Availability Analysis of the ONOS Architecture

Michael Müller and Michael Köhler-Bußmeier

HAW Hamburg, Berliner Tor 7, 20099 Hamburg
michael.weitnau@live.de
michael.koehler-bussmeier@haw-hamburg.de

Abstract. In this work, we compare two ONOS architectures, old (before v1.14) and new (v1.14 and after), in terms of their availability and answer the question if the new outperforms the old architecture. ONOS is a widely used and popular open source SDN controller that changed his architecture with version 1.14 to enable in service software upgrades. For this we create a GSPN model, upon which we can present that the new architecture has a higher availability, especially in environments with less available hardware.

Keywords: SDN · ONOS · Consensus · Raft · Availability · GSPN.

1 Introduction

In 2014 the development of software defined networks (SDN) rapidly gained traction which kept up till today. Since then, ideas have been implemented, tested and altered [4, 16, 21, 25]. One such alteration was the extraction of the embedded data store in ONOS into a separate executable, how this change affected ONOS' overall availability is the focus of this work.

ONOS is one of the most popular open source SDN controllers, and it uses Atomix as its shared data store for a logical centered but physical distributed deployment. Atomix implements the Raft consensus protocol, for which the number and location of Atomix instances need to be immutable. From ONOS version 1.14 onwards, ONOS and Atomix are separated, this separation allows ONOS to be upgraded and horizontally scaled much easier than before.

Performance analysis of architectures [5, 20], SDNs [14, 15, 21, 10] and older ONOS versions [17–19] have been done before, but not with the newer ONOS versions and not with focus on its architectures. Additionally, authors write about the need for more performance evaluations in SDNs, especially regarding reliability and availability [15, 14].

As ONOS has availability within its core focus, this raises the question, if this architectural change was beneficial to ONOS' availability. To the best of our knowledge, this is the first attempt to evaluate and compare the availability of the two ONOS architectures. The contribution of this paper is an availability analysis to determine under which circumstances which architecture, old or new, offers higher availability. This analysis is based upon simulation results of a generalized stochastic Petri net model simulated within the tool GreatSPN.

The paper is structured as follows: In Section 2, we describe important basics for this work. Published work that is related to this paper, either by a similar goal, tool or formalism, is presented in Section 3. Then, we present and validate our model in Section 4 and Section 5 respectively. The last two Sections 6 and 7 answer the research question, conclude the paper and point out interesting future work.

2 Basics

In this section relevant basics are presented and explained, so the reader is well suited for the upcoming technicalities.

2.1 Dependability

Dependability is a group of concepts and attributes, the most relevant are explained in this section. [2, 24]

Reliability is the continuity of correct service [2]. A highly reliable system is a system that continuously works as expected for a long period of time, at best forever. An important metric for this attribute is 'Mean Time To Failure' (MTTF), as the name suggests, this represents the average of how long the service works without interruptions. The measurement of how long it takes to repair the interruption is 'Mean Time To Repair' (MTTR) [7, 24].

Availability is the probability for correct service in a given moment. A highly available system is a system that has a high probability to work as expected in any given moment, at best always. This probability can be calculated with $\frac{MTTF}{MTTF+MTTR}$ [2, 7, 24]. Once the system is stabilized and the availability is roughly a constant value, we can talk of it as being the 'Steady State Availability'. This is used in previous publications as the basis of their availability evaluations [2, 20, 21].

Threats to the dependability of a system are faults, errors and failures.

Faults are the basis of the threats, they can activate errors. They are either introduced during development, via incorrect code, due to physical problems, e.g. an old hard drive stops working, or are generated from outside the system, e.g. during interaction with the user [2].

Errors are part of the system's state and may activate failures if the error has external consequences. Errors can be detected if they generate any kind of message or signal [2, 24].

A **Failure** is in [2] defined as '[...] an event that occurs when the delivered service deviates from correct service', they can lead to the activation of further faults. [24] describes failures as unmeet specifications. An example failure is an *uncaught error* of a software that leads to a complete crash of the software.

2.2 Consensus

Consensus protocols help to synchronize a state across distributed systems. In consensus protocols there is one leader which will order incoming updates. It will then ask the participating systems if a certain order is consistent with their respective state. If the majority of systems acknowledge the update, it is committed by the leader and each system applies the update. The same way consensus can be used to elect a participating system to their leader [8, 21, 24]. One such consensus protocol is Raft[1]. Raft can work if the majority $(1 + \frac{instancecount}{2})$ of the participating instances are available [16, 25, 8, 21].

[1] Raft Visualizations: http://thesecretlivesofdata.com/raft/, https://raft.github.io/

2.3 Software Defined Networks (SDN)

Networks have a control and a data plane, while the control plane is more of a logical nature, the data plane is more of a physical nature. The control plane can insert, modify or delete forwarding rules to impact routes through the network. The data plane forwards packets from one ingress port to certain egress ports according to the set forwarding rules. In traditional networks, both planes reside in each router. SDNs split these planes, the SDN controller has the control over the routers which are only left with the data plane. Routers are then called SDN switches. The control plane can be 'in-band', on the same links as the data plane, or 'out-of-band', on own links. As an example, when in an out-of-band control plane a controller has a direct connection to a certain switch, the same connection in in-band control planes may require additional switches in between. [14, 15]

The controller communicates in three major ways [4, 14, 15]. 'Southbound' (e.g. via OpenFlow) to the switches, 'Northbound' (e.g. via HTTP) to SDN applications and 'East-/Westbound' to other SDN controllers. SDNs have the benefit, that protocols and devices are easier to update or replace. Furthermore, SDNs can improve resource optimization, ease of maintenance and ease of operation. [14, 15]

2.4 ONOS

The 'Open Network Operating System' (ONOS) is, next to OpenDaylight, the largest open source SDN controller and is widely mentioned in publications [25, 23, 21]. ONOS was created in 2014 on the basis of the Floodlight SDN controller. Its target was to tackle high throughput, low latency, large network state sizes and high availability, detailed performance numbers can be found in [4]. To handle this in an efficient and partition tolerant manner, the controller is physically distributed and logically centered. It uses the Raft consensus algorithm to synchronize the shared network state between instances, currently implemented via Atomix [9, 4, 16].

3 Related Work

This section will present published work with a similar context in Section 3.1 and work with similar tools or formalism in Section 3.2.

3.1 Performance Evaluations with similar context

[5] uses Markov-based reliability prediction in the context of highly adjustable software. Their model considers the reliability of software, hardware and network. In the first case study they have artificial failure probabilities and in the second they extend previously published work by considering e.g. new fault tolerance mechanisms.

The authors of [14] predict the steady state availability of SDNs and traditional IP networks. Their model considers the availability of traditional IP routers, SDN switches, SDN controllers and links. The model is split in two hierarchical layers to avoid a potential '[...] uncontrolled growth in model size [...]'. One contains the connections between the network elements, based on minimal cut and path sets, and the other contains the failures and recoveries of each network element, modeled with Markov chains.

[15]'s goal is to assess the steady state availability of a generic SDN using a stochastic availability model. Like [14], they propose a hierarchical availability model. Their model does

not consider failures of the SDN controller. The model was implemented using the Symbolic Hierarchical Automated Reliability and Performance Evaluator.

The goal of [21] is to evaluate the response time and availability of distributed SDN clusters like the old (pre v1.14) ONOS architecture and the OpenDayLight SDN controller with Raft as its distributed data store. They use stochastic activity networks as a model generation framework. Their 'RAFT Recovery SAN Model' contains hardware and software failures that can impact the SDN controllers. To evaluate the response time they also model failure injection to cause further failures that can be correlated. They consider a coarse-grained static data plane reliability and propose the evaluation of the worst case only, to be able to scale out this performance evaluation approach for larger models.

The goal of [10] is to offer mitigations on the basis of their proposed modelling framework that considers reliability, availability and security in distributed consensus protocols like Raft. Their model considers among others, failure probabilities from published work (includes ONOS related ones), detectability of failures and multiple repair rates.

3.2 Performance Evaluations with similar tooling or formalism

The GreatSPN tool and the GSPN formalism are widely used in the context of performance evaluation, for example in following publications.

[3] presents a case study to analyze and evaluate UML diagram types based upon Stochastic Well-Formed Nets (SWN). GreatSPN is used as a translator with the 'GreatSPN-to-PROD' utility, as a solver with the 'algebra', 'Multisolve' utilities and to create images of the models.

[11] evaluates performance characteristics of a LAN with a single bus and multiple devices. GreatSPN and GSPN are used for validation and solving.

[22] validates a transaction protocol in a mobile environment. GreatSPN and SWNs are used for validation and solving with the 'WNSIM' utility.

[6] analyzes the performance of query routing to create a new algorithm for needed meta-data. GreatSPN and SWNs are used for validation, evaluation and simulation with the 'WNSIM' utility.

[12] creates a translator from UML activity diagrams to GSPNs for performance evaluations, which extends their previous work with similar goals. The created GSPNs are recommended to be analyzed with GreatSPN.

4 Model

In this section, we will present and reason the elements and parameters of our assumed model.

4.1 The ONOS Architectures

The work on a new ONOS architecture arose 2017 with the formation of the ONOS internal 'In Service Software Upgrade' team[2]. An ISSU means to upgrade software that is currently running and answering requests. The goal is to upgrade without a loss of availability. For this goal the ISSU team decided to change ONOS' architecture: '[...] In past versions, ONOS embedded Atomix nodes to form Raft clusters, replicate state, and coordinate state changes. In ONOS 1.14, that functionality is moved into a separate Atomix cluster' [16].

[2] Also called ISSU Brigade lead by Jordan Halterman

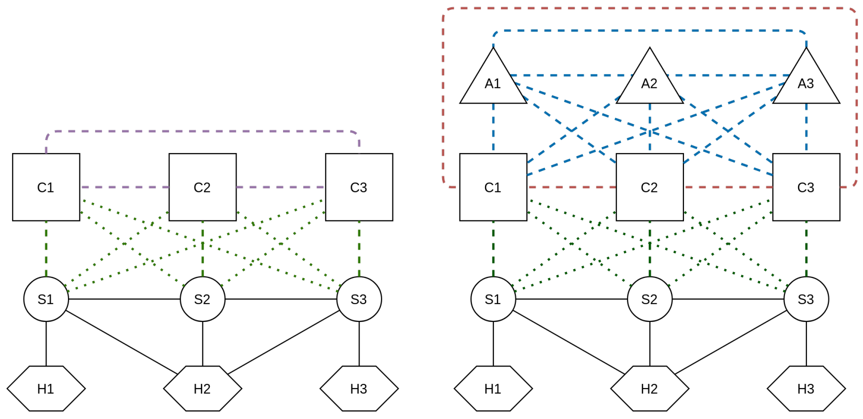

Fig. 1. Cluster of ONOS before v1.14 (left) and (after) v1.14 (right)

The differences between the architectures is best presented with two example clusters. The example cluster of an old ONOS architecture in Figure 1 contains three item types, ONOS controllers (C1-C3, squares), SDN switches (S1-S3, circles) and hosts (H1-H3, hexagons). Each controller has contact to each switch (green dotted lines), but only one controller is the master of a switch (bold green dotted lines). Also, there is a communication between the controllers (purple dotted lines). The black lines between switches and hosts are signaling the connectivity between these items. The example cluster of a new ONOS architecture in Figure 1 is noticeably noisier. Here we have one item type more in the cluster, Atomix instances (A1-A3, triangles). Additionally, to the Atomix instances, we have more links, these connect each Atomix instance with each controller (blue dotted lines). Our controllers still have their controller to controller communication (red dotted lines).

The interesting point is the mentioned additional noise in the new example cluster. This separation of Atomix and ONOS brings certain flexibility benefits like dynamic horizontal scaling of the ONOS instances and separate and easier upgrade of ONOS and Atomix instances. It also brings some costs in form of additional links, communication, instances and nodes, if each instance is deployed on its own node. This leads us to question if this new architecture benefits the overall availability. We want to analyze the following points:

- Do the additional elements in the cluster harm the overall availability?
- If so, does it only decrease cluster availability in certain scenarios?

4.2 Model Elements

To make the description of the model easier, we classify elements of our model as either objects or behaviour of these objects. In the following we will present included and excluded model elements and argue why.

Objects are either *up* or *failed*, like for example in [5], *up* objects can fail, *failed* objects can recover. Initially all objects are *up*.

Included model elements :

- We consider five versions of **ONOS instances** in our work. v1.13, v1.14 and three future placeholder versions of ONOS that are based upon v1.14. The placeholder versions are named 'v1.14 vX' where X is 2, 3 or 4, 'vX' for short. 'v1.14 v1' would be the normal v1.14. Each version has its own software failure rate.

- We also consider **Atomix instances**. Together with the ONOS instances, these elements are the core of our model.
- We consider links between Atomix and Atomix and between ONOS and Atomix.
- As an own object we model the **consensus protocol status**. The consensus protocol is *up*, if the majority of Atomix instances are *up*. Otherwise, it is *failed*. **Reason**: This is due to the basic requirement of the Raft consensus protocol, which needs an active participation of the majority of instances as explained in Section 2. In the following sections we will use 'consensus' and 'consensus protocol' interchangeably.
- The **cluster availability** is also modeled as an object. It is *up*, if consensus and at least one ONOS instance is *up*, in any other case it is *failed*. **Reason**: Only if consensus is *up*, which includes that enough Atomix instances are *up*, and at least one ONOS instance is *up*, the cluster can work as intended and so only then it is available.
- **Failure of ONOS and Atomix** instances can be due to hardware failures, software failures and network partitions. Every instance has its own hardware node and all nodes are equal. **Reason**: The consideration of these failures is intuitive. Besides that, we assume that each instance has its own node to achieve a simpler model in contrast when we would consider shared nodes. This node separation can also be found in [14].
- If consensus fails, the **Atomix and ONOS instances become idle** and can no longer fail from software problems. **Reason**: This is based upon the behaviour of Atomix and ONOS described in [16], in that case they can not fully answer incoming requests and only execute a limited amount of their code, which again means less possibility to execute faulty code.
- For the old ONOS architecture we consider the **combined failure of ONOS and Atomix**. If one fails, the other fails too. **Reason**: This is due to the deeply coupled deployment of ONOS and Atomix instances in the old architecture as described in [16].
- We consider worst case **network partitions** by accounting the number of failed links. Once a certain amount of links has failed, one respective instance fails. Links are either between the Atomix instances or between Atomix and ONOS instances. In the example in Figure 1 this would mean that for each two Atomix links one Atomix instance fails, or that after three failed ONOS-Atomix links one ONOS instance fails. **Reason**: The consideration of link failures and partition is an interesting topic for this work as it hits one of the main points of the architecture change, the additional links. We do not consider specific node connections of links, as it would drastically increase the models complexity and also would overstep set time limits of this work. As for why we see this as a failure, with reference to [16]: Atomix / ONOS instances stop serving requests and wait for the partition recovery once they are partitioned. This is similar to the case when consensus fails, as explained above. The ONOS-Atomix partition does not lead to a failed Atomix instance, as Atomix is not dependent on ONOS instances, but it is the other way around.
- In the new architecture, **the cluster can be upgraded** if all Atomix and ONOS instances are *up*. First Atomix will be upgraded, to be compatible with the new ONOS version, then ONOS is upgraded. The influence of the upgrades onto ONOS' availability are dependent on the used parameters for each version and theoretically could improve or worsen its availability. We will perform a rolling upgrade, upgrading one instance after another. This also means that the multiple versions are compatible to each other. The upgrade can fail, which leads to a longer upgrade time, but it can not fail entirely, so a rollback is not necessary to consider. **Reason**: The rolling upgrade procedure described

is based upon their mailing list[3] and [16]. That an upgrade only temporary fails, is based upon the assumption that an expert performs the upgrade and also verifies the success, as the ISSU team mentions in their presentation[4] of the upgrade procedure.

Excluded model elements :

- Our model focuses on the control plane, so the **data plane** is not considered. **Reason**: As it can be observed from the old cluster to the new cluster in Figure 1 the connections (green dotted lines) between SDN controller, SDN switches and hosts do not differ. We reason that the architectural change in the control plane does not affect the data plane in any way and so the impact of the data plane on the overall availability is equally in both architectures and so can be omitted as we are only interested in the changes between the two architectures.
- We model a **failure detection** probability of 100%, which in other words excludes this from the model. **Reason**: This approach simplifies our model and can also be found in [14, 15].
- We do not consider **load** related software, hardware or link failures. **Reason**: In our opinion, this would lead to a much more complex model that would cost sparse time for creation we rather spend on other areas we think have a greater benefit for this work.
- The **stacking of failures** is not considered. A stacking failure could be, that on top of being already partitioned, one ONOS instance also then fails due to a software failure. **Reason**: An idea of how complex it is to model this detail, can be seen in the Section A.1 in the Appendix. It would extend the time limits of this work to incorporate this into the model.
- As described, **Atomix** instances must be **upgraded** before ONOS can, but this has **no impact on the overall availability**. **Reason**: Since this upgrade is done so Atomix is compatible with the new ONOS version, we do not expect it to also include an impact of availability for example through bug fixes. Henceforth, we exclude this detail from our model.
- The **old architecture can not be upgraded**. **Reason**: We reason this exclusion with our goal to compare between less complexity and less flexibility in the old architecture, against more complexity and more flexibility in the new architecture, as discussed in Section 4.1.
- The **gossip protocol** that is implemented in ONOS is excluded. **Reason**: As the gossiping is only used for bidirectional exchange of state, it is used to detect smaller state drifts between instances and to bring new instances faster up-to-date. If consensus is not longer working as mentioned above, the gossip protocol can't work either. [16]
- Unavailability due to attacks or other **security** related threats to availability are not modeled. **Reason**: As security is a whole new topic that has a depth on its own which would extend the scope of this work too much. This approach can also be found in [10], as mentioned in Section 3.

[3] See https://groups.google.com/a/onosproject.org/g/onos-dev/c/iu_iP8pFs-U/m/OGtYzVy_CwAJ, https://groups.google.com/a/onosproject.org/g/onos-dev/c/hzfjjEyruGo/m/xsnEiMCdAwAJ and https://docs.google.com/document/d/1-xZ3Wnr6VZS34paYVdZhF8WWo3M6VWKNJXQTmBnnM7Y

[4] See https://wiki.onosproject.org/display/ONOS/ISSU within [16]

4.3 Used Parameters

According to the included elements explained above, we will now present the used parameters. As for wording we use 'parameter' and 'rate' interchangeably.

Rate and Source	Value
ONOS v1.10 failure rate (from [10, 13])	50 per year
ONOS v1.13 failure rate (calculated based upon v1.10 failure rate and the relative percental difference from v1.13 to v1.10 in Table 4)	30 per year
ONOS v1.14 failure rate (calculated like v1.13 failure rate)	28.28 per year
ONOS 1. Upgrade failure rate (calculated like v1.13 failure rate)	22.47 per year
ONOS 2. Upgrade failure rate (calculated like v1.13 failure rate)	20.10 per year
ONOS 3. Upgrade failure rate (calculated like v1.13 failure rate)	18.05 per year
ONOS v1.10 recovery rate (from [10, 13])	2 per hour
Atomix Software failure rate (see above, like ONOS v1.13)	30 per year
Atomix Software recovery rate (see above)	6 per minute
Cluster upgrade rate (see Section 4.2 element **EB4**)	4 per year
Atomix upgrade rate (see above)	12 per hour
ONOS upgrade rate (see above)	4 per hour
ONOS upgrade rate with encountered failures (see above)	6 per day
Hardware failure rate (from [21, 14])	2 per year
Hardware recovery rate (from [21, 14])	2 per day
Link failure rate (from [14])	3 per year
Link recovery rate (from [14])	4 per hour

Table 1. Used failure, recovery and upgrade rates

The ONOS parameters from [10, 13, 19] do not include Atomix. We can therefore use them as purely ONOS parameters. This is important to point out as in older versions Atomix and ONOS were very closely coupled, as described in Section 4.1.

For Atomix we could not find any published work that states specific values for it. What we could find was a software failure rate of one week in [21], but this value is for the complete SDN controller that implements the Raft consensus protocol. The same programmers that wrote ONOS wrote Atomix. Therefore, we assume that both have the same reliability, so we set the **Atomix failure rate** equal to the failure rate of ONOS v1.13.

The Atomix software repair duration is set by us to 10 seconds. This number is gathered by a coarse grained test we did by starting and stopping an Atomix container within a Virtual Machine with 6 vCPUs and 14 GB RAM with an i7-9750H. We assume that an ONOS deployment could be done via Kubernetes, as they suggest on their website[5]. The number above is reasoned like following, one second is the time Kubernetes needs to perform liveness and readiness probes to check the status of the Atomix instance. If the instance is detected as unhealthy, it is shutdown which takes up around two seconds and starts up a new one on the same host which takes up roughly seven seconds. As we deploy the new Atomix instance on the same node we need to wait for the old one to be shutdown.

Getting the Atomix and ONOS upgrade durations is hard, as there exists no documentation for either one online. Furthermore, the tools that an installed ONOS version provides, do not contain hints on how to upgrade an existent cluster. As the upgrade of Atomix and

[5] See https://atomix.io/docs/latest/user-manual/deployment/kubernetes/

ONOS instances is mainly done via shutting down the old version and starting a new version instance in its place[6], one could argue that we plainly could take the time that needs. So for Atomix that would be nine seconds, as described above minus the one second for failure detection that is not necessary in this proactive process. For ONOS a restart takes around seven seconds, measured in the same environment. The problem is that these numbers seem pretty low for a major version upgrade. For this reason we set the upgrade duration arbitrarily.

For **Atomix upgrade** this means five minutes with regard to the easy upgrade as described in Section 4.2. We set the **ONOS upgrade** durations to 15 minutes, for the failure free upgrade, and four hours, when failures are encountered by the expert. 15 minutes, since we assume that upgrading ONOS is more complex than to upgrade Atomix due to expected data migration. The four hours are a mean of considering pretty simple errors, like wrong IP address in configuration, and more complex ones, like (partly) failed data migration.

In Table 4 we listed all sourced or calculated bugs per hour per ONOS version. Bugs per hour of v1.10, v1.12 and v1.13 are based upon published work, these are also the basis for our calculation of v1.11, v1.14 and the upgrades after v1.14. Version 1.14 and the upgrades are based upon the logarithmic trend, as we expect that fewer bugs are getting fixed for each release as it gets more time intensive to fix them.

4.4 GSPN model in GreatSPN

In the following section we will present one part of our GSPN model in detail[7]. Figure 2 contains the following Atomix elements:

Places
A_{up} : Holds tokens that represent *up* Atomix instances.
$A_{SWfailed}$: Holds tokens that represent *failed* Atomix instances due to software failure.
$A_{HWfailed}$: Holds tokens that represent *failed* Atomix instances due to hardware failure.
CO_{up} : Holds at most one token, representing that consensus is *up*.
CO_{failed} : Holds at most one token, representing that consensus is *failed*.

Transition
F_{SW} : This transition represents a software failure of an Atomix instance. When it fires, it moves one token from the A_{up} place into the $A_{SWfailed}$ place. Additionally, it has an inhibitor arc to CO_{failed} and its rate is defined by the $AtomixSWFailRate$ parameter.
R_{SW} : This transition represents a software recovery of an Atomix instance. When it fires, it moves one token from the $A_{SWfailed}$ place into the A_{up} place. Its rate is defined by the $AtomixSWRecRate$ parameter.
F_{HW} : This transition represents a hardware failure of an Atomix instance. When it fires, it moves one token from the A_{up} place into the $A_{HWfailed}$ place. Its rate is defined by the $HWFailRate$ parameter.
R_{HW} : This transition represents a hardware recovery of an Atomix instance. When it fires, it moves one token from the $A_{HWfailed}$ place into the A_{up} place. Its rate is defined by the $HWRecRate$ parameter.
F_{CO} : This transition represents the failure of the consensus protocol. When it fires, it moves one token from the CO_{up} place into the CO_{failed} place. Additionally, it has an inhibitor arc to A_{up} with the multiplicity of $atomixmajority$, whereas $atomixmajority$ is equal to the rounded down result of $\frac{atomixcount}{2+1}$. Its rate is immediate.

[6] See https://wiki.onosproject.org/display/ONOS/ISSU within [16]
[7] The full model can be seen on GitHub (https://github.com/HansZimmer5000/PNSE21) or in the Appendix (Figure 11)

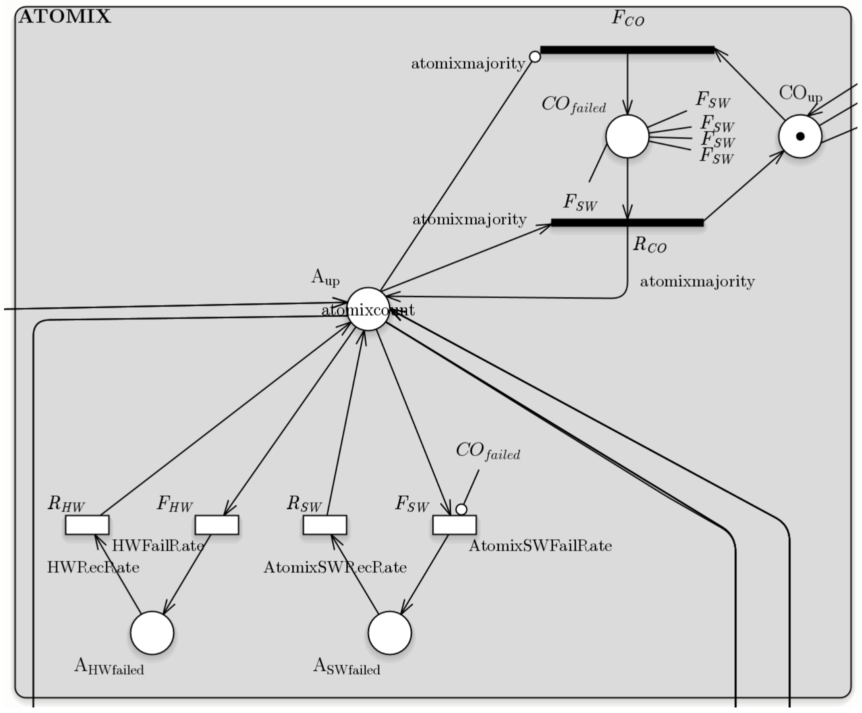

Fig. 2. The grey Atomix square in GreatSPN

R_{CO} : This transition represents the recovery of the consensus protocol. When it fires, it moves one token from the CO_{failed} place into the CO_{up} place. Additionally, it has one input and output arc to A_{up} with the multiplicity of *atomixmajority* each. Its rate is immediate.

5 Validation

We validated our model with simulations. GreatSPN features tools to analyze performance aspects of a GSPN or SWN model, e.g. with the 'WNSIM' tool mentioned which was used by [6, 22]. The results of a WNSIM simulation are the mean number of tokens for each place and the mean throughput for each transition throughout the simulation time. As our cluster steady state availability, described in Section 2.1, we take the mean number of tokens in the $Cluster_{up}$ place. This is possible since the number of tokens is at most one, as described in Section 4.2, which can represent a percentage with its value between 0 and 1. To get this value, we do not have to wait for the simulation to finish as it constantly creates fine-grained logs from which one can gather the cluster availability. This allows us to interrupt simulations if they are running too long. For all simulations we use a confidence of 90% and an approximation of 50%. Lower approximation leads to more precise results, but also to immensely increased simulation times. With the current value we have simulations that do not finish within six hours. This time is not negligible, as we execute for example 160 simulations[8]

[8] $architectures * stepcount * repititions = 2 * 16 * 5$

for the upcoming *HWFailRate* parameter sensitivity analysis alone. The simulations are executed on an i7-9750H CPU with 12 threads and 32 GB RAM.

5.1 Sensitivity Analysis

For our sensitivity analysis, we interrupt a simulation after one hour. Each sensitivity analysis analyzes one parameter. All others parameters are unaltered, see Section 4.3 for parameter default values. The value of an analyzed parameter is altered with each 'step'. We configured our steps to be between -5 and 10 inclusive, each step differs 10% $*$ *step* from the default value which is used in step 0. For example with the *ONOSSWRecRate* parameter we have the different values per step:

Step -5, -50% : $1440 * 0,5 = 720$
...
Step -1, -10% : $1440 * 0,9 = 1296$
Step 0, +0% : $1440 * 1 = 1440$ (default value)
...
Step 10, +100% : $1440 * 2 = 2880$

For parameters with integer values, the difference is $1 *$ *stepno*. Each step is repeated five times. Each repetition starts with its own seed which is the value of BASHs $RANDOM variable at that moment. So from a different perspective, each analyzed parameter, step, repetition and architecture is a separate simulation with a unique seed. As an optimization, not all parameters are executed in all architectures. For example, the upgrade parameters are only analyzed in the new architecture as they have no impact in the old architecture.

In all simulations, the step difference did not propagate to the cluster availability. In other words, when the value of a parameter was increased by 10%, the cluster availability did change by less than 10%. We present one sensitivity result in more depth, as an example how these results have to be read and interpreted. In Figure 3 we can see the mean result of multiple sensitivity analysis of the *UpgradeClusterRate* parameter. In the sensitivity analysis we focus on how much a paramter change influences the overall result. The results have a blue line if the sensitivity analysis was done only once, and it is green like in our example if the analysis was repeated multiple times, the green line then represents the mean. Like explained above, the sensitivity analysis has 16 steps, from -5 to 10 inclusive. Three exemplary steps of the mentioned sensitivity results and their meaning are described next:

Step 0 : The base value of the sensitivity analysis is always step 0 as it represents results with the default values of our model from Section 4.3. In this case step 0 results in a cluster availability of $0,9999872$ ($99,99872\%$).
Step -5 : The parameter has -50% of its default value ($1440 * 0,5$). This results in a cluster availability of $0,9999817$. This is a difference to our base cluster availability of about $0,00055\%$, this is far from -50%.
Step 4 : The parameter has +40% of its default value ($1440 * 1,4$). This results in a cluster availability of $0,9999904$. This is a difference to our base cluster availability of about $0,00032\%$, which is far away from +40%.

So the change of the parameter value did influence to the cluster availability, but not by the same percentage.

This can be observed for all other results too. More result examples like the one just described. Some of them will be discussed multiple times in the upcoming sections, the figures are placed next to the discussion that depends the most on it.

5.2 Simulation Results Plausibility

Besides looking at sensitivity we are now looking at the plausibility of some simulation results, partly already created during sensitivity analysis.

In contrast to the sensitivity analysis, it is very important for the plausibility analysis to consider the context of the analyzed parameter, as different parameters have different impact on the cluster's availability. For example an increased recovery rate is expected to increase the cluster availability while an increased failure rate is expected to decrease it.

For the basic architecture results we take both unaltered architectures, execute each five times and calculate the mean of all repetitions. Like a sensitivity analysis without steps. The simulation results reveal an availability of 99,9981% for the new and 99,99550% for the old architecture. These results are just slightly above of the 99,99% availability that ONOS specified themselves [16, 4]. Keep in mind that these sources are based upon an older version of ONOS in the old architecture. With reference to Table 4, ONOS' availability most certainly improved over the last few years, so we are confident that the basic architecture results are plausible.

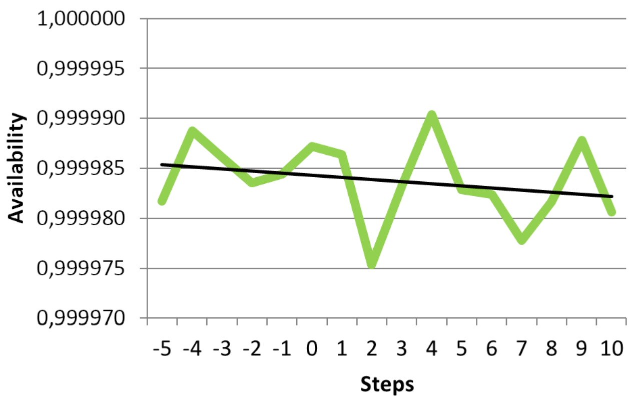

Fig. 3. Cluster availability (green) and linear trend line (black) of the *UpgradeClusterRate* parameter steps in the new architecture

The plausibility of our sensitivity analysis results is shown in the following.

Exemplary in the Figures 4 and 6 we can observe the **hardware failure rate** and **link failure** results. The results match our expectations, as the increasing failure rate decreases the cluster availability in both architectures.

In Figure 3 we can also observe that sometimes the results line takes rapid changes. This could be a hint that we have set a too low repetition count, and we need a higher variety of results per steps to get a more consistent course of the steady state availability. Even though, the differences between steps are minimal, they begin to differ from the fifth decimal place onwards. For this reason we do not further investigate this problem.

The **upgrade cluster rate** results in Figure 3 show a mean of multiple sensitivity analysis as the result is counterintuitive, hence a green instead of a blue line. With a faster upgrade

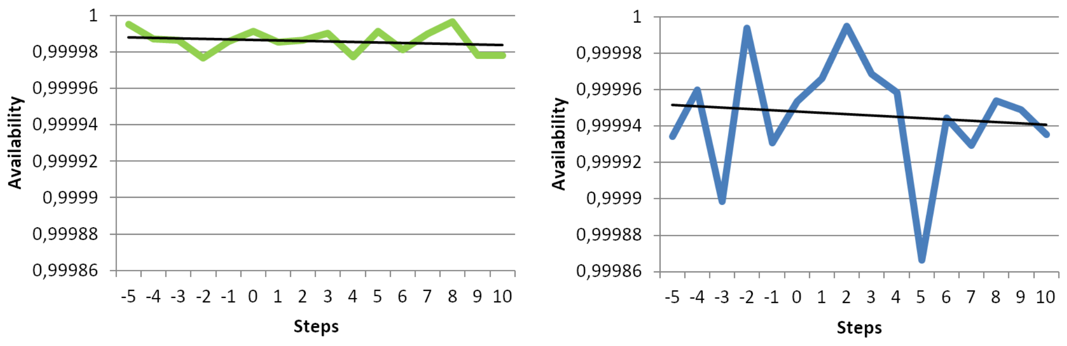

Fig. 4. Cluster availability (green, blue) and linear trend line (black) of the $LinkFailRate$ parameter steps in the new (left) and old (right) architecture

rate the cluster availability decreases slightly. Even though the observed difference from the start of the trend line to its end is minimal[9], we want to give an idea why the trend line could behave this way. With a higher upgrade rate, the cluster has more often instances that are being upgraded, which takes the cluster one instance closer to unavailability. By that we mean, if a cluster has three Atomix instances and one is being upgraded, the cluster is only left with two *up* Atomix instances, if one of them fails, the cluster becomes unavailable as too few *up* Atomix instances are left to keep the consensus *up*.

6 Evaluation

In this section we will compare the availability of the two ONOS' architectures. First by taking a new look at the results from the sensitivity analysis from Section 5.1 in Section 6.1. Afterwards we will present specific deployment scenarios and interpret their results in Section 6.2. Finally, in Section 6.3 we will answer the research question, if the architecture change was beneficial for ONOS' availability. It will turn out that the new architecture has an overall higher availability than the old architecture.

6.1 Sensitivity Results interesting for architecture comparison

In the following we will present results of the sensitivity analysis results originally intended for Section 5, but they fit much better here as they let us compare the availability in both architectures. These results may not have been gathered if we had executed the simulations manually like mentioned in Section 5.

Link failure and recovery rate sensitivity results can be seen in Figures 4 and 5. We can observe that the new architecture does not depend on link availability as much as the old architecture does. It also has a higher mean availability throughout the comparison.

We come to this conclusion for the link failure rate since the new architecture trend line changes slightly. The old architecture trend line has a bigger difference between its start and end value. In other words, the higher failure rate is more noticeable in the old architecture.

[9] With an absolute value of about 0,0000025 $(0,999985 - 0,9999825)$

Additionally, the new architecture trend line stays on a high level between 0,99998 and 1. The old architecture comes into that range with two steps, but its trend line is way lower, between 0,99995 and 0,99994.

For the link recovery rate we can observe and interpret the same. The new architecture trend line is on a higher level and also changes slightly, while the old architecture trend line is way lower and has a higher change from the trend line start to end.

At last, one remark to [15]. It concludes that link availability is of vital importance to the SDN steady state availability. In the following, we will present the hardware rates. When we compare their availability results with the ones of the links, we can see, that in the new architecture the hardware availability has a greater impact than link availability. We reason this as the hardware results are generally on a lower level and their trend lines are steeper. The same could be said about the old architecture, although here the results from the hardware rates and link rates are noticeably closer.

Hardware failure and recovery rate sensitivity results can be seen in Figures 6 and 7. Both architectures behave the same when the hardware failure and recovery rates change. Different is the degree of which the availability is impacted.

The hardware failure rate sensitivity analysis for the new and old architecture in Figure 6 clearly shows, that the new architecture is less dependent on its hardware, as its trend line declines less steep when compared to the old architecture.

In the old architecture the hardware recovery rate develops slightly worse than in the new architecture, as with better recovery rates the availability does not rise as fast as in the new architecture. It also does not reach cluster availabilities beyond $0,99998$ as often as the new architecture does, especially in later steps when the recovery rate is 180% (step 8) to 200% (step 10) of its default value.

Like observed in the link rate results, the availability of the new architecture is generally higher and less dependent on other elements, compared to the old architecture.

6.2 Scenarios

Additional to our results from Section 5, we now present scenarios to compare the availability in both architectures with specific viewpoints for a more in depth comparison.

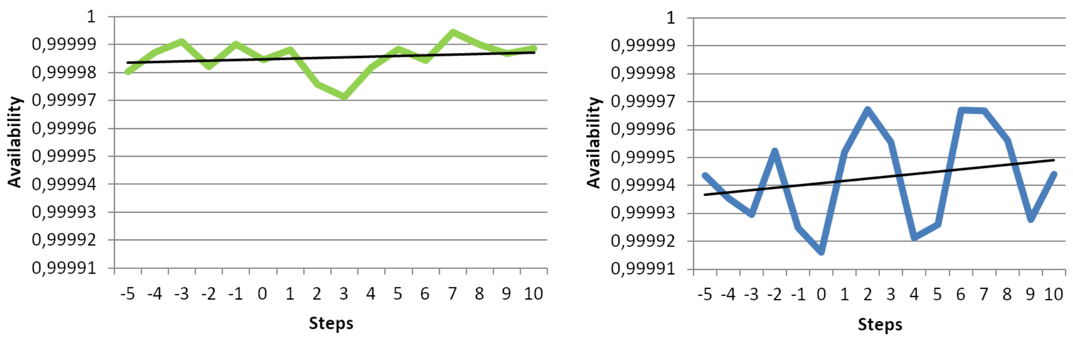

Fig. 5. Cluster availability (green, blue) and linear trend line (black) of the *LinkRecRate* parameter steps in the new (left) and old (right) architecture

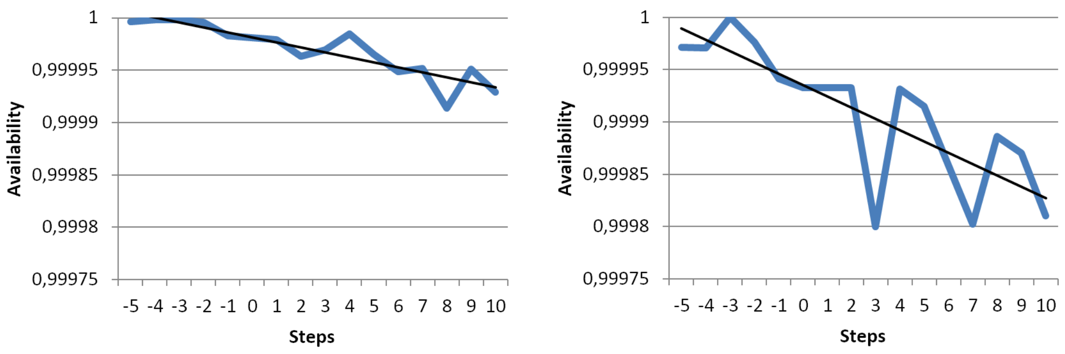

Fig. 6. Cluster Availability (blue) and linear trend line (black) of the *HWFailRate* parameter steps in the new (left) and old (right) architecture

For the upgrade scenarios we simulate four deployments. Two are the unaltered v1.14 and v1.13 architectures. The other two are altered v1.14 architectures. One has no upgrade at all, which means that ONOS is never upgraded to improve its availability. The other considers upgrades, but all ONOS versions have the same availability, so if for example new bugs are introduced with a release, others are fixed. In the following this scenario is called 'stagnating'.

The results of this deployment scenario can be seen in Figure 8. On the left side we can observe that the deployment with a stagnating upgrade in the new architecture has the worst availability. Second is the old architecture which is followed up by the unaltered new architecture and the new architecture without any upgrade.

It makes sense, that the new architecture has a higher availability than the old architecture, as already presented in Sections 5.1 and 5.2. That a new architecture with unchanging failure rates after each upgrade performs the worst can be reasoned with the point that an upgrade always means that during an upgrade of an instance we are one instance closer to unavailability, as described in Section 5.2. So in other words we 'buy' the upgrade with availability, but the availability does not improve afterwards to balance out the cost.

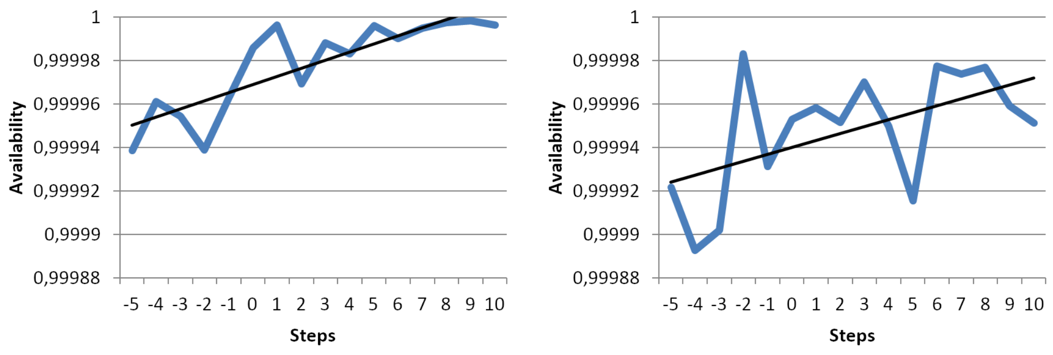

Fig. 7. Cluster Availability (blue) and linear trend line (black) of the *HWRecRate* parameter steps in the new (left) and old (right) architecture

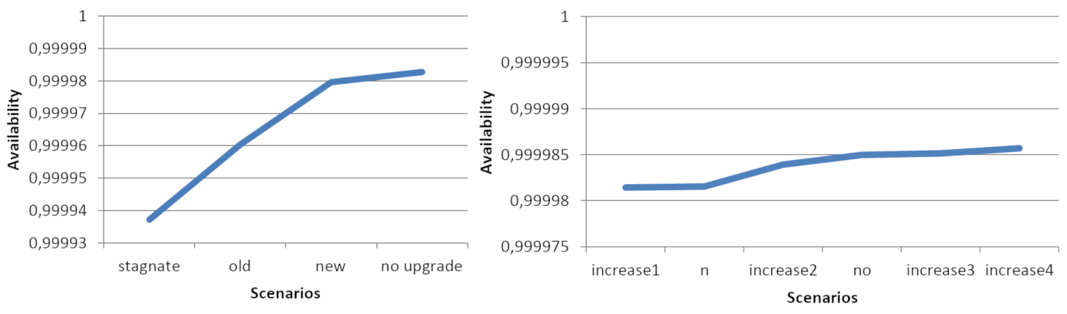

Fig. 8. Cluster availability of the old and new architectures, including different upgrade behaviours

We expected that the unaltered new architecture would result in the highest availability, as it contains the improving ONOS availability with each upgrade. Because of this, it is a surprise that the new architecture without any upgrade has the best availability, even though the difference is very low. This surprise maybe due to the fact, that the availability improvement after an upgrade is not high enough to balance out the mentioned cost.

	ONOS software failure rate decrease		
Scenario	v1.14 v2	v1.14 v3	v1.14 v4
'increase1'	2%	4%	6%
'increase2'	8%	10%	12%
'increase3'	14%	16%	18%
'increase4'	20%	22%	24%

Table 2. Decreases of the ONOS software failure rates in the 'increase' scenarios

To test if this assumption is true, we simulated the new architecture in four additional scenarios as an extension to the presented scenarios above. Each 'increase' scenario increases the ONOS availability, by decreasing the ONOS software failure rate of the upgrades by $(2 + (increasenumber - 1) * 6)\%$. This is done to see, how much the upgrades need to improve the ONOS availability before we can balance out the cost of an upgrade. All decrease percentages are listed in Table 2.

The results can be seen on the right side in Figure 8. We do not further explain the difference between the 'increase1' and the basic new architecture results since their difference is negligible[10]. We can observe, that with increasing ONOS availability, the results come closer to the 'no upgrade' scenario and even can surpass it with 'increase3' and 'increase4'. So if our assumed ONOS software failure rates would decrease significantly of about the value of 'increase3' in Table 2 or more, the cluster availability would benefit from upgrades. In other words, the ONOS software failure rate would be needed to decrease around 23% for the first, 33% for the second and 41% for the fourth upgrade when compared to the v1 software failure rate to balance the overhead. An overview of the default and needed software failure rates and their decreases in percent can be found within Section A.2 in Table A.2.

[10] 0,000000069 $(0,99998152 - 0,9999814151)$

Different Atomix and ONOS counts are set in our next deployment scenarios. The combinations of the two counts are arbitrarily set.

The results can be seen in Figure 9. Scenarios that deploy the old architecture are named like *<onoscount>'o'* and the new architecture deployments are named like *<onoscount><atomixcount>'n'*. We also will use the term of the 'consensus instances', by which we mean ONOS instances in the old and Atomix instances in the new architecture. We are creating this term as it shortens the explanation, and they respectively are the basis for the consensus status.

Interesting to see, is that the consensus instance count mainly influences the cluster availability, to observe between *23n* and *57n* as they are sorted by the second number. With one exception to *29n* which has a very high consensus instance count, but with a very low ONOS count which we assume diminishes the availability, because the cluster is close to unavailability with just two ONOS instances. Especially during an upgrade, with reference to Section 6.2.

Although this work only looks at the availability a deployment can offer and does not consider for example load on the instances or operational costs, we want to point out, that an availability beyond 99,999% is achieved in the old architecture with fewer instances than in the new architecture. To reach this availability, the old architectures needs at least five ONOS instances, while the new architecture needs at least three ONOS and five Atomix instances. This result could be interesting for future work, e.g. if this is still true once the model considers load on the instances or the availability is put into context with operational costs.

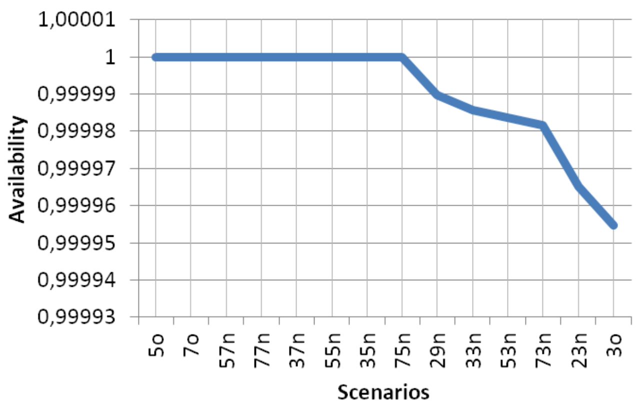

Fig. 9. Cluster availability of the recommended deployed instance counts

6.3 Evaluation Summary

The new architecture within ONOS v1.14 or newer, should especially be considered in deployments with lower hardware and link availability as shown in Section 6.1. Especially if the hardware's reliability or maintenance is worse than what we assumed in Section 4.3. This is important as it could mean in this case to increase unavailability within a year from 26 minutes (99,995%) to 657 minutes (99,875%)[11].

That the new architecture performs better in terms of availability than the old architecture can also be seen in Section 6.2 in our upgrade and deployment scenarios. In our upgrade scenarios we could observe that the upgrade comes with an availability cost. So operators of ONOS in the new architecture should not update every release but pick the ones that are needed, when viewed from the availability point of view. With reference to [1], one should also wait a few months after the release before upgrading.

When we looked at the instance counts, we could see that for the old architecture at least five ONOS instances should be deployed. In the new architecture the operator should deploy at least five instances of Atomix and at least three instances of ONOS from an availability point of view. In these deployments, both architectures can reach a cluster availability beyond 99,999%.

So, to answer our research question which architecture has a higher cluster steady state availability, we strongly recommend ONOS v1.14 or newer with the new architecture over ONOS v1.13 and older with the old architecture. Especially in environments with worse hardware or link availability than we assumed. The higher availability of the new architecture in these environments does mainly come from the increased count of links and hardware nodes. And so its higher tolerance towards single failures of such elements.

7 Conclusion

In this work we presented and evaluated the research question, if the additional complexity in the new architecture harms the overall availability and if so, if this is only the case in certain scenarios.

This research question was formulated and reasoned in Section 4. In this section we also defined our model elements. Our GSPN model was then validated in Section 5, with emphasis on the results of automated simulations. With them, we could show, that no parameter overwhelmingly influences the results. And we explained that these are plausible. Although some results were counterintuitive at first sight, they could be reasoned. In Section 6 we took once more a look at the results of simulations and could observe that the new architecture performs better than the old architecture in terms of availability. This was confirmed with specific deployment scenarios. The section was concluded with, besides others, recommendations for ONOS deployments. Even though, the fact that the new architecture can be easily upgraded, was not the main factor for the availability gain over the old architecture in our model. This was foreshadowed in the counterintuitive results in Section 6 mentioned above. The availability gain over the old architecture is based upon the addition of new hardware nodes and especially links. Single hardware or link failures can be much easier tolerated as the cluster has a higher replication degree.

We conclude this work, with a clear recommendation for the new ONOS architecture in ONOS version 1.14 and onwards.

Our work can be extended in multiple ways. One idea is to create a more realistic model, e.g. include load dependent reliability into the model or include a more sophisticated network partitioning. Additional ideas can be found in Sections 4.2 and 6.2. Here we want to remind the reader, that some of our rates, as explained in Section 4.3, are based upon older literature

[11] Calculated with $(1 - \mathit{availability}) * 365 * 24 * 60$.

or are guessed. So our results should be taken with a grain of salt. Further specific and reliable rates are needed for future work for more realistic performance analyzations.

When the model becomes more complex, a switch from GSPN to Stochastic Well-Formed nets, may help to improve the models readability and maintainability. Additionally, one could think about creating a hierarchical model like done in [14, 15] as mentioned in Section 3.

References

1. Assessing the maturity of sdn controllers with reliability growth models (Juni 2019), https://wiki.onosproject.org/download/attachments/12422167/tum-onfworkshop.pdf?version=1&modificationDate=1561629548175&api=v2
2. Avizienis, A., Laprie, J.., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1), 11–33 (2004). https://doi.org/10.1109/TDSC.2004.2
3. Ballarini, P., Bernardi, S., Donatelli, S.: Validation and evaluation of a software solution for fault tolerant distributed synchronization. In: Proceedings International Conference on Dependable Systems and Networks. pp. 773–782 (2002). https://doi.org/10.1109/DSN.2002.1029023
4. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., Parulkar, G.: Onos: Towards an open, distributed sdn os. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. p. 1–6. HotSDN '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2620728.2620744, https://doi.org/10.1145/2620728.2620744
5. Brosch, F., Buhnova, B., Koziolek, H., Reussner, R.: Reliability prediction for fault-tolerant software architectures. p. 75–84. QoSA-ISARCS '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2000259.2000274, https://doi.org/10.1145/2000259.2000274
6. Diallo, O., Sene, M., Sarr, I.: Freshness-aware metadata management: Performance evaluation with swn models. In: ACS/IEEE International Conference on Computer Systems and Applications - AICCSA 2010. pp. 1–6 (2010). https://doi.org/10.1109/AICCSA.2010.5586954
7. Gray, J., Siewiorek, D.P.: High-availability computer systems. Computer **24**(9), 39–48 (1991). https://doi.org/10.1109/2.84898
8. Kleppmann, M.: Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media (2017)
9. Kobo, H.I., Abu-Mahfouz, A.M., Hancke, G.P.: Efficient controller placement and re-election mechanism in distributed control system for software defined wireless sensor networks. Transactions on Emerging Telecommunications Technologies **30**(6), e3588 (2019). https://doi.org/10.1002/ett.3588, https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3588, e3588 ett.3588
10. Kriaa, S., Papillon, S., Jagadeesan, L., Mendiratta, V.: Better safe than sorry: Modeling reliability and security in replicated sdn controllers. In: 2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020. pp. 1–6 (2020). https://doi.org/10.1109/DRCN48652.2020.1570604424
11. Lai, R.: Performance modelling for the csma/cd protocol using gspn. In: Proceedings of IEEE Singapore International Conference on Networks and International Conference on Information Engineering '95. pp. 126–130 (1995). https://doi.org/10.1109/SICON.1995.526031
12. López-Grao, J.P., Merseguer, J., Campos, J.: On the use of formal models in software performance evaluation. Actas de las X Jornadas de Concurrencia pp. 367–387 (2002)
13. Mendiratta, V.B., Jagadeesan, L.J., Hanmer, R., Rahman, M.R.: How reliable is my software-defined network? models and failure impacts. In: 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 83–88 (2018). https://doi.org/10.1109/ISSREW.2018.00-26

14. Nencioni, G., Helvik, B.E., Gonzalez, A.J., Heegaard, P.E., Kamisinski, A.: Availability modelling of software-defined backbone networks. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W). pp. 105–112 (2016). https://doi.org/10.1109/DSN-W.2016.28
15. Nguyen, T.A., Eom, T., An, S., Park, J.S., Hong, J.B., Kim, D.S.: Availability modeling and analysis for software defined networks. In: 2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC). pp. 159–168 (2015). https://doi.org/10.1109/PRDC.2015.27
16. Onos confluence (October 2020), https://wiki.onosproject.org/display/ONOS/ONOS
17. Onos security and performance analysis (Dezember 2017), https://opennetworking.org/wp-content/uploads/2017/07/ONOS-security-and-performance-analysis-brigade-report-no1.pdf
18. Onos security and performance analysis (report no. 2) (November 2018), https://opennetworking.org/wp-content/uploads/2018/11/secperf_report_2.pdf
19. Security and performance comparison of onos and odl controllers (September 2019), https://opennetworking.org/wp-content/uploads/2019/09/ONOSvsODL-report-4.pdf
20. Owre, S., Rushby, J., Shankar, N., von Henke, F.: Formal verification for fault-tolerant architectures: prolegomena to the design of pvs. IEEE Transactions on Software Engineering **21**(2), 107–125 (1995). https://doi.org/10.1109/32.345827
21. Sakic, E., Kellerer, W.: Response time and availability study of raft consensus in distributed sdn control plane. IEEE Transactions on Network and Service Management **15**(1), 304–318 (2018). https://doi.org/10.1109/TNSM.2017.2775061
22. Sanghare, O.A., Sene, M., Rodrigues, J.J.P.C.: Distributed transactions on mobile systems: Performance evaluation using swn. In: 2011 IEEE International Conference on Communications (ICC). pp. 1–6 (2011). https://doi.org/10.1109/icc.2011.5963020
23. Scott-Hayward, S.: Design and deployment of secure, robust, and resilient sdn controllers. In: 2015 1st IEEE Conference on Network Softwarization (NetSoft). Institute of Electrical and Electronics Engineers (IEEE) (Apr 2015). https://doi.org/10.1109/NETSOFT.2015.7258233, iEEE Conference on Network Softwarization (NetSoft 2015) ; Conference date: 13-04-2015 Through 17-04-2015
24. Tanenbaum, A.S., Steen, M.v.: Verteilte Systeme - Prinzipien und Paradigmen. Pearson Studium (2008)
25. Vizarreta, P., Trivedi, K., Mendiratta, V., Kellerer, W., Mas-Machuca, C.: Dason: Dependability assessment framework for imperfect distributed sdn implementations. IEEE Transactions on Network and Service Management **17**(2), 652–667 (2020)

A Appendix

In the appendix we present additional information that may interest readers. We explain further information of the stacked failure extension in Section A.1, ONOS software failure rates to surpass the 'no upgrade' scenario from Section 6.2 in Section A.2, and the full GSPN model in Section A.3.

A.1 Stacked Failure Extension

To model stacked failures, we need to consider the dependencies the have between each other. For a start, any failure (software, hardware, network partition) can happen on top of any other, with one exception. Software failures are overwritten by a hardware failure, as a software cannot be executed on top of failed hardware. This can be seen in Figure 10. To improve readability, the failure transitions are horizontal and the recovery transitions are immediate and vertical. This has no semantic meaning.

This becomes more complex if we want to incorporate that into the presented model, we need to keep track which instance has currently which failures, to make sure it is recovered the correct way. For example a partitioned node with a software failure, must recover both before that instance is *up* again. Especially for ONOS, this would drastically increase the amount of transitions and places, as we have to model each specific failure scenario for each version.

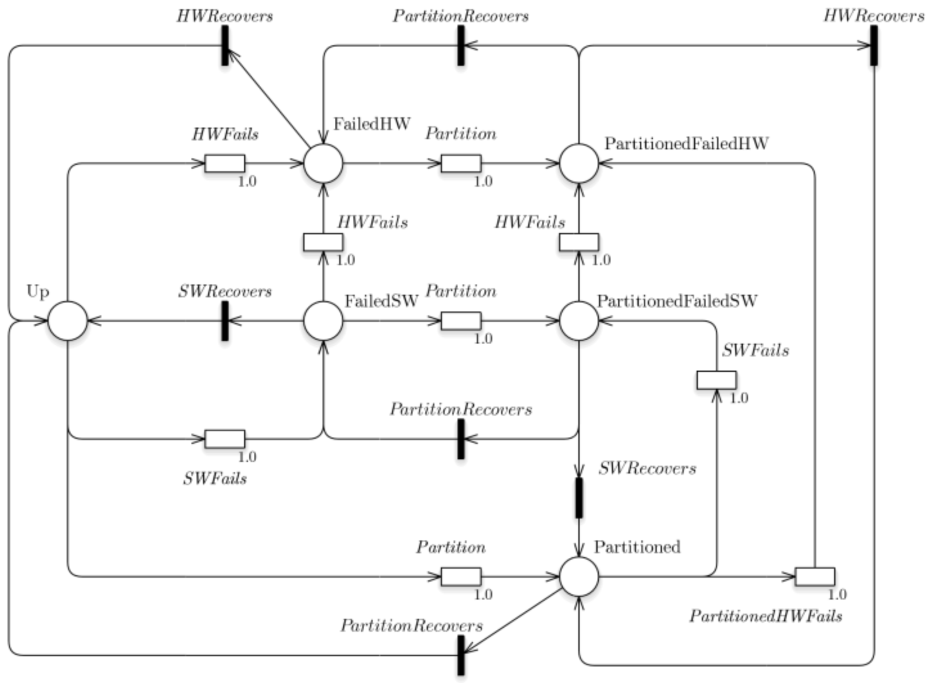

Fig. 10. Basic idea for stacked failure extension

A.2 ONOS Software failure rates to surpass 'no upgrade' scenario

With reference to Section 6.2, we now list the needed ONOS software failure rate for each version, so our model could benefit from upgrades in terms of availability. These numbers are based upon our parameters described in Section 4.3.

	Default		To surpass 'no upgrade' scenario	
Version	Rate	Decrease	Rate	Decrease
v1.14	2,1067	0%	2,1067	0%
v1.14 v2	1,8725	11,12%	1,6104	23,56%
v1.14 v3	1,6750	20,49%	1,4070	33,21%
v1.14 v4	1,5042	28,60%	1,2334	41,45%

Table 3. Decreases of the ONOS software failure rates to surpass the 'no upgrade' scenario

A.3 Full GSPN model

In Figure 11 we added the full GSPN model, so that one can better understand how the squares are connected.

Version	Bugs per hour	Relative to v1.10	Source
v1.10	0,01750	100,00%	From published work: [10, 13]
v1.11	0,01517	86,67%	Calculated: mean of v1.10 and v1.12
v1.12	0,01080	61,71%	From published work: [19]
v1.13	0,01050	60,00%	From published work: [19]
v1.14	0,00885	50,56%	Calculated: Based upon logarithmic trend from Excel: $f(x) = -0,005386747003005 * ln(x) + 0,017517342986504$
1. Upgrade	0,00787	44,95%	
2. Upgrade	0,00704	40,20%	
3. Upgrade	0,00632	36,09%	

Table 4. Bugs per hour from ONOS v1.10 to v1.14

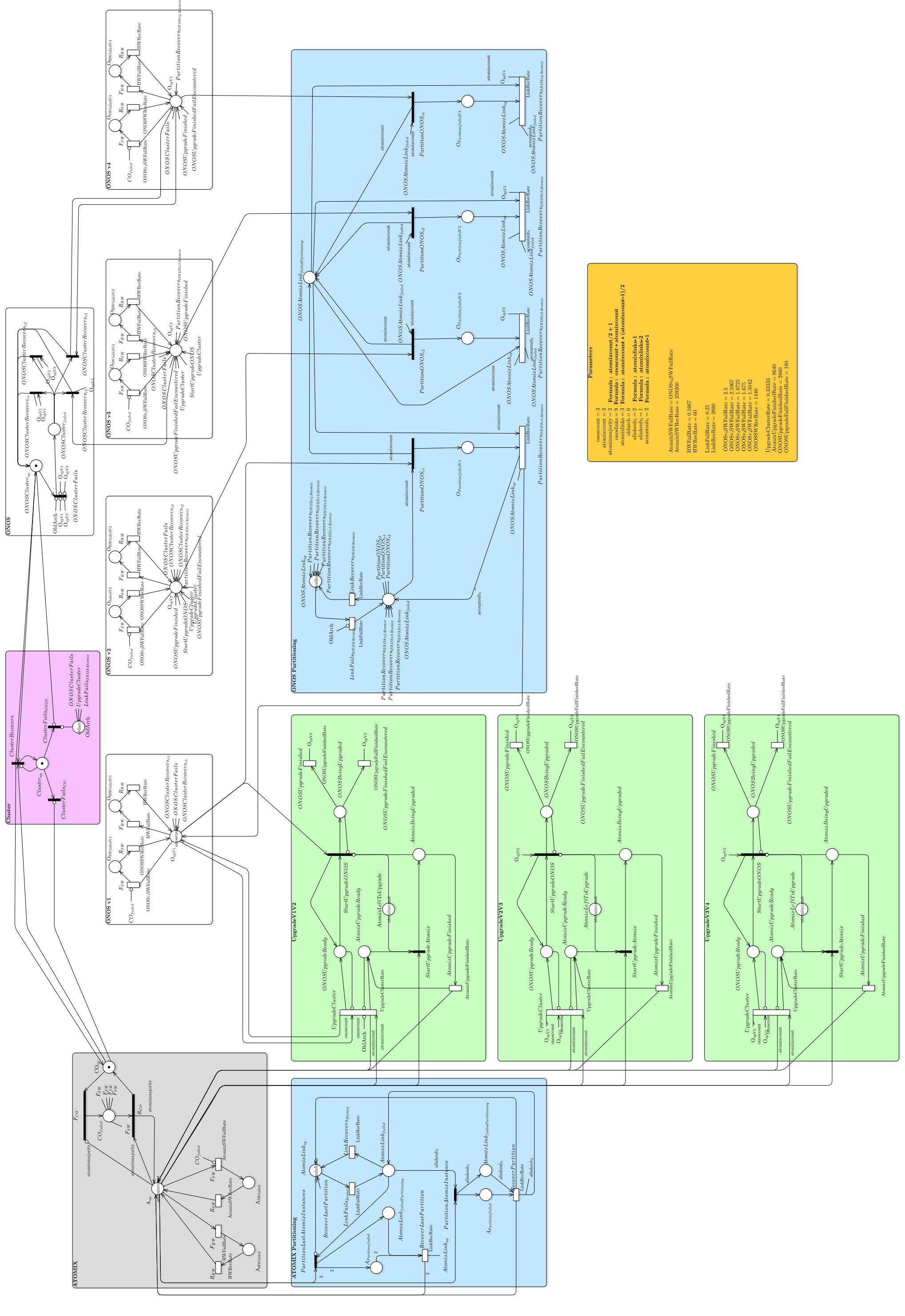

Fig. 11. Our model in GSPN and GreatSPN