e.g., with out-of-memory error. The controller restart time (rate) $\mu_{restart}$ accounts not only for the application restart, but also for the loading time of all dependent bundles, as well as the time to reconnect with peers, and discover the leaders of data store shards.

A distribution between soft and hard failures are $p^{ok}_{hard} = 1 - p^{ok}_{soft}$ in the initial state (sw_ok) and $p^{ok}_{hard} = 1 - p^{ok}_{soft}$ in failure prone state (sw_prone).

3) Transient State Inconsistencies and Cluster Crash: A qualitative analysis in the previous section showed that software failures can be either i) successfully recovered from, or can result in ii) transient state inconsistencies, or iii) cluster-wide failure. *Transient state inconsistencies* (inconsist_state) occur with probability p_{state}, due to missing notifications, wrong event ordering, lagging followers, data loss in journals and snapshots, which are used upon recovery. *Cluster failures* reflect the cases when the failure contention fails (prob. p_{crash}) and crash of a single controller brings down the entire control plane. The cluster repair rate $\mu_{cluster}$ is longer than the restart of the single controller node.

Distribution between successful repairs (p_{ok}), transient state inconsistencies (p_{state}) and cluster crash (p_{crash}) are hard to estimate from the bug reports. Hence, we make a reasonable assumption, and conduct the sensitivity analysis to evaluate the impact of uncertainty regarding its value.

4) Operational Failures: From all operational issues discussed in Section IV-D the interaction with the environment, e.g., host operating system and computing hardware will have the highest impact on the services in the production environment. The operating system, including the virtualization layer, fails with the rate λ_{os}, and is rebooted with the rate μ_{os}. Similarly, computing hardware fails with the rate λ_{hw}, and is repaired with the rate μ_{hw}. Bugs in the failure contention mechanism may lead to a complete system crash, which happens in p^{os}_{crash} and p^{hw}_{crash} of the cases, respectively.

We introduce a common failure mode to account for different deployment scenarios. In cases when controller replicas are deployed as virtual machines (VM) on the same server, the crash of a server will render all the replicas down. Similarly, in case when the replicas run in docker containers (DC), the host operating system is shared as well, and its failure will lead to a cluster-wide failure. This effect is modelled by adding reset_cluster output gate following the transitions {os,hw}_fail (see Fig. 12).

B. Reference Stand-Alone Model

A controller operating in a stand-alone mode is used as a reference model, to evaluate the gains in terms of control plane dependability in a distributed setup. In stand-alone mode many of the failure modes will be shared, since the controller uses the same data structures as in the cluster mode. State inconsistencies between control and data plane can still occur, due to the slow node failing to process the network events on time, or faulty journal recovery upon restart. Resource leaks, especially those related to the natural increase in memory usage, do occur as well in the stand-alone operation. Hence, we reuse the model in Fig. 4, but remove cluster_crash place, and all the transitions associated with it ($p_{crash} \rightarrow 0$).

C. Modelling Abstraction for Control Plane Services

SDN controllers provide services, such as management of the forwarding devices through the south-bound interface, and implementation of the high level policies through the north-bound interface. The generic modelling abstractions for control plane services is illustrated in Fig. 5. A given number of requests $N^{\text{req}}_{\text{sent}}$ arrives with a given rate λ_{req}, and are served at the rate μ_{req}. In cases when the majority of the controllers is down, new requests cannot be processed ($N^{\text{req}}_{\text{unavail.}}$), and the ongoing requests will be interrupted ($N^{\text{req}}_{\text{interrupt.}}$). λ_{req} and μ_{req} depend on a particular service, as well as the performance of the particular control plane configuration. The serving rate can be tied to the number of operational controllers and current simulation time, accounting for a degraded performance due to resource leaks. For the simplicity, we keep μ_{req} constant throughout the experiment, leaving it to the future work to study more complex parameter relationships.

D. Preventive Maintenance Policies

The failure rate after long hours of operation (sw_prone) is higher due to the lower amount of available resources, caused by resource leaks, i.e., *software ageing* [58]. An operator can decide to preventively restart the controller, cleaning up the internal data structures, dead objects, zombie processes, and unclosed connections. Such preventive measure can be implemented by starting a timer (deterministic action with rate λ_R), once the certain utilization threshold is reached. Duration of the planned outage (1/μ_R), also called *software rejuvenation*, of the controller depends on the level of rejuvenation, e.g., process or application restart. We assume rejuvenation is triggered only when majority of controllers is available.

E. Dependability Metrics of Interest

Dependability metrics are defined by assigning rewards.

1) Availability: steady state availability (SSA) is evaluated as the probability of being in the operational states: sw_ok and sw_prone. Note that the place related to state inconsistency (sw_state) is a transient state.

Depending on the replication style, the control plane will need the majority of the nodes participating in the cluster to be available (Raft), at least two operational controllers for primary-backup replication, or at least one operational replica for Gossip style replication. The model does not differentiate leader and followers, since different nodes may be leaders for different data shards. We define availability metrics as:

$$A_i = \begin{cases} A_{1/\mathrm{N}} = P\{N_{\text{operational}} \geq 1\} & (Gossip) \\ A_{2/\mathrm{N}} = P\{N_{\text{operational}} \geq 2\} & (P-B) \\ A_{\mathtt{maj.}} = P\left\{N_{\text{operational}} > \left\lfloor \frac{\mathrm{N}}{2} \right\rfloor\right\} & (Raft) \end{cases} \qquad (1)$$

where the number of operational controllers is defined as:

$$N_{\text{operational}} = \mathtt{Tokens(sw\_ok)} + \mathtt{Tokens(sw\_prone)}\,.$$

2) Failure Dynamics: We are interested in time spent in individual failure states (rate reward), in order to quantify the contribution of different failure modes to control plane outages, as well as the frequency (impulse rewards) of different controller and system failures.

3) User-Perceived Service Availability: The control plane services are not needed continuously. Depending on the service, the control plane availability will be sampled at different times, i.e., at request arrival, and for a different duration, i.e., request serving. We define *Service Availability* (*SA*), *Service Continuity* (*SC*) and *Request Completion Success Rate* (*SR*):

$$SA = \frac{N^{\text{req}}_{\text{received}}}{N^{\text{req}}_{\text{sent}}} = \frac{N^{\text{req}}_{\text{received}}}{N^{\text{req}}_{\text{received}} + N^{\text{serv.}}_{\text{unavail.}}} \qquad (2)$$

$$SC = \frac{N^{\text{req}}_{\text{served}}}{N^{\text{req}}_{\text{received}}} = \frac{N^{\text{req}}_{\text{served}}}{N^{\text{req}}_{\text{served}} + N^{\text{req}}_{\text{interrupt.}}} \qquad (3)$$

$$SR = \frac{N^{\text{req}}_{\text{served}}}{N^{\text{req}}_{\text{sent}}} = \frac{N^{\text{req}}_{\text{served}}}{N^{\text{req}}_{\text{received}} + N^{\text{serv.}}_{\text{unavail.}}} = SA \times SC \qquad (4)$$

VI. Characterization of Failure Dynamics and User-Perceived Service Availability

Next, we present the case study on realistic SDN controller platforms. The proposed models are used to quantify control plane dependability metrics. Moreover, we show the practical applications for network operators, by analysing different deployment scenarios and preventive maintenance policies.

Model parameters are based on empirical data presented in Section IV, and on the studies of software components of a similar complexity. Parameters related to the software failure rates [45], [59], resource leaks [62], and recovery procedures [59], [61], as well as the parameters related to the availability of operating system and computing hardware [45], [63], are presented in Tab. I.

A. Control Plane Availability

TABLE I
SRN Model Parameters [45], [59]–[62]

Parameter	Description	Baseline value
λ_0^{-1}	Baseline failure rate (sw_ok)	7 days
λ_{high}^{-1}	High failure rate (sw_prone)	3 days
λ_{leak}^{-1}	Resource leak rate	1 day
μ_{retry}^{-1}	Retry operation upon timeout	5 sec
$\mu_{restart}^{-1}$	Application process restart	3 min
$\mu_{cluster}^{-1}$	Restarting cluster nodes	10 min
p_{soft}^{ok}	Proportion of soft failures (sw_ok)	0.75
p_{soft}^{prone}	Prop. of soft failures (sw_prone)	0.25
$p_{hard}^{ok,prone}$	Prop. of hard failures	$1-p_{soft}^{ok,prone}$
p_{ok}	Probability of successful recovery	$1-p_{st.}-p_{cr.}$
p_{state}	Prob. of transient state inconsistency	0.40
p_{crash}	Prob. of cluster-wide crash	0.05
λ_{os}^{-1}	Mean time between OS failures	60 days
μ_{os}^{-1}	OS reboot time	30 min
λ_{hw}^{-1}	Mean time between HW failures	6 months
μ_{hw}^{-1}	HW replace time	2 hours

1) Steady-State Availability: SSA for different cluster configurations is presented in Tab. II. Since the availability for larger clusters are rather small, we present the Steady State Unavailability (SSU), to illustrate the magnitude of difference between various cluster configurations. We observe that unavailability of stand-alone controller is an order of magnitude higher than in distributed setup (N>1), because of better fault tolerance to the failures of single controller instance. However, as the number of controllers in the cluster increases, the unavailability of the cluster actually slightly decreases. This effect is in part due to specific, cluster-induced, failures, such as the ones due to faulty failure contention. Other reason why the unavailability of strongly-consistent application is lower in larger clusters, is due to the fact that larger number of the cluster members (i.e., majority) is required to be operational.

2) Parameter Uncertainty: sensitivity analysis for $A_{2/3}$ is conducted to study the impact model parameters uncertainty.

We observe in Fig. 7 that cluster recovery failures $\mu_{cluster}$ and failure contention success rate $1-p_{crash}$ have the largest impact on availability of strongly consistent services $A_{2/3}$. The qualitative analysis in the previous section exposed many defects in failure contention mechanism. The results of the sensitivity analysis only emphasize the need to prioritize the hardening of failure contention mechanism.

The following parameters, by the impact of their uncertainty of the strongly consistent services are failure rate in failure-prone state λ_{high} and resource leak rate λ_{leak}. Unfortunately, these parameters depend on many factors, such as workload, service request type, hardware configuration, available resources (CPU, RAM, etc.), and hence, have to be measured for a particular distributed setup and use case.

The uncertainty of software λ_0, operating system λ_{os} and hardware λ_{hw} failure rates has slightly lower impact. Fortunately, these parameters are well reported in the past empirical and model-based studies.

B. Failure Dynamics

The failure modes differ significantly in terms of their frequency and control plane outage, which we define here as an event in which the majority of the controllers were unavailable.

In total 10.25 control plane outages, of cumulative duration of 2.8 hours are expected within one year of operation of

TABLE II
STEADY STATE UNAVAILABILITY (1 − SSA)

Unavailability	N=1	N=3	N=5	N=7
1-$A_{1/N}$	1.2176e-03	3.1460e-04	5.1909e-04	7.197700e-04
1-$A_{2/N}$	1.0	3.1899e-04	5.1909e-04	7.197700e-04
1-$A_{maj.}$	-	3.1899e-04	5.1911e-04	7.197701e-04

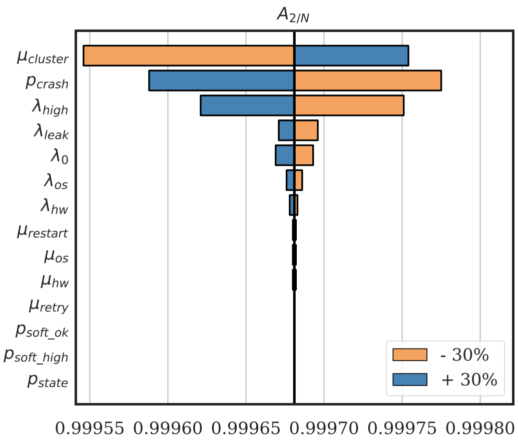


Fig. 7. Sensitivity analysis for $A_{2/3}$.

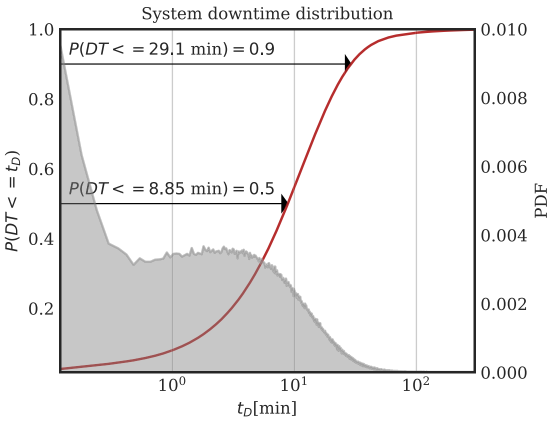


Fig. 8. Downtime (DT) distribution.

3-node cluster. In 9.88 (96%) of the cases the control plane is caused by unsuccessful failure contention. Moreover, the state inconsistency between control and data plane elements is expected to occur 73.91 times within one year. Although transient state inconsistencies are resolved quickly, not affecting the control plane availability, they can have adverse effect on data plane operation. State inconsistency can cause traffic loss by installing the paths with blackholes, or overload the links by installing the paths with loops, and install flow rules implementing conflicting policies. Practical experience reports on operational SDN networks demonstrate that state consistency issues cannot be neglected [1].

Downtime distribution (ECDF) is presented in Fig. 8. As a reference, PDF is also presented (shaded grey area). We observe that the median of system outage in a reference setup of 3-node cluster duration is below 10 min, while 90%-tile is below 30 min, with many short-term interruptions due software failures. Such failure dynamics of the control plane failures has a detrimental impact on which services get affected by the system outages.

TABLE III
SERVICE REQUEST AND SERVING RATES

Service	Service A	Service B	Service C	Service D
λ_{req}	5 min	5 min	1 h	1 h
μ_{req}	500 ms	5 sec	5 sec	5 min
SA	99.9790%	99.9807%	99.9802%	99.9814%
SC	99.9999%	99.9998%	99.9999%	99.9890%
SR	99.9790%	99.9805%	99.9801%	99.9704%

C. User-Perceived Service Availability

User-perceived availability depends on the service dynamics, i.e., request arrival (λ_{req}) and serving rates (μ_{req}). The impact of λ_{req} and μ_{req} on service availability metrics, *SA*, *SC* and *SR*, is presented in Tab. III and Fig. 9. In Tab. III several typical services are presented. Request serving rate ranges from 500 ms for an installation of large batch of flows, up to 15 min for in-service software upgrades (ISSU). Request arrival rate varies between 1 min to 1 hour, representing different control plane traffic patterns, e.g., PACKET_IN or network statistics poll. We observe that $SR = SA \times SC$ is mainly affected by *SA*, service unavailability at the moment of request arrival, more than service continuity *SC*, which is an order of magnitude higher in a given setup.

Fig. 9, illustrating unsuccessful service request completion rate ($1 - SR$), demonstrates how the longer serving rate can increase the service unavailability up to an order of magnitude. Similarly, higher request arrival rate, resulting in frequent sampling of the control plane availability, results in lower user-perceived service availability, as it is more likely to be affected by short, but frequent software failures.

D. Comparison of Different Deployment Scenarios

In small resource-constrained networks, such as industrial networks [6], the network operator may choose to run the cluster of controller nodes on shared physical machines. Deploying the controllers in separate virtual machines (VM) provides better isolation between software instances, but introduces additional overhead, since every instance runs its own operating system. Docker containers (DC) provide a lightweight virtualization, but imply an additional common mode failure, since a crash of the operating system will render all instances unavailable.

Control plane availability $A_{2/N}$ for different deployment scenarios is illustrated in Fig. 10. We observe that in the case of VMs running on the same physical machine, i.e., shared hardware failures, the availability is only slightly lower. In the case of DC, the availability loss is much higher, being an order of magnitude lower than in the first two deployment scenarios.

E. Optimization of the Preventive Maintenance Policies

The impact of different rejuvenation policies, i.e., rejuvenation scheduling λ_R, for different rejuvenation duration is

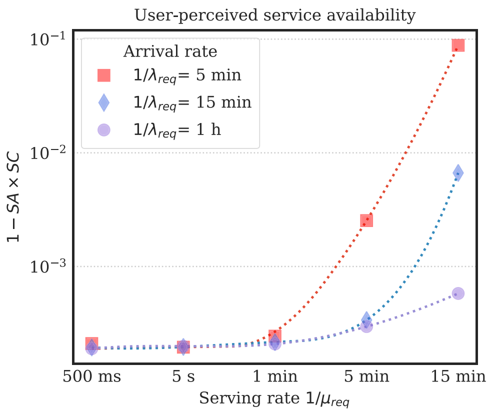


Fig. 9. User-perceived service unavailability.

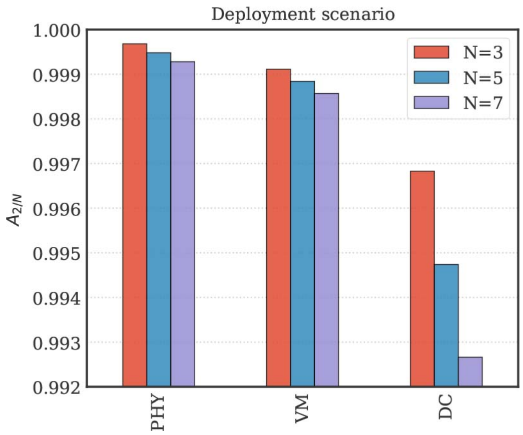


Fig. 10. $A_{2/N}$ in different deployment scenarios: separate physical machines (PHY), virtual machines (VM) and docker container (DC) sharing the same physical hardware.

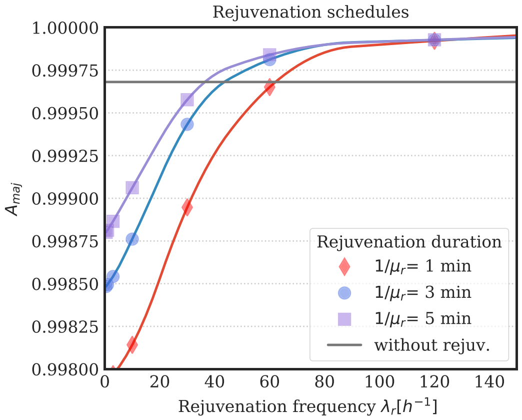
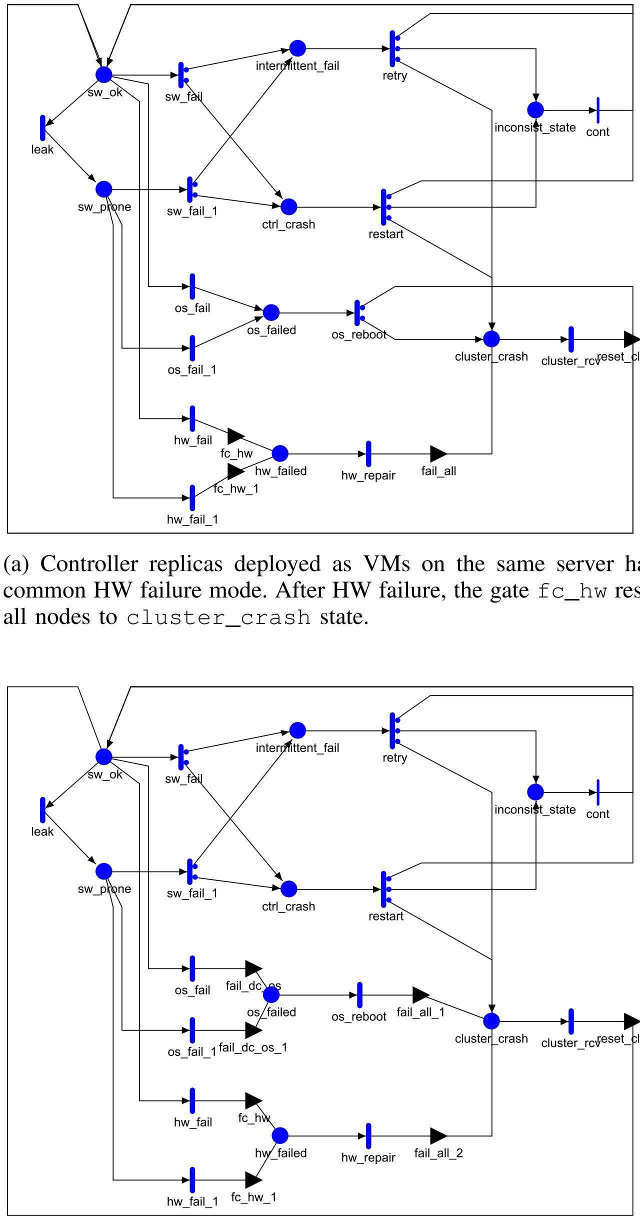


Fig. 11. Software rejuvenation policies.



(a) Controller replicas deployed as VMs on the same server have common HW failure mode. After HW failure, the gate `fc_hw` resets all nodes to `cluster_crash` state.

(b) Controller replicas deployed as DCs on the same server additionally have common OS failure mode. After OS failure, the gate `fail_dc` resets all nodes to `cluster_crash` state.

Fig. 12. Modelling abstractions for different deployment scenarios. In cases when controller replicas are deployed as virtual machines (VM) on the same server, the crash of a server will render all the replicas down. Similarly, in case when the replicas run in docker containers (DC), the host operating system is shared as well, and its failure will lead to a cluster-wide failure.

illustrated in Fig. 11. We observe that early rejuvenation is beneficial in all studied scenarios, and optimally is done as soon as the `sw_prone` state is entered ($\lambda_R \rightarrow \infty$). The operator and/or controller software developers should determine the precise threshold when this state is reached, by measuring the resource leak rates (λ_{leak}) for a given configuration setup and operational workload profiles. We leave it to the future work to conduct an exhaustive measurement campaign for real-life distributed SDN controller platforms.

VII. Concluding Remarks

This article presents a comprehensive analysis of defects and vulnerabilities in real-life SDN controller platforms, as well as the modelling abstractions of imperfect distributed controller plane. In the first part, we demonstrate that while some of the defects have been already studied, e.g., stability under overload and overhead of Raft-based synchronization, there are

TABLE IV
DEFECTS IN THE IMPLEMENTATION OF DISTRIBUTED CONTROL PLANE IN OPEN SOURCE SDN CONTROLLERS: ONOS AND ODL. THE BUG IDS IN THE FIRST COLUMN ARE HYPERSENSITIVE, CONTAINING THE HYPERLINK TO THE BUGS IN THE PUBLIC ISSUE TRACKERS

Bug ID (article)	Bug ID (repository)	Short description from bug repository
dp1	ONOS-7705	Only master of a device sees correct flow count
dp2	ONOS-7726	Links disappear after balancing masters
dp3	ONOS-7623	Device events are not replicated to other instances
dp4	ONOS-2121	Some ConsistentMap update operations do not publish events
dp5	ONOS-1883	Links disappear when devices change master
dp6	ONOS-436	Hosts learned via Gossip sometimes are missing ips
dp7	ONOS-4423	In releaxedConsistencyMode it is possible for a ConsistentMap instance to be out of sync indefinitely
dp8	CONTROLLER-1630	Follower not sync'ing up after rejoining cluster
dp9	CONTROLLER-1755	RaftActor lastApplied index moves backwards
dp10	CONTROLLER-1580	sal-remoterpc-connector: do not use calendar time for Bucket versions
dp11	CONTROLLER-1735	Entityownership leastload policy doesn't work normally
dp12	CONTROLLER-1717	RequestTimeoutException due to "Failed to transfer leadership" after become-prefix-leader with RoleChangeNotification not delivered
dp13	ONOS-4515	Cluster Device Role States out of Sync
dp14	ONOS-4529	intermittently OVS1.3 device lost mastership
dp15	ONOS-1400	BGPRouter crashes while kryo serialization due to recent change of distributed group store
dp16	CONTROLLER-1572	ReadDataReply Message was too large can result in "Received UnreachableMember" in cluster
sp1	CONTROLLER-1703	Tweak Akka and Java timeouts to a reasonable compromise between stability and failure detection
sp2	ONOS-356	Timeout in OpenFlowProvider when Intstalling large number of Intents
sp3	ONOS-2106	with a 625-sw topo, balance master does not balance well
sp4	ONOS-581	Chordal ring topology does not converge on ONOS until ONOS restart
sp5	ONOS-4785	Potential data loss during cluster scaling
sp6	ONOS-4567	Max number of intents install is less with cluster than standalone
sp7	ONOS-5279	Resource reservation takes too long in multi node cluster
sp8	ONOS-7024	Atomix 2.x timeouts
sp9	ONOS-6859	ResourceStore opens new Raft session on each transaction
sp10	ONOS-7382	Memory leak in ECFlowRuleStore
sp11	ONOS-6205	Memory leaks in DistributedMeterStore
sp12	ONOS-7412	Memory leaks in NettyMessagingManager
sp13	ONOS-3531	GossipApplicationStore throws StackOverflowError
sp14	OPNFLWPLUG-962	Multiple "expired" flows take up the memory resource of CONFIG DS which leads to Controller shutdown.
sp15	ONOS-4212	Memory leak problem when running CHO test
ha1	ONOS-6149	Not able to configure heartbeatInterval and phiFailureThreshold properties in DistributedClusterStore
ha2	ONOS-7754	Configuration change causes false positives in failure detectors
ha3	ONOS-7755	False positives in failure detection when applying initial cluster configuration
ha4	ONOS-6682	Cluster becomes unavailable after a node becomes unavailable
ha5	ONOS-5992	ONOS HA cluster failure
ha6	ONOS-5347	ONOS cluster not able to recover after killing one of cluster member
ha7	ONOS-7528	Limit memory/CPU usage when Raft partitions are overloaded
ha8	ONOS-3423	When ONOS gets an out of memory exception it essentially becomes a zombie
ha9	ONOS-1673	Fail fast when DatabaseManager does not start up cleanly
ha10	ONOS-7586	ONOS leadership change does not occurs sometimes.
ha11	CONTROLLER-1693	UnreachableMember during remove-shard-replica prevents new leader to get elected
ha12	ONOS-1883	Links disappear when devices change master
ha13	CONTROLLER-1491	Entity Ownership Service: support graceful state handoff
ha14	ONOS-6042	Flows are not getting persisted after enabling the "persistenceEnabled" flag
ha15	ONOS-5690	Intent Persistence can't be enabled in ONOS
ha16	CONTROLLER-1794	Controller fails to join cluster
ha17	CONTROLLER-1630	Follower not sync'ing up after rejoining cluster
ha18	ONOS-1965	Deadlock can occur when a old candidate restarts and does not re-enter ledership race
ha19	ONOS-2015	Some devices have no ports after ONOS cluster restart
op1	CONTROLLER-1385	Make manual-down the default for akka-cluster
op2	CONTROLLER-1581	Clustering: Maintain a script to generate default akka configuration for multinode CSIT tests.
op3	CONTROLLER-1420	Clustering: Add a count field to stress-test RPC in car yang model
op4	CONTROLLER-779	Add test-case to check Install Snapshot functionality is handled correctly
op5	ONOS-7213	New cluster configuration cannot be serialized to JSON on configuration change
op6	ONOS-3453	Bundles not loaded in all nodes in a cluster
op7	ONOS-6647	Cluster formation using docker + kubernetes
op8	ONOS-7219	Single node ONOS from Docker image can't read cluster metadata
op9	ONOS-6401	ONOS nodes timeout when trying to connect to the cluster in vm test cluster
op10	ONOS-7436	Port latency and switch latency up/down went up dramatically after atomix 2.0.14

many more critical defects that have been overlooked, e.g., resource leaks and failure contention. In the second part, we provide modelling abstractions accounting for all the failure modes identified during our qualitative analysis. Dependability models, in the formalism of SRN, are used to evaluate different dependability metrics, such as steady state availability, failure dynamics, as well as the impact on the user-perceived service availability. Moreover, we demonstrate how an SRN

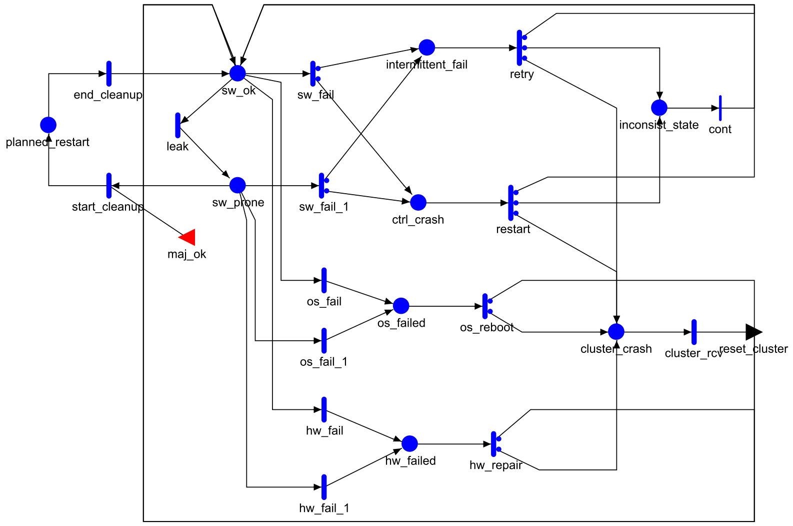


Fig. 13. Modelling abstractions for software rejuvenation policies. An operator can decide to preventively restart the controller, preventing the effects of software ageing, e.g., leaks leading to resource exhaustion [58]. One such preventive measure is implemented by starting a timer once the certain utilization threshold is reached (sw_prone state). Controller node is rejuvenated, i.e., taken out of service (planned_restart state) after the timer has expired (start_cleanup → $1/\lambda_R$). We assume rejuvenation (end_cleanup → $1/\mu_R$) is triggered only when majority of controllers is available (maj_ok).

model can assist the operators and network architects to compare different deployment scenarios and optimize preventive maintenance policies. The main threats to validity in our study is the accuracy of model parameters. While majority of the model parameters are based on the empirical data, and reported values in similar studies, few parameters are based on reasonable assumptions. Hence, we focus on the methodology and model structure, rather than numerical results. Nevertheless, we believe that the quantitative analysis give a reasonable estimation of the order of magnitude of the impact of different failure modes. We hope that the proposed framework is only the first step towards robust, data-driven, model-based certification of softwarized networks.

The proposed DASON framework can aid the operators to assess and improve the network dependability in several ways. First, the analysis in Section IV can help the developers to *design more reliable distributed network control software*. A taxonomy and localization of defects in distributed SDN platforms should raise awareness about potential vulnerabilities in network control software. Such analysis should guide the software development process and design of test suites, in order to prevent recurrence of the same defects in future releases and facilitate their early detection/mitigation (before releasing it to operational networks). Second, the models presented in Section V offer a valuable *tool for forecasting the control plane outages and dependability benchmark platform*. We have provide high fidelity stochastic models, based on a data collected from the real-life bug reports. The modelling abstractions can replicate all failure modes in imperfect SDN controller platforms, from transient state inconsistencies to failures of the operational environment. The model parameters can be easily tuned based on the control plane configuration (e.g., cluster size, virtualization flavor), and measurements (e.g., recovery times in a particular setup). This offers the operators a statistical benchmark platform to compare different "what-if scenarios". Third, DASON can be applied to *quantify user satisfaction*. The proposed modelling abstractions capture the interplay between network control plane and services offered to users and application. These compound models allow us to quantify user-perceived service quality, in terms of KPIs, such as service accessibility, continuity and probability of successful request completion. Up to the best of our knowledge, this is the first work that quantifies the impact of SDN control plane dependability to service quality.

Moreover, the identified vulnerabilities and modelling abstractions are applicable to other commercial platforms based on ONOS and ODL, such as Cisco Open SDN Controller and Ericsson Cloud SDN.

Appendix A
Dataset From Bug Repositories

This section presents the mapping of *aliases for software defects* (used in Section IV) to their respective keys used in Jira repositories are presented in (Tab. IV). The bug IDs used in this article (first column) are mapped to the real bug IDs that can be found in actual bug repositories (second column). Short bug descriptions from the repositories (third column) are also included for the convenience.

Appendix B
Modelling Abstractions

In Section V only baseline clustering model was presented in detail, due to space limitations. In this section we include the modelling abstractions for different deployment scenarios (Fig. 12), and preventive rejuvenation policies (Fig. 13).

Acknowledgment

The authors would like to thank L. Jagadeesan from Nokia Bell Labs, whose thorough review and feedback greatly improved the quality of this manuscript.

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 *Proc. ACM SIGCOMM*, 2016, pp. 58–72.

[2] K.-K. Yap *et al.*, "Taking the edge off with espresso: Scale, reliability and programmability for global Internet peering," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 432–445.

[3] H. H. Liu *et al.*, "CrystalNet: Faithfully emulating large production networks," in *Proc. ACM Symp. Oper. Syst. Principles*, 2017, pp. 599–613.

[4] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 305–319.

[5] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[6] P. Vizarreta *et al.*, "Incentives for a softwarization of wind park communication networks," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 138–144, May 2019.

[7] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6.

[8] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE Int. Symp. World Wireless Mobile Multimedia Netw.*, 2014, pp. 1–6.

[9] C. Hirel, B. Tuffin, and K. S. Trivedi, "SPNP: Stochastic Petri nets. Version 6.0," in *Proc. Int. Conf. Model. Techn. Tools Comput. Perform. Eval.*, 2000, pp. 354–357.

[10] P. Vizarreta, P. Heegaard, B. Helvik, W. Kellerer, and C. M. Machuca, "Characterization of failure dynamics in SDN controllers," in *Proc. IEEE Int. Workshop Resilient Netw. Design Model.*, 2017, pp. 1–7.

[11] P. Vizarreta, E. Sakic, W. Kellerer, and C. M. Machuca, "Mining software repositories for predictive modelling of defects in SDN controller," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag.*, 2018, pp. 1–9.

[12] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 1st Quart., 2017.

[13] T. Koponen *et al.*, "ONIX: A distributed control platform for large-scale production networks," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, vol. 10, 2010, pp. 1–6.

[14] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manag. Conf. Res. Enterprise Netw.*, 2010, p. 3.

[15] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multi-domain SDN controllers," in *Proc. IEEE Netw. Oper. Manag. Symp.*, 2014, pp. 1–4.

[16] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. ACM 1st Workshop Hot Topics SDN*, 2012, pp. 19–24.

[17] P. Vizarreta *et al.*, "Assessing the maturity of SDN controllers with software reliability growth models," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 3, pp. 1090–1104, Sep. 2018.

[18] R. Hanmer, L. Jagadeesan, V. B. Mendiratta, and H. Zhang, "Friend or foe: Strong consistency vs. overload in high-availability distributed systems and SDN," in *Proc. IEEE Int. Symp. Softw. Rel. Eng.*, 2018, pp. 1–6.

[19] C. Di Martino, U. Giordano, N. Mohanasamy, S. Russo, and M. Thottan, "In production performance testing of SDN control plane for telecom operators," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2018, pp. 642–653.

[20] E. Sakic and W. Kellerer, "Response time and availability study of RAFT consensus in distributed SDN control plane," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 304–318, Mar. 2018.

[21] E. Sakic and W. Kellerer, "Impact of adaptive consistency on distributed SDN applications: An empirical study," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2702–2715, Dec. 2018.

[22] E. Sakic, N. Derić, and W. Kellerer, "MORPH: An adaptive framework for efficient and Byzantine fault-tolerant SDN control plane," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2158–2174, Oct. 2018.

[23] V. Bhuvaneswaran, A. Basil, M. Tassinari, V. Manral, and S. Banks, "Benchmarking methodology for software-defined networking (SDN) controller performance," Internet Eng. Task Force, Fremont, CA, USA, Rep. RFC-8456, Oct. 2018. [Online]. Available: https://goo.gl/xqpDEE

[24] "OpenFlow controller benchmarking methodologies," Open Netw. Found., Menlo Park, CA, USA, Rep. ONF TR-539, Nov. 2016. [Online]. Available: https://goo.gl/HuursP

[25] D. Suh, S. Jang, S. Han, S. Pack, T. Kim, and J. Kwak, "On performance of OpenDaylight clustering," in *Proc. IEEE NetSoft Conf. Workshops*, 2016, pp. 407–410.

[26] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier, "Inter-controller traffic to support consistency in ONOS clusters," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 1018–1031, Dec. 2017.

[27] "SDN control plane performance," Menlo Park, CA, USA, ONOS, White Paper, pp. 1–23, 2017. [Online]. Available: http://onosproject.org/wp-content/uploads/2017/08/ONOS_Performance_White_Paper-2.pdf

[28] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: A comprehensive study on real world concurrency bug characteristics," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 2, pp. 329–339, 2008.

[29] J. Miserez, P. Bielik, A. El-Hassany, L. Vanbever, and M. Vechev, "SDNRacer: Detecting concurrency violations in software-defined networks," in *Proc. ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, p. 22.

[30] A. El-Hassany, J. Miserez, P. Bielik, L. Vanbever, and M. Vechev, "SDNRacer: Concurrency analysis for software-defined networks," in *Proc. ACM SIGPLAN Notices*, vol. 51, 2016, pp. 402–415.

[31] R. May, A. El-Hassany, L. Vanbever, and M. Vechev, "BigBug: Practical concurrency analysis for SDN," in *Proc. ACM Symp. SDN Res.*, 2017, pp. 88–94.

[32] D. Yuan *et al.*, "Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, 2014, pp. 249–265.

[33] Y. Zhang, E. Ramadan, H. Mekky, and Z.-L. Zhang, "When RAFT meets SDN: How to elect a leader and reach consensus in an unruly network," in *Proc. 1st Asia–Pac. Workshop Netw.*, 2017, pp. 1–7.

[34] F. Bannour, S. Souihi, and A. Mellouk, "Adaptive state consistency for distributed ONOS controllers," in *Proc. Int. Conf. Global Commun.*, 2018, pp. 1–6.

[35] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: Simplifying distributed SDN control planes," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2017, pp. 329–345.

[36] R. Potharaju and N. Jain, "When the network crumbles: An empirical study of cloud network failures and their impact on services," in *Proc. ACM Symp. Cloud Comput.*, 2013, p. 15.

[37] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *Proc. IEEE INFOCOM*, vol. 4, 2004, pp. 2307–2317.

[38] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 350–361, 2011.

[39] H. S. Gunawi *et al.*, "What bugs live in the cloud? A study of 3000+ issues in cloud systems," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–14.

[40] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. ACM 1st Workshop Hot Topics SDN*, 2012, pp. 7–12.

[41] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *Proc. IEEE Int. Workshop Resilient Netw. Design Model.*, 2016, pp. 253–259.

[42] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proc. ACM 3rd Workshop Hot Topics SDN*, 2014, pp. 31–36.

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

[44] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, W. Kellerer, and C. M. Machuca, "An empirical study of software reliability in SDN controllers," in *Proc. IEEE Int. Conf. Netw. Service Manag.*, 2017, pp. 1–9.

[45] G. Nencioni, B. E. Helvik, A. J. Gonzalez, P. E. Heegaard, and A. Kamisinski, "Availability modeling of software-defined backbone networks," in *Proc. IEEE Int. Conf. Depend. Syst. Netw. Workshop*, 2016, pp. 105–112.

[46] G. Nencioni, B. E. Helvik, and P. E. Heegaard, "Including failure correlation in availability modeling of a software-defined backbone network," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 1032–1045, Dec. 2017.

[47] V. B. Mendiratta, L. J. Jagadeesan, R. Hanmer, and M. R. Rahman, "How reliable is my software-defined network? Models and failure impacts," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2018, pp. 83–88.

[48] A. J. Gonzalez, G. Nencioni, B. E. Helvik, and A. Kamisinski, "A fault-tolerant and consistent SDN controller," in *Proc. IEEE Global Commun. Conf.*, 2016, pp. 1–6.

[49] A. Gonzalez, P. Gronsund, K. Mahmood, B. Helvik, P. Heegaard, and G. Nencioni, "Service availability in the NFV virtualized evolved packet core," in *Proc. IEEE Global Commun. Conf.*, 2015, pp. 1–6.

[50] D. Wang and K. S. Trivedi, "Modeling user-perceived service availability," in *Proc. Int. Service Availability Symp.*, 2005, pp. 107–122.

[51] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K. S. Trivedi, "Scalable analytics for IaaS cloud availability," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 57–70, Jan.–Mar. 2014.

[52] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric)," in *Proc. IEEE Symp. Rel. Distrib. Syst.*, 2017, pp. 253–255.

[53] H. Sukhwani, N. Wang, K. S. Trivedi, and A. Rindos, "Performance modeling of hyperledger fabric (permissioned blockchain network)," in *Proc. IEEE Int. Symp. Netw. Comput. Appl.*, 2018, pp. 1–8.

[54] N. Hayashibara, D. Xavier, R. Yared, and T. Katayama, "The φ accrual failure detector," in *Proc. IEEE Symp. Rel. Distrib. Syst.*, 2004, pp. 66–78.

[55] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 91–96.

[56] J. Bonér (Lightbend). *Akka*. Accessed: Apr. 5, 2019. [Online]. Available: https://akka.io/

[57] K. S. Trivedi and A. Bobbio, *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2017.

[58] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Proc. IEEE Int. Symp. Fault Tolerant Comput.*, 1995, pp. 381–390.

[59] S. A. Vilkomir, D. L. Parnas, V. B. Mendiratta, and E. Murphy, "Availability evaluation of hardware/software systems with several recovery procedures," in *Proc. IEEE Int. Comput. Softw. Appl. Conf.*, vol. 1, 2005, pp. 473–478.

[60] S. Chandra and P. M. Chen, "Whither generic recovery from application faults? A fault study using open-source software," in *Proc. IEEE Int. Conf. Depend. Syst. Netw.*, 2000, pp. 97–106.

[61] V. B. Mendiratta, "Reliability analysis of clustered computing systems," in *Proc. IEEE Int. Symp. Softw. Rel. Eng.*, 1998, pp. 268–272.

[62] W. Xie, Y. Hong, and K. S. Trivedi, "Software rejuvenation policies for cluster systems under varying workload," in *Proc. IEEE Pac. Rim Int. Symp. Depend. Comput.*, 2004, pp. 122–129.

[63] D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," in *Proc. IEEE Pac. Rim Int. Symp. Depend. Comput.*, 2009, pp. 365–371.

Petra Vizarreta received the bachelor's degree from the University of Belgrade in 2009, the master's degree from the Karlsruhe Institute of Technology and Polytechnic University of Catalonia in 2011, and the Dr.-Ing. (Ph.D.) degree from the Technical University of Munich in 2019. She joined the Chair of Communication Networks, as a Researcher in September 2015. Her research interest include modeling and design of dependable softwarized networks.

Kishor Trivedi is a Professor of ECE with Duke University. He has authored a text book *Probability and Statistics With Reliability, Queuing and Computer Science Applications*. His latest book, *Reliability and Availability Engineering* (Cambridge University Press, 2017). He has supervised 46 Ph.D. dissertations. He was a recipient of the IEEE Computer Society's Technical Achievement Award for his research on Software Aging and Rejuvenation, he works closely with industry in carrying out reliability/availability analysis and in the development and dissemination of software packages, such as SHARPE and SPNP.

Veena Mendiratta (Senior Member, IEEE) received the B.Tech. degree in engineering from the Indian Institute of Technology, New Delhi, India, and the Ph.D. degree in operations research from Northwestern University, USA. She is a Consulting Member of Technical Staff in the End-to-End Network and Service Automation Lab, Nokia Bell Labs, Naperville, IL, USA. Her current research is focused on network reliability and analytics—architecting and modeling the reliability of next-generation programmable networks and development of analytics-based anomaly detection algorithms for improving network performance and reliability. She is a member of INFORMS, SIAM, and ASA.

Wolfgang Kellerer (Senior Member, IEEE) received the Dipl.-Ing. (Master) and Dr.-Ing. (Ph.D.)degrees from the Technical University of Munich in 1995 and 2002, respectively. He was with NTT DOCOMO's European Research Laboratories for over ten years. He is a Full Professor with the Technical University of Munich, heading the Chair of Communication Networks, Department of Electrical and Computer Engineering. His research resulted in over 200 publications and 35 granted patents. He currently serves as an Associate Editor for the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and on the editorial board for IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. He is a member of ACM and VDE ITG.

Carmen Mas-Machuca (Senior Member, IEEE) received the M.Sc. degree in 1996, and the Ph.D. degree in 2000. She is a Privat Dozent/Adjunct Teaching Professor with the Chair of Communication Networks, Technical University of Munich, Germany. She has published more than 100 peer-reviewed papers. Her main research interests are in the area of techno-economic studies, network planning and resilience, SDN/NFV optimization problems, and next generation converged access networks.