An Empirical Study of Software Reliability in SDN Controllers

Petra Vizarreta*, Kishor Trivedi†, Bjarne Helvik‡, Poul Heegaard‡, Wolfgang Kellerer*, and Carmen Mas Machuca*
* Chair of Communication Networks, Technical University of Munich, Germany
† Department of Electrical and Computer Engineering, Duke University,USA
‡ Department of Telematics, Norwegian University in Science and Technology, Norway
petra.vizarreta@lkn.ei.tum.de,ktrivedi@duke.edu,{bjarne,poul.heegaard}@item.ntnu.no,{wolfgang.kellerer,cmas}@tum.de

***Abstract*—Software Defined Networking (SDN) exposes critical networking decisions, such as traffic routing or enforcement of the critical security policies, to a software entity known as the SDN controller. Controller software, as written by humans, is intrinsically prone to bugs, which may impair the network performance as a whole, if activated. Software reliability growth models (SRGM) are often used to estimate and predict the reliability of the software in the operational phase based on the fault report data during the testing phase. These models can be used to predict the number of residual bugs in the software, as well as failure intensity, software reliability and optimal software release time. In this paper we analyze ten releases of ONOS open source controller, whose uncensored fault reports are available online.**

I. Introduction

Software Defined Networking (SDN) opened a new era in networking by decoupling control and data plane, and outsourcing all control plane decisions to a logically centralized software entity known as SDN controller. The SDN controller acts as a network operating system providing an integrated interface for the forwarding devices, switches and routers, which significantly simplifies the network management and augments its programmability [1]. The controller is responsible for making routing decisions, gathering of the network state information and reacting to events, such as congestion on the links or failures of forwarding devices. Fulfilling this set of tasks requires a rather complex piece of software. Today's production-grade controllers have grown to have more than 3 million lines of code (OpenDaylight [2]). Such a large[1] and complex software inevitably contains bugs, that if activated may have a huge impact on the controller and network performance. The performance reports on SDN controllers typically include scalability and latency related metrics, such as flow burst install throughput or flow reroute latency, while reliability, which is still a big concern and a major obstacle for the wide spread adoption of SDN in commercial telecom and industrial networks [3], has been overlooked.

Software reliability growth modelling is a statistical framework, which is often used to estimate the reliability of the software components in their operational phase, based on their fault reports from the testing phase. During the testing and early operational phase of the software lifecycle the faults are detected and removed, which eventually leads to reliability growth (hence the name). In the past, many SRGM models have been proposed to estimate and predict the software reliability growth. In this paper we focus on a particular class of models that describe the fault detection and fault resolution process as Non-Homogeneous Poisson Process (NHPP), due to their widespread use in the literature. We have compared eight most frequently used SRGMs to model the fault detection process, and proposed four new ones for the modelling of the fault resolution process. The best fitting model is selected based on three Goodness of Fit (GoF) measures: Mean Square Error (MSE), coefficient of determination (R^2) and Theil's statistics (TS). Once the best model to describe the fault report data is found, it can be used to derive many useful parameters for both software developers and the users, such as residual bug content, failure intensity, and most importantly software reliability.

We have applied the SRGM framework to study reliability of ten releases of ONOS SDN controller, which is one of the biggest production grade open source controllers. ONOS was chosen because of its maturity, richness in features, wide adoption by the network operators, its focus on high-availability, and most importantly, because of the detailed and uncensored fault reports are available online [4]. We have found that NHPP models can be used to describe the fault detection and resolution process. We have analysed the residual bug content, fault intensity and software reliability, and noted that there is a consistent behaviour across the releases. We leverage this fact to improve the accuracy of reliability estimation for the latest ONOS release, for which still not enough data samples are available. In the remainder of the paper the terms software fault and bug will be used interchangeably.

The rest of the paper is organized as follows. Section II provides an overview of the related work on software reliability growth models for open source software. An overview of the NHPP class of SRGM models is presented in Section III. In Section IV the gathering, processing and analysis of the ONOS fault reports is discussed. The results of the software reliability modelling and evaluation are presented in Section V. We conclude the paper with a summary and an outlook for the future work.

[1]As a reference, the latest Linux kernel has around 20 million lines of code.

978-3-901882-98-2 © 2017 IFIP

II. Related work

Software reliability growth modelling has been widely used to estimate and predict the reliability of the software, and in the past, many different models have been proposed, out of which the class of Non-Homogeneous Poisson Process (NHPP) has received the most attention. A good overview of different classes of reliability growth models, together with their inherent assumptions and input data requirements, can be found in [7].

The applicability of SRGMs for the modeling, analysis and evaluation of software reliability of open source products was demonstrated in several case studies. Zhou et al. [12] showed that the Weibull distribution can describe well the bug manifestation rate for eight unnamed open source projects. Rahmani et al. [13] confirmed this result by analyzing the bug reports for several popular big open source projects, such as Apache HTTP server, Eclipse and Firefox. Rossi et al. [14] studied failure occurrence pattern across several releases of Mozilla Firefox, OpenSuse and OpenOffice.org. All studied releases have shown the learning curve pattern, where the fault detection rate is slow at the beginning until the community gets familiar with the product, then it increases rapidly until only very few faults remain whose discovery is rare. This effect can be captured well with S-shaped models. In this work we have compared eight most widely used software reliability growth models in terms of its capability to describe the fault detection process.

The most of the SRGM models in the literature assume that once the bug is detected, it is corrected immediately. However, Ullah et al. [15] showed that existing NHPP models cannot describe well the fault resolution process. Gokhale et al. [16] applied the Non-Homogeneous Continuous Time Markov Chains (NH-CTMC) to model the impact of arbitrary debugging policy. Wu et al. [17] described the fault resolution as a delayed fault detection process, while Kapur et al. [18] generalized this result and proposed unified approach to model the fault resolution process, when both fault detection and fault removal are Non-Homogeneous Poison Processes. We propose a new class of models, based on the framework in [18], with their corresponding fitting procedure.

Fitting of SRGMs to fault report data was done either with proprietary general purpose statistical packages, such as SPSS, or specialized tools, such as CASRE [19] and SREPT [20]. In order to account for the newly proposed models, we have developed our own tool based on the libraries provided by the Python scientific computing package [21].

III. Software reliability growth models

During the testing and early operational phase of the software lifecycle the faults are detected and removed, which eventually leads to reliability growth. The class of models that describe the fault detection process as Non-Homogeneous Poisson Process (NHPP) have widely been used in the literature.

A. Fault detection process as NHPP

NHPP is similar to the ordinary Poisson process, except for the fact that the arrival rate changes over time [7]. The probability of observing n faults by the time t is described with the equation Eq.(1).

$$P(N(t) = n) = \frac{m(t)^n}{n!} e^{-m(t)} \tag{1}$$

Mean value function $m(t)$ represents the expected number of detected faults in the time interval $(0, t]$:

$$E[N(t)] = m(t) = \int_0^t \lambda(x)dx \tag{2}$$

From the mean value function of the fault detection process many reliability features of the software can be predicted. The expected time between successive fault manifestations at a given time t is defined as:

$$TBF(t) = \frac{1}{\lambda(t)} \tag{3}$$

Assuming that the number of initially introduced faults in the software is finite $\lim_{t\to\infty} m(t) = a$, the expected number of the undetected faults in the software can be defined as:

$$r(t) = E[a - N(t)] = a - m(t) \tag{4}$$

Software reliability is defined as the probability that software will not cause the failure for a specified time under specified conditions. The conditional software reliability is defined as the probability of detecting a new fault in the time interval $(t, t + x]$:

$$R(x, t) = e^{-\int_t^{t+x} \lambda(x)dx} = e^{m(t)-m(x+t)} \tag{5}$$

Model	Abbreviation	Shape	Mean value function	Failure intensity
Musa-Okumoto logarithmic [5]	MUSA(Log)	Concave	$m_{mo}(t) = a\ln(1 + bt)$	$\lambda_{log}(t) = \frac{ab}{1+bt}$
Goel-Okumoto exponential [6]	GO(Exp)	Concave	$m_{go}(t) = a(1 - e^{-bt})$	$\lambda_{go}(t) = abe^{-bt}$
Generalized Goel-Okumoto [7]	GGO	S-shaped	$m_{ggo}(t) = a(1 - e^{-bt^c})$	$\lambda_{ggo}(t) = abct^{c-1}e^{-bt^c}$
Ohba's inflection S-shaped [8]	ISS	S-shaped	$m_{iss}(t) = a\frac{1-e^{-bt}}{1+\phi e^{-bt}}$	$\lambda_{iss}(t) = abe^{-bt}\frac{1+\phi}{(1+\phi e^{-bt})^2}$
Yamada delayed S-shaped [9]	DSS	S-shaped	$m_{dss}(t) = a(1 - (1 + bt)e^{-bt})$	$\lambda_{dss}(t) = ab^2te^{-bt}$
Yamada exponential [10]	YEX	Concave	$m_{yex}(t) = a(1 - e^{-r(1-e^{-bt})})$	$\lambda_{yex}(t) = abre^{-bt}e^{-r(1-e^{-bt})}$
Gompertz [7], [9], [11]	GOMP	S-shaped	$m_{gomp}(t) = ak^{b^t}$	$\lambda_{gomp}(t) = a\ln b\ln kb^tk^{b^t}$
Logistic [7], [9], [11]	LOGIST	S-shaped	$m_{logist}(t) = \frac{a}{1+ke^{-bt}}$	$\lambda_{logist}(t) = \frac{abke^{-bt}}{(1+ke^{-bt})^2}$

TABLE I: Fault detection process as Non-Homogeneous Poisson Process (NHPP)

We have compared eight most widely used NHPP models for modelling of the fault detection process: Musa-Logarithmic, Goel-Okumoto Exponential, Generalized Goel-Okumoto, Inflection S-shaped, Delayed S-Shaped, Yamada-Exponential, Gompertz and Logistic, whose mean value function and failure intensity are given in the Table I.

All considered models are finite models, assuming that the number of initially introduced faults in the software is a. The mean value function an instantaneous fault intensity $\lambda(t)$ can be written as:

$$m(t) = aF_D(t) \tag{6}$$

$$\lambda(t) = aF_D^{'}(t) = af_D(t) \tag{7}$$

where $F_D(t)$ represents per fault failure detection time distribution.

B. Fault resolution process

The fault resolution process consists of two phases, fault detection and fault correction. Assuming that the two processes are independent, the resulting fault resolution process is also a NHPP process. The time to resolve a fault is the sum of the times to detect and to correct the fault. Hence, the fault density of the time to resolve a fault is a convolution of the fault detection and fault correction densities [17]:

$$f_R(t) = \int_0^t f_D(t-x) f_C(x) dx = [f_D * f_C](t) \tag{8}$$

where $f_D(t)$ and $f_C(t)$ represent fault detection and fault correction densities. The mean value function of the resulting fault resolution process can be written as:

$$m_R(t) = aF_R(t) = a\int_0^t [f_D * f_C](x) dx \tag{9}$$

Equation Eq. (9) can be used to generate different SRGMs for fault resolution process. However, the proposed models so far have been limited to combinations of NHPP models for which this integral has a closed form solution, e.g. when both fault detection and correction are Goel-Okumoto processes [17], [18].

$$m_R^{go-go}(t) = a\left[1 - \frac{b_1 e^{-b_2 t} - b_2 e^{-b_1 t}}{b_1 - b_2}\right] \tag{10}$$

By replacing the integral in Eq. (9) with its Piecewise Constant Approximation (PCA), we can obtain a numerical approximation for an arbitrary combination of NHPP models, which can be used for the fitting of the fault report data.

$$\widetilde{F}_R(t) = \sum_{i=0}^{n=t/\Delta x} [f_D * f_C](i\Delta x)\Delta x \tag{11}$$

$$F_R(t) = \lim_{\Delta x \to 0} \widetilde{F}_R(t) \tag{12}$$

Due to the space limitation, in this paper we compare the four combinations of Generalized Goel-Okumoto and Inflection S-shaped models for fault resolution process, using combined Goel-Okumoto Eq.(12) from [18] as a reference.

C. Model fitting

Non-linear regression based on the least squares Levenberg-Marquardt method provided by Python scientific computing package [21], was used to fit the parameters of the proposed models to the actual data for the cumulative number of observed number of detected and resolved bugs.

The best fitting model among all proposed ones is selected based on Goodness of Fit (GoF) measures: Mean Square Error (MSE), coefficient of determination (R^2) and Theil's statistics (TS), which are defined as follows:

$$MSE = \frac{1}{k}\sum_{i=1}^{k} (m(t_i) - m_{est}(t_i))^2 \tag{13}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{k} (m(t_i) - m_{est}(t_i))^2}{\sum_{i=1}^{k} (m(t_i) - \overline{m})^2} \tag{14}$$

$$\overline{m} = \frac{1}{k}\sum_{i=1}^{k} m(t_i)$$

$$TS = \sqrt{\frac{\sum_{i=1}^{k} (m(t_i) - m_{est}(t_i))^2}{\sum_{i=1}^{k} m(t_i)^2}} * 100\% \tag{15}$$

where $m(t_i)$ and $m_{est}(t_i)$ represent the actual and fitted data at time t_i of the i-th sample.

IV. ONOS DATA SETS

Open Network Operating System (ONOS) [4] is one of the two major open source SDN controllers, that offers scalability, high availability and production-grade performance. Over 300 developers from more than 60 organizations have contributed to its code base. The code is written mostly in Java and contains at the moment 665,730 lines of code[2].

New releases are published quarterly, and are named after the birds in alphabetical order. Eleven releases have been published since December 2014. The latest release, Kingsfisher, has been published recently, on June 5, 2017, and we do not consider it in our analysis. The release cycle starts with the release planning during the first week, followed by three months of code development and integration on the master branch. Two weeks before the official release date feature integration is stopped and only bug fixes are allowed. Issues related to every release are reported in the JIRA tracking system, which is available online. For the purpose of our analysis we are interested in the issues labelled as "Bugs" rather than new feature requests or enhancements.

The bug reports in the issue tracker contain the fault type, priority, affected versions, bug description and short summary, date of the report creation, date of its resolution (if applicable). The issue tracker contains all data necessary to conduct the software reliability analysis. The cumulative number of detected and resolved faults reported over time are shown in Fig 1. The official dates of ONOS releases are indicated with

[2]Source:https://www.openhub.net/p/onos

the vertical line in the figure. It can be observed from the figure that there is a steady increase in the number of bugs over the course of the past 32 months[3].

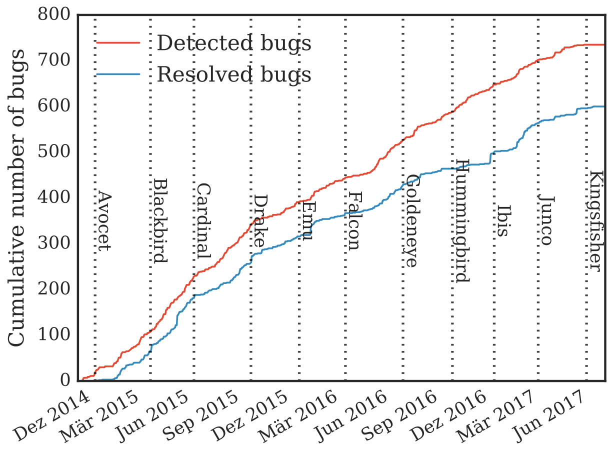


Fig. 1: Number of detected and resolved faults over time. The official dates of ONOS releases are indicted with the vertical line in the figure.

We have filtered and separated the fault reports based on the ”affected release version” field. The number of the bugs reported for every release, sorted by the priority, are presented in Fig. 2. Note that due to the time overlap between the support periods some of the fault reports affected more than one release. Since trivial and minor issues (e.g. loading of the GUI too slow) do not have direct impact on the software reliability, we have used only bug entries labelled with major, critical and blocker priority in our reliability growth analysis.

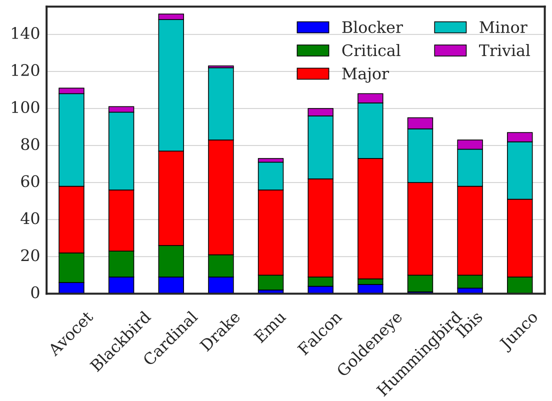


Fig. 2: Number of faults reported for each release, by priority.

In order to investigate if there is a consistent behaviour across the releases, we have performed a trend analysis. Due to the space limitations we show only the distribution of the time between faults in Fig. 3.

[3]Data retrieved on June 12, 2017 from JIRA dashboard: https://jira.onosproject.org/

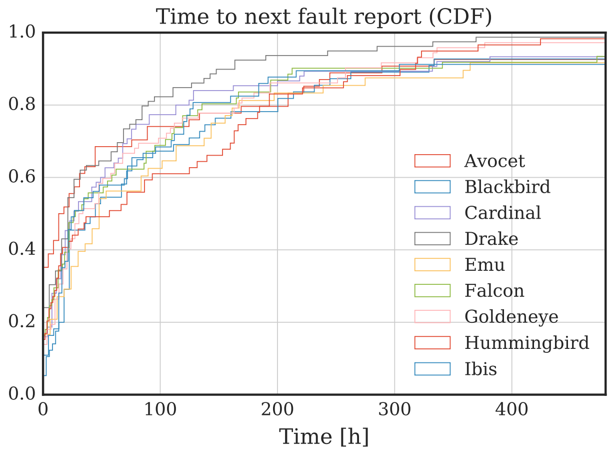


Fig. 3: Distribution of the time between the successive faults reports for the first nine ONOS releases.

V. Results

We have analysed the data sets of ten ONOS releases. For the first nine releases we have compared GoF metrics of SRGM models described in Sections III-A and III-B, and found the best one to model fault detection and fault resolution process. After finding the parameters of the best fitting models, we have estimated software reliability features, such as residual bug content, failure intensity and conditional software reliability. Based on the parameters of the best models obtained for the first nine releases, we propose the fitting method to improve the fitting accuracy for the tenth release.

A. Finding the best model

1) Fault detection process: We have compared GoF metrics for eight SRGM models presented in the Table I. The comparison of different models for the Emu data set is illustrated in Fig. 4.

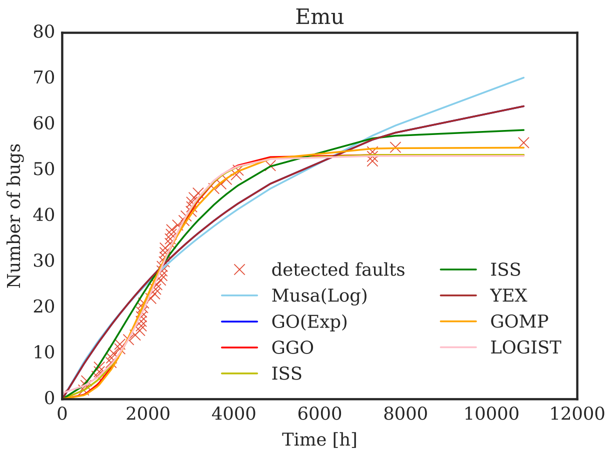


Fig. 4: Comparison of eight SRGM models for fault detection process for Emu data set.

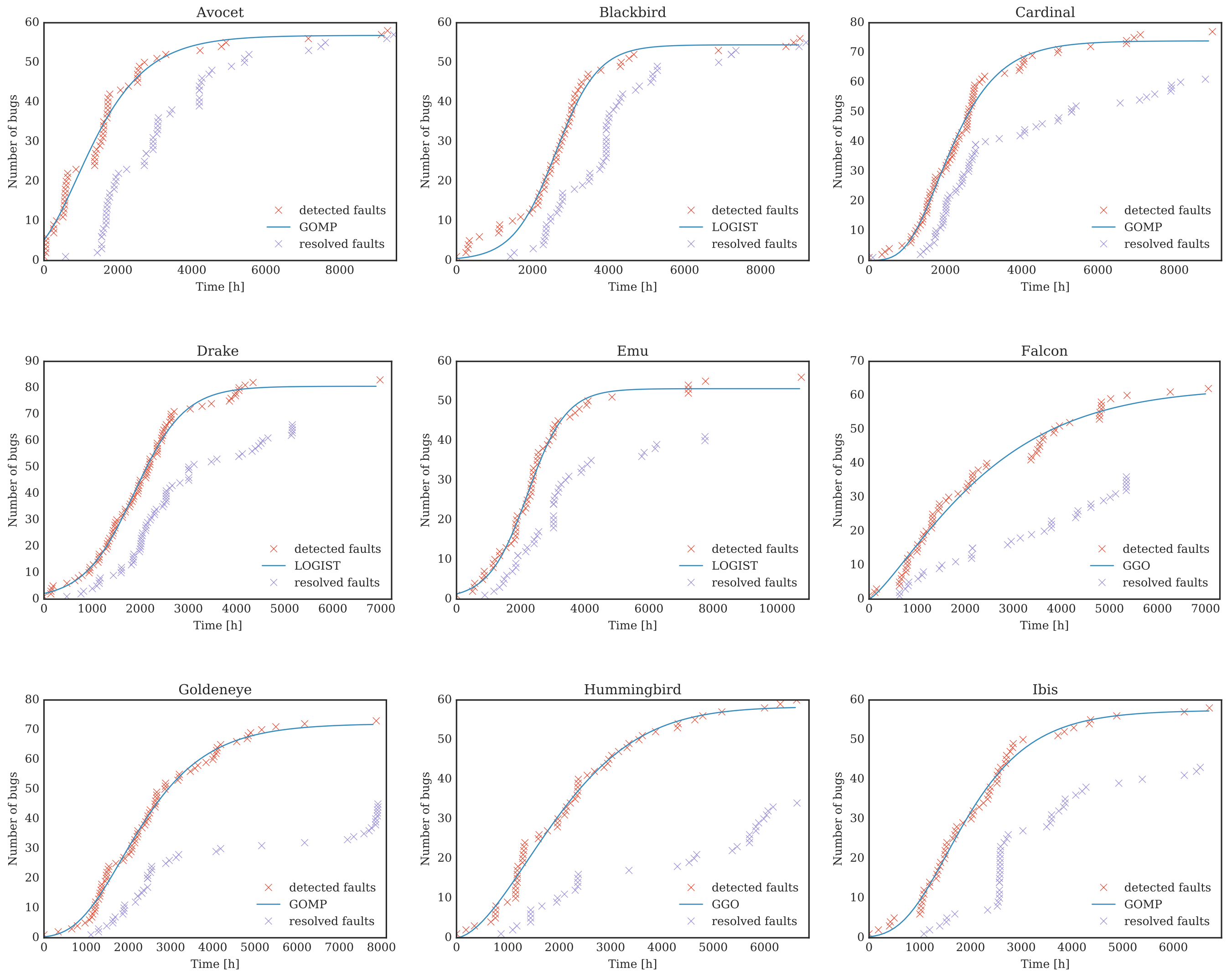
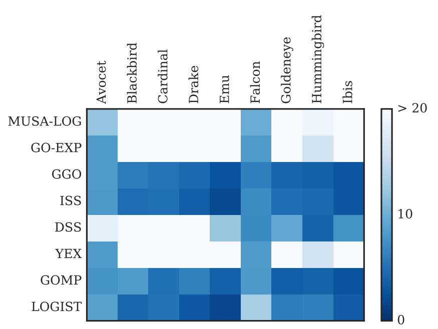
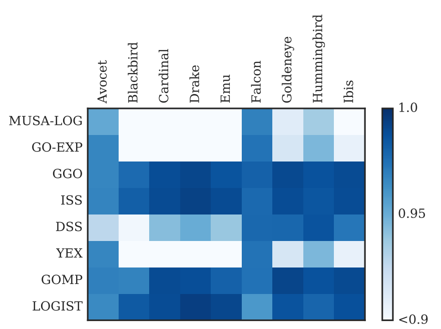


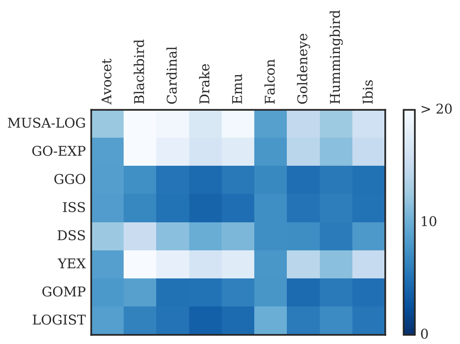
Fig. 5: Comparison of the best fitting models for fault detection process for all ONOS releases. For most of the releases the best fitting model is Gompertz (4 out of 9 releases), followed by Logistic (3) and Generalized Goel-Okumoto (2). It has to be noted that Inflection S-shaped model also shows very good GoF results, being the second best fit for most of the releases.



(a) Mean Square Error (MSE)



(b) Coefficient of determination (R^2)



(c) Theil's Statistics (TS)

Fig. 6: All GoF indicators show consistent results: the best model to describe the number of detected faults across all releases are S-shaped models: Gompertz, Logistic, Generalized Goel-Okumoto and Inflection S-shaped, while concave models Musa-Logaritmic, Goel-Okumoto Exponential and Yamada Exponential could not explain the data.

The best fitting models for fault detection process for all ONOS releases are presented in Fig. 5. It can be seen that for most of the releases the best fitting model is Gompertz (4 out of 9 releases), followed by Logistic (3) and Generalized Goel-Okumoto (2). It has to be noted that the Inflection S-shaped model also shows very good GoF results, being the second best fit for most of the releases.

GoF measures for all the models are compared in the Fig. 6. All GoF indicators show consistent results: the best model to describe the number of detected faults across all releases are S-shaped models Gompertz, Logistic, Generalized Goel-Okumoto and Inflection S-shaped, while concave models Musa-Logaritmic, Goel-Okumoto Exponential and Yamada Exponential could not explain the data. Delay S-shaped shows slightly worse results, compared to the other S-shaped models. This effect is probably due to the fact that this model has only two parameters to tune, while the other S-shaped models have three parameters.

2) Fault resolution process: Arbitrary combination of NHPP models can be used for fitting of the cumulative number of resolved bugs applying the Eq.(9). Based on the results from the previous section, we have selected combinations of S-shaped models: Generalized Goel-Okumoto (GGO) and Inflection S-shaped (ISS). The models are abbreviated as a combination of the initials of detection and resolution NHPP processes. For the sake of comparison we also include the model from [18] where both fault detection and resolution are modeled as Goel-Okumoto processes.

The best fitting model for three releases, Avocet, Blackbird and Ibis, are illustrated in Fig. 7. The best fitting models are GGO-ISS and ISS-GGO. Note that these two combinations result essentially in the same fault resolution process, since convolution is a symmetrical operation, but they assume a different fault detection process. ISS-GGO and GGO-ISS outperformed the reference GO-GO model, for all the releases, where fitting was possible.

It can be seen that although the proposed models for the fault resolution process could describe the data for some of the releases, the actual data shows higher deviation from the fitted model, then in the previous case. The same pattern can be observed also in the Fig. 8, where the fitting capabilities of the five proposed models for all releases are compared.

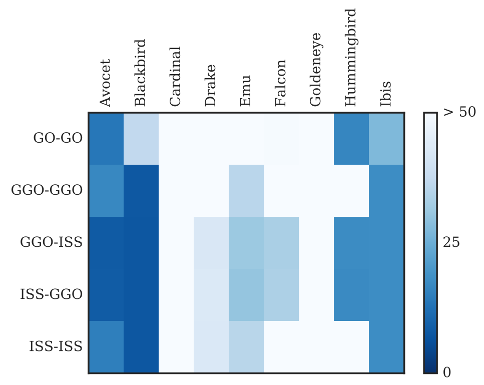


Fig. 8: Comparison of MSE of SRGM models for the fault resolution process: ISS-GGO and GGO-ISS outperformed the reference GO-GO model, for all the releases, for which the fitting was possible.

Cumulative number of resolved bugs for all ONOS releases can be seen in Fig. 5 for reference. It can be seen from the figure that cumulative number of resolved bugs experience the sudden trend changes, which cannot be captured by the simple NHPP models. The trend changes are probably due to the changes in the debugging effort shortly before the new upcoming release (or the release of the patches) and can be modelled by introducing the (time) change points in the underlying NHPP models, as in [22], or using the more complex models accounting for the test effort, such as [23]. The available ONOS data set does not provide the precise information about the debugging effort, that could be used for such models.

Since the accurate modelling of the fault resolution process requires more information and more complex models, we focus on the software reliability metric that can be derived from fault detection SRGMs.

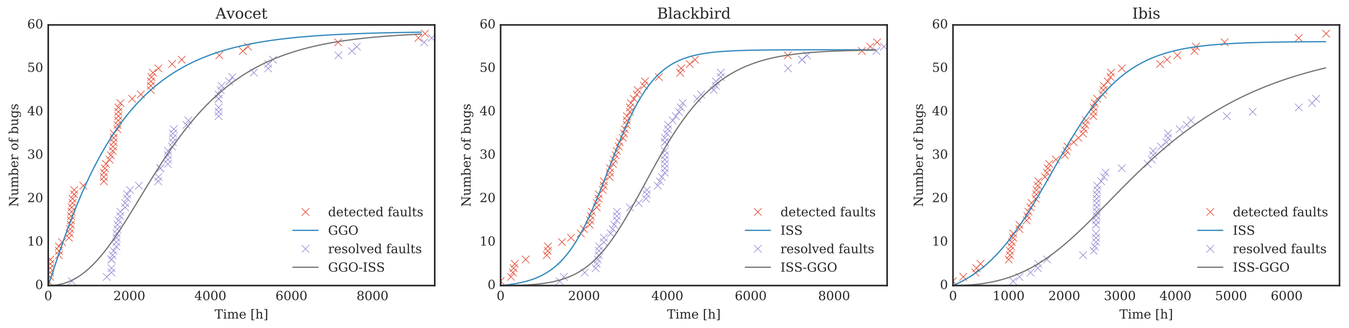


Fig. 7: Comparison of the best fitting models for fault resolution process: the proposed models could describe the data for some of the releases, although the actual data shows higher deviation from the fitted model due to the sudden trend changes that require more complex model and more information about debugging effort.

B. Evaluation of the software reliability

Once the best model to describe the fault report data is selected and the parameters of its mean value function have been estimated, it can be used to estimate and predict several software reliability parameters: residual bug content, instantaneous fault intensity and conditional software reliability, as defined in section III-A by Eq.(3)-(5).

The software reliability metric for the Falcon release are presented in Fig. 9 for illustration. The official release date is indicated with the vertical line in the figure, and the time is expressed as the relative time since the start of the testing. Note that only severe bugs (bugs with major, critical and blocker priority) are included in the study.

Residual bug content represents the number of undetected faults remaining in the software, as defined in Eq.(4). It can be seen that the residual bug content was relatively high, as 28 severe bugs were still remaining in the software on the day of its official release.

Instantaneous fault intensity, or alternately expected mean time between successive faults, can be derived from the parameters of the mean value function, as defined in Eq.(2) and Eq.(3). The instantaneous fault intensity on the day of Falcon's release was at the level of $0.0131h^{-1}$, or equivalent to approximately 3 days between detection of severe faults. The fault intensity is highly relevant for the software developers, as it can indicate when is the software ready for the release. Had this been a commercial software, a penalty for the severe software bugs detected in the operational software would be paid. This metric could help the developers optimize the testing effort before and after official release date.

Conditional software reliability represents the probability of encountering a severe failure in the time interval $(t, t+x)$, as defined in Eq.(5). We observe the interval starting with the software adoption time t for a duration x, specified by the user. We have shown that in order to achieve reliability of $R(x,t) = 0.90$, during maintenance interval of $x = 3$ months, the user should defer the software adoption up to $\Delta t \geq 12$ months after its official release.

C. Predictive ability

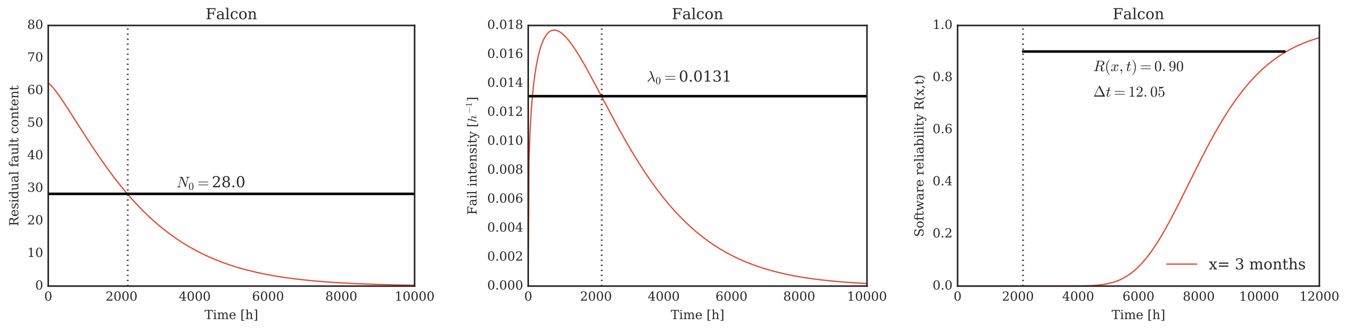
Estimating the parameters of the SRGM models when only few data samples are available may lead to inaccurate model. From the previous analysis we have seen that Gompertz model had the best performance across all releases, being the best fit for 4 out of nine releases, and showing very good results for the other five. Moreover, we have noted that the parameters across different releases of ONOS tend to be very similar, as it can be seen in Table II for Gompertz model. We leverage this fact to guide the fitting procedure and improve the accuracy of the model parameters for Junco release, for which still not enough data samples are available.

Release	a	b	k
Avocet	56.81	0.99903	0.09119
Blackbird	55.59	0.99877	3.12e-8
Cardinal	73.90	0.99884	0.00025
Drake	84.31	0.99879	0.00106
Emu	54.92	0.99896	0.00091
Falcon	57.80	0.99917	0.04774
Goldeneye	72.07	0.99908	0.00425
Hummingbird	58.61	0.99908	0.01841
Ibis	57.42	0.99889	0.00420

TABLE II: Fitted parameters of Gompertz model

It can be seen from the table that the parameters of the Gompertz model do not vary much, especially between the consecutive releases. The parameter a varied between 54 and 85; parameter b was in the range 0.99879 and 0.99917, showing the least variance, while parameter k was in the range 3.12e-8 to 0.04774. We have used these values to bound the parameters during the fitting procedure, as $p_{low} = p_{min} - 10\%$, and $p_{high} = p_{max} + 10\%$, where p stands for the parameters of the Gompertz model a, b and k. We leave it for the future work to improve the prediction, based on the observed variance and trend analysis of each of the parameters.

Prediction of the number of software defects for the period of one year after its release is presented in Fig. 10.



(a) Residual bug content $r(t)$

(b) Failure intensity $\lambda(t)$

(c) Software reliability $R(x,t)$

Fig. 9: SRGM can be used to estimate the quality of the controller software, in terms of residual bug content, failure intensity and software reliability. These software quality indicators can be used by the software developers, to estimate the optimal release time, and by the network operators, to estimate the optimal controller software adoption time.

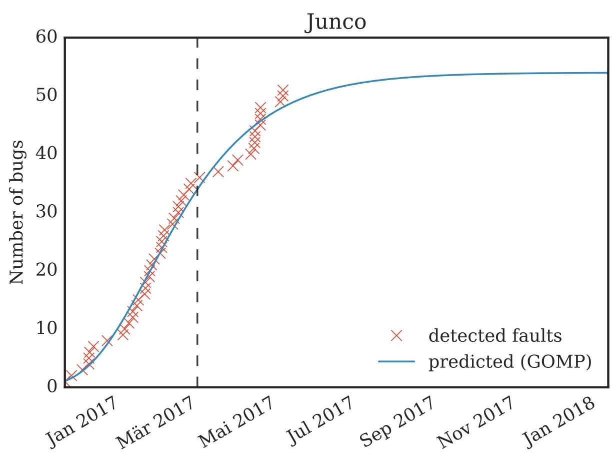


Fig. 10: Prediction of the number of software defects for Junco release for the period of one year after its release.

D. *Limitations of the proposed approach*

There are few limitations of the proposed approach. The first limitation comes from the fault reports, as the results are only as good as the accuracy of the data sets. While doing the data mining we noticed that some of the entries were not complete (e.g. affected version field was empty) and that several entries had a creation date in the future. SRGM models require the uncensored fault reports, meaning that if some faults were not reported the estimated parameters will not be as accurate. Since we cannot fully guarantee the accuracy nor the completeness of the reported data in the ONOS issue tracker, we do not emphasize the numerical results, but rather focus on the general approach to quantify the software reliability.

The second limitation comes from SRGM models. The models assume independent times between the consecutive fault reports, which is not entirely true since occasionally several related bug were reported at the same time. The models also assume that every undetected fault contributes the same to the fault manifestation rate. The time in our study represents the calendar time. It would be more accurate to consider the CPU time and the actual test effort (men-hours), but this information is not available in such a large open source project.

VI. Conclusion

In this paper, we have studied the reliability of ONOS, a production-grade SDN controller, whose detailed fault reports from the test and operational phase are available online. From these fault reports we have found fairily consistent behaviour across the releases, in terms of number of bugs, fault detection and resolution time.

We have shown that the fault detection process can be described with Non-Homogeneous Poisson Process (NHPP) class of Software Reliability Growth Models (SRGM). We have found that the best fitting model across all releases were S-shaped models, Gompertz, Logistic, Generalized Goel-Okumoto and ISS, while concave models Musa-Logaritmic, Goel-Okumoto Exponential and Yamada Exponential could not explain the data.

A new class of models to describe the number of resolved bugs over time, as well as their corresponding fitting technique, is proposed. We have shown that although the proposed models for the fault resolution process could describe the data in some cases, data shows higher deviation and sudden trend changes that cannot be captured by simple NHPP models. In general more complex models accounting for debugging effort are needed. For the releases where fitting was reasonable, the proposed models GGO-ISS and ISS-GGO outperformed the reference GO-GO model.

Once the best model is selected and parametrized to fit the fault report data, it can be used to estimate many reliability-related parameters, such as residual bug content, failure intensity and software reliability. We have illustrated in the example how these metrics can help the developers to tune better the software release cycles, and optimize the test effort during different phases of software lifecycle, as well as to assist the users to take the calculated risk and chose the best software adoption time, based on the reliability requirements of their network applications.

We have observed that the parameters across different releases of ONOS tend to be similar, and leverage this fact to guide the fitting procedure and improve the accuracy of the model parameters for last release, for which still not enough data samples are available. We leave it for the future work to study the trend of the parameters and improve the accuracy of our forecasting algorithm.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671648 (VirtuWind), COST Action CA15127 Resilient communication services protecting end-user applications from disaster-based failures (RECODIS) and was supported in part by US NSF Grant number CNS-1523994, CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1) and the German BMBF (Project ID 16KIS0473).

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[2] Linux Foundation, "Opendaylight." [Online]. Available: https://www.opendaylight.org/

[3] J. Vestin, A. Kassler, and J. Akerberg, "Resilient software defined networking for industrial control networks," in *2015 10th International Conference on Information, Communications and Signal Processing (ICICS)*. IEEE, 2015, pp. 1–5.

[4] ON.Lab, "ONOS: Open Neetwork Operating System," http://onosproject.org/, 2017.

[5] J. D. Musa and K. Okumoto, "A logarithmic poisson execution time model for software reliability measurement," in *Proceedings of the 7th international conference on Software engineering*. IEEE Press, 1984, pp. 230–238.

[6] A. Goel and K. Okumoto, "Time-dependent errordetection rate model for software and other performance measure," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, p. 285, 1985.

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

[8] M. Ohba, “Software reliability analysis models,” *IBM Journal of research and Development*, vol. 28, no. 4, pp. 428–443, 1984.

[9] S. Yamada, M. Ohba, and S. Osaki, “S-shaped reliability growth modeling for software error detection,” *IEEE Transactions on reliability*, vol. 32, no. 5, pp. 475–484, 1983.

[10] S. Yamada, H. Ohtera, and H. Narihisa, “Software reliability growth models with testing-effort,” *IEEE Transactions on Reliability*, vol. 35, no. 1, pp. 19–23, 1986.

[11] C.-Y. Huang, M. R. Lyu, and S.-Y. Kuo, “A unified scheme of some nonhomogenous poisson process models for software reliability estimation,” *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 261–269, 2003.

[12] Y. Zhou and J. Davis, “Open source software reliability model: an empirical approach,” in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–6.

[13] C. Rahmani, H. Siy, and A. Azadmanesh, “An experimental analysis of open source software reliability,” *Department of Defense/Air Force Office of Scientific Research*, 2009.

[14] B. Rossi, B. Russo, and G. Succi, “Modelling failures occurrences of open source software with reliability growth,” in *IFIP International Conference on Open Source Systems*. Springer, 2010, pp. 268–280.

[15] N. Ullah and M. Morisio, “An empirical analysis of open source software defects data through software reliability growth models,” in *EUROCON, 2013 IEEE*. IEEE, 2013, pp. 460–466.

[16] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi, “Analysis of software fault removal policies using a non-homogeneous continuous time markov chain,” *Software Quality Journal*, vol. 12, no. 3, pp. 211–230, 2004.

[17] Y. Wu, Q. Hu, M. Xie, and S. H. Ng, “Modeling and analysis of software fault detection and correction process by considering time dependency,” *IEEE Transactions on Reliability*, vol. 56, no. 4, pp. 629–642, 2007.

[18] P. Kapur, H. Pham, S. Anand, and K. Yadav, “A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation,” *IEEE Transactions on Reliability*, vol. 60, no. 1, pp. 331–340, 2011.

[19] M. R. Lyu and A. Nikora, “CASRE: a computer-aided software reliability estimation tool,” in *Computer-Aided Software Engineering, 1992. Proceedings., Fifth International Workshop on*. IEEE, 1992, pp. 264–275.

[20] S. Ramani, S. S. Gokhale, and K. S. Trivedi, “SREPT: software reliability estimation and prediction tool,” *Performance evaluation*, vol. 39, no. 1, pp. 37–60, 2000.

[21] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–. [Online]. Available: http://www.scipy.org/

[22] C.-Y. Huang, T.-Y. Hung, and C.-J. Hsu, “Software reliability prediction and analysis using queueing models with multiple change-points,” in *Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009. Third IEEE International Conference on*. IEEE, 2009, pp. 212–221.

[23] H. Pham and X. Zhang, “Nhpp software reliability and cost models with testing coverage,” *European Journal of Operational Research*, vol. 145, no. 2, pp. 443–454, 2003.