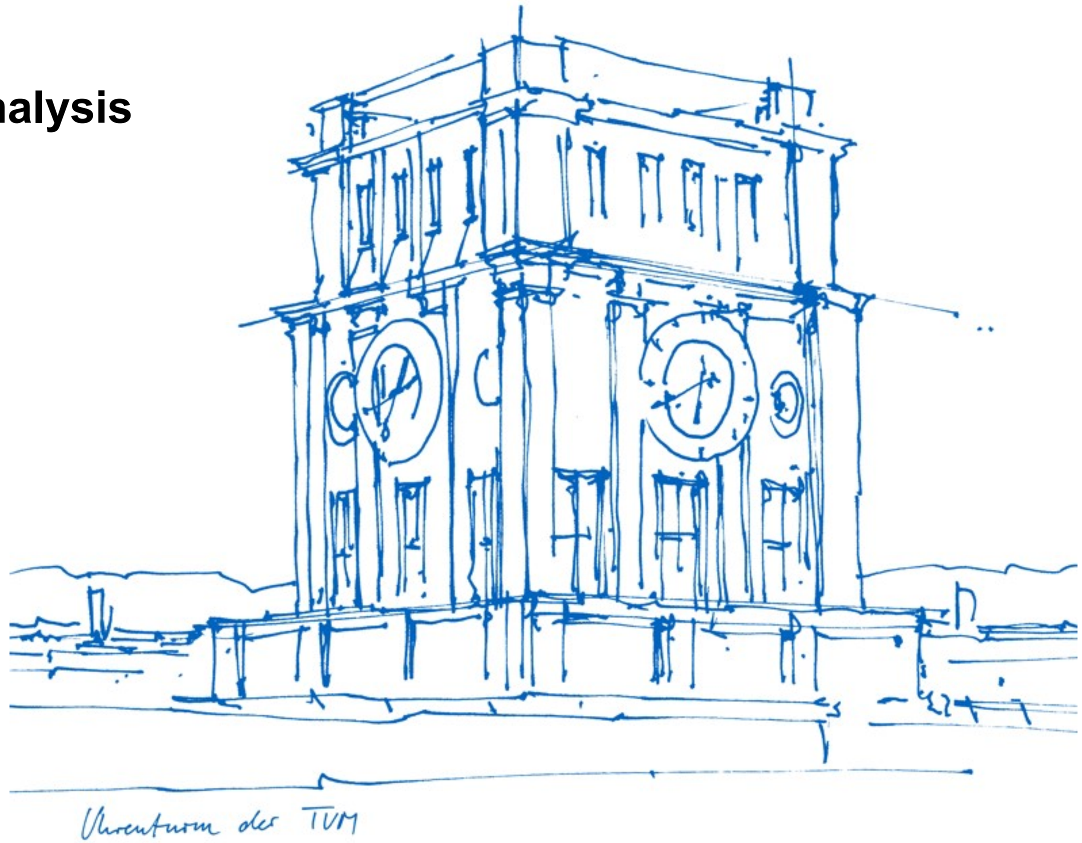
Chair of Communication Networks
Department of Electrical and Computer Engineering
Technical University of Munich

Assessing the Maturity of SDN Controllers with Reliability Growth models

Network Traffic Measurement and Analysis Conference (TMA 2019)

Workshop of the ONOS security and performance analysis brigade

June 17, 2019



Authors:

Petra Vizarreta Paz

Carmen Mas Machuca

Outline

- Motivating examples
  - Complexity of controllers in Software Defined Networking (SDN)
  - Ubiquity and magnitude of software failures → operational SDN networks
  - Open source orchestration platforms in SDN – empirical reliability study
  - The only constant is change! – Fast pace of network control software evolution

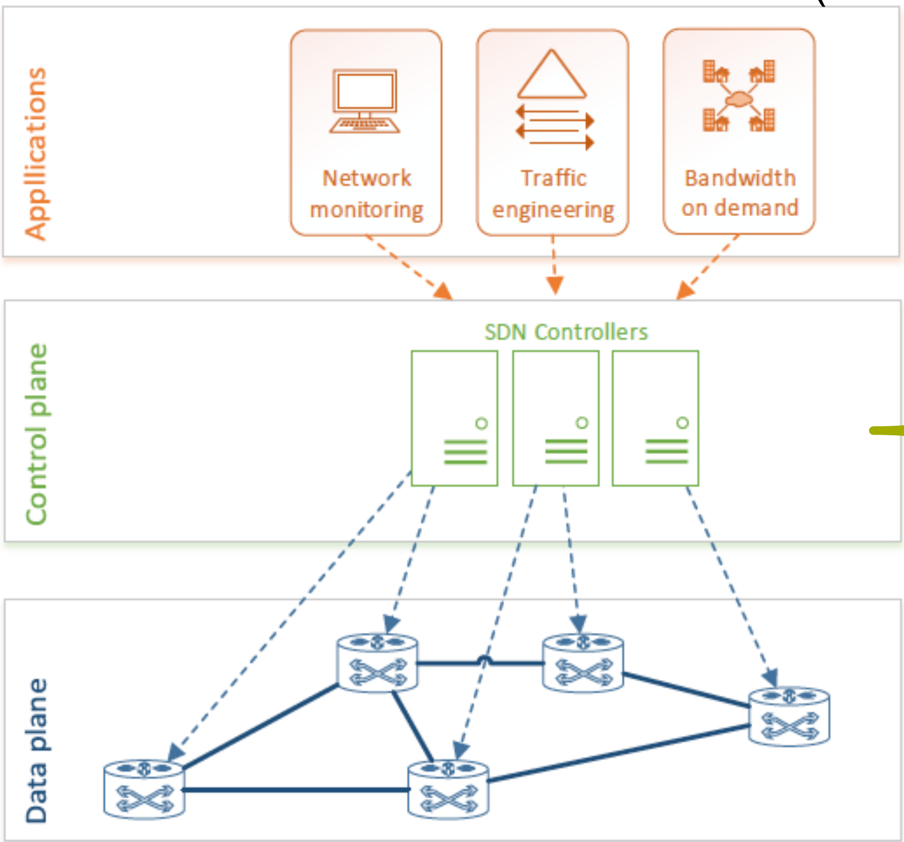
- Assessing Software Maturity with Reliability Growth Models
  - Software Reliability Growth Models (SRGM)
  - Evaluating and forecasting the software reliability metrics
  - Management KPIs: optimal software release time and software maturity metrics

- Discussion and further steps
  - Limitations of existing SRGMs: early prediction of software maturity
  - Per-project evaluation of software maturity
  - The role of machine learning in Software Reliability Engineering

Complexity of SDN controller software

Commercial controllers have more than 3 million lines of code [Odl2017]

ARCHITECTURAL CONCEPT OF SOFTWARE DEFINED NETWORKING (SDN)



The role of SDN controller

1) Implement network application intents
   - traffic steering, bandwidth calendaring, ...

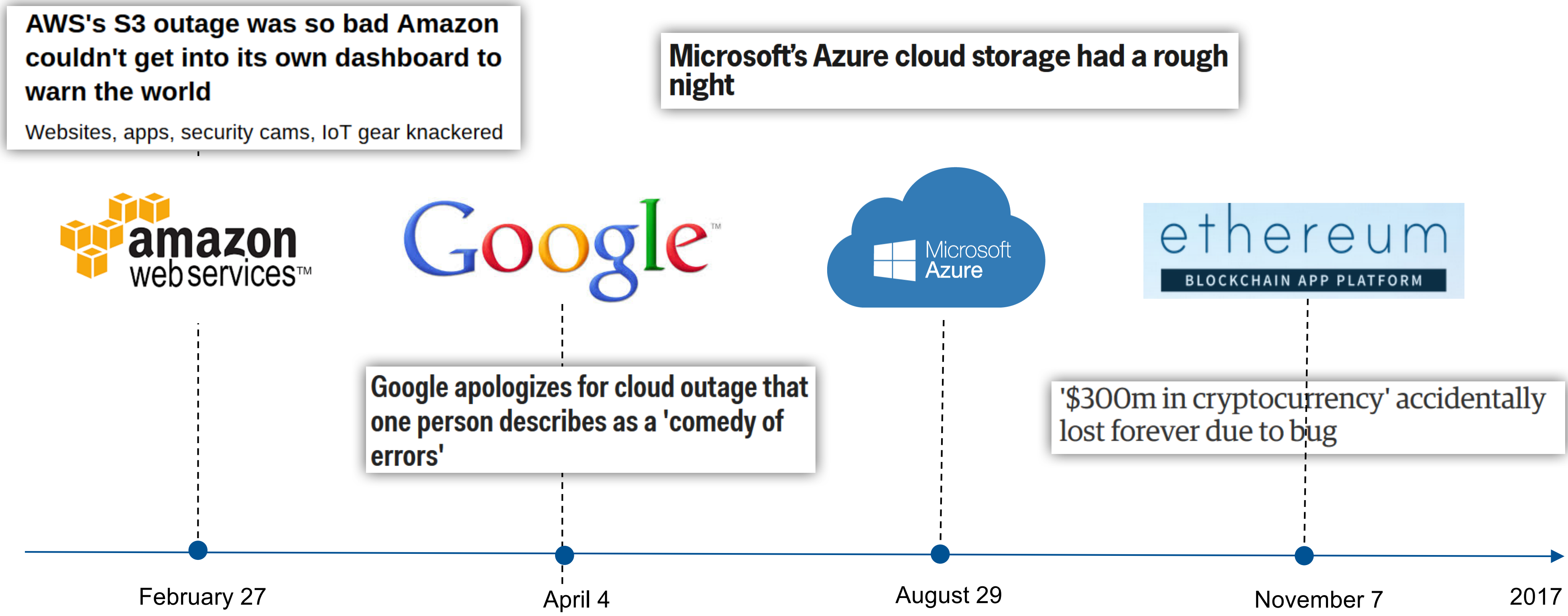
2) Provide an integated interface to diverse set of network devices

3) React to the events from data plane
   - topology inspection, routing of unknown packets, re-routing in case of failures...

4) Support plethora of built-in applications
   - VTN management, SFC embedding, ...

Ubiquity and magnitude of *software* failures

Software bugs are major root cause of customer-impacting incidents [Microsoft2017]



https://status.aws.amazon.com/
https://status.cloud.google.com/
https://azure.microsoft.com/en-us/status/history/
https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether

Ubiquity and magnitude of *network control software* failures

SOFTWARE

Google goes down for 5 minutes, Internet traffic drops 40%

Google suffered an outage affecting all of its services on Friday night, causing global Web traffic to plummet.

BY JOE SVETLIK | AUGUST 19, 2013 3:20 AM PDT

Google

BUT I HAVE WORK TO DO!

502. That's an error.

The server encountered a temporary error and could not complete your request.

Please try again in 30 seconds. That's all we know.

Source: CNET News https://goo.gl/DRmirb

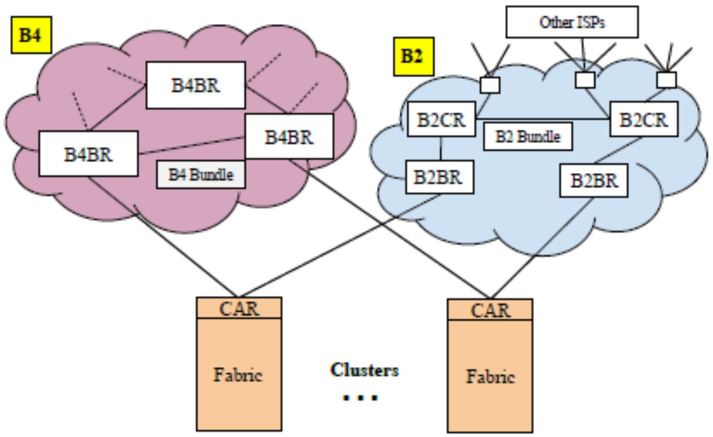
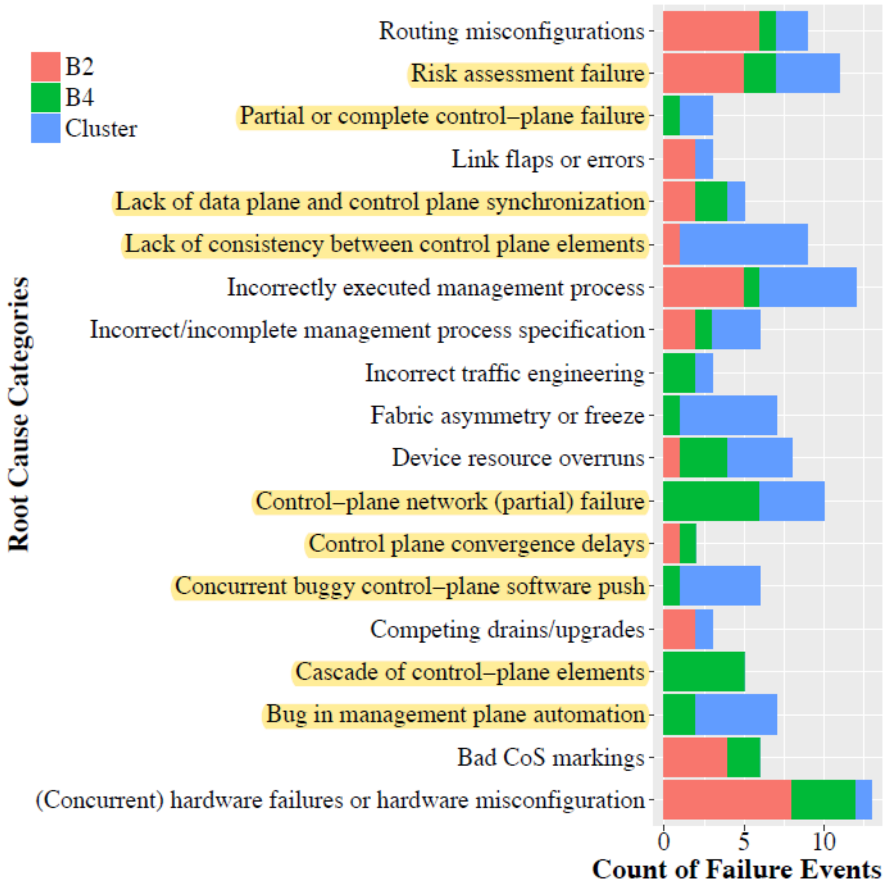


Figure 1: Google's Global Network

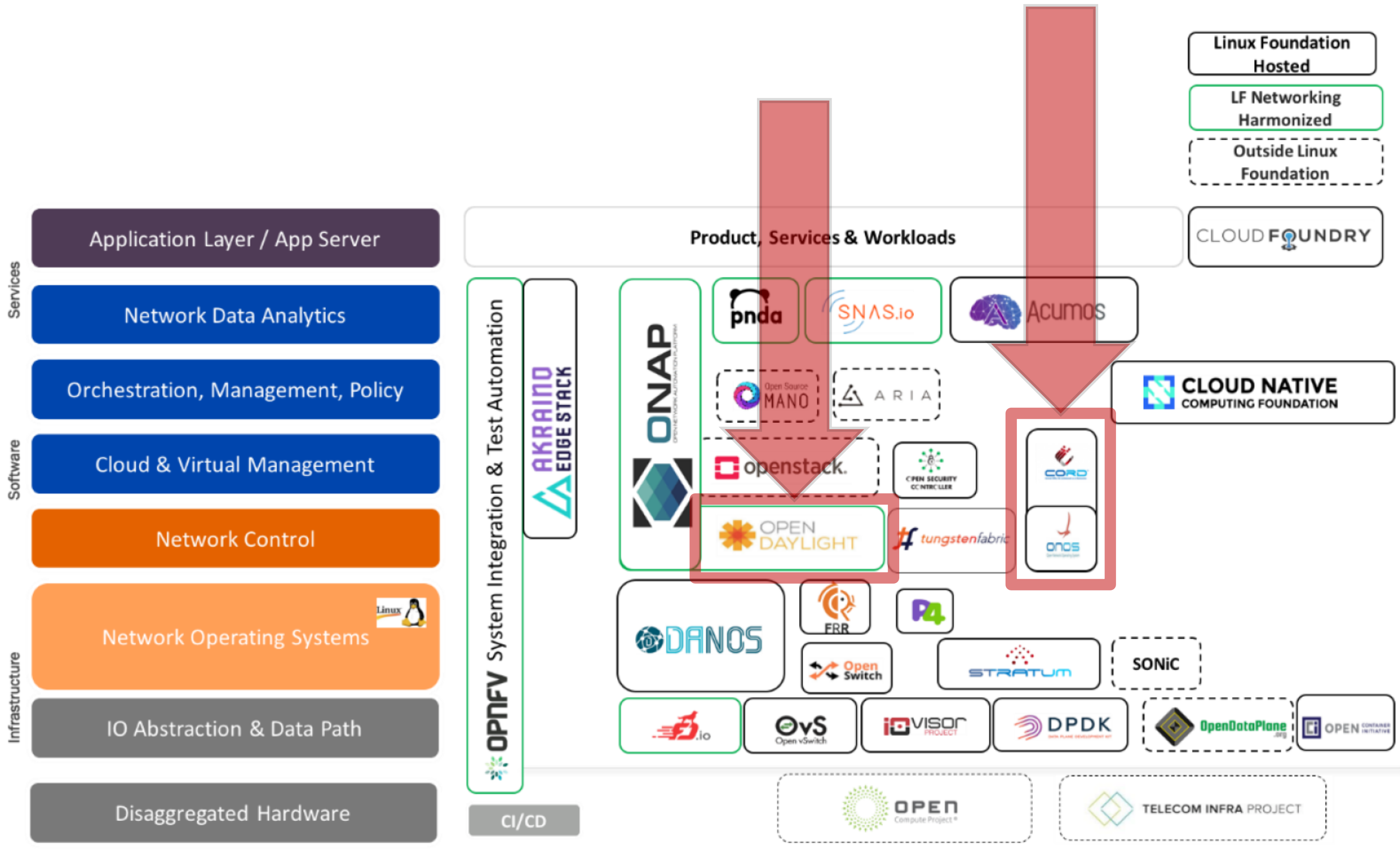
GOOGLE'S POSTMORTEM ANALYSIS

CONTROL PLANE ISSUES PREVAIL!!



Source: Govidan et al, "Evolve or die: High-availability design principles drawn from googles network infrastructure." ACM SIGCOMM Conference. 2016.

Open Source Networking Ecosystem



Automation of Network + Infrastructure + Cloud + Apps + IOT

Source: https://www.linuxfoundation.org/projects/networking/

Open source SDN orchestration platforms

ONOS – Open Network Operating System

- Service provider networks
- Focus on scalability, high-availability and carrier grade performance
- AT&T, NTT Communications, Google

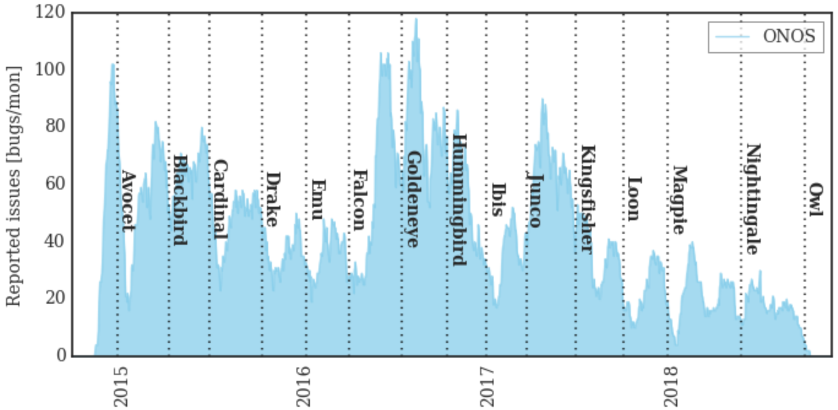
OpenDaylight

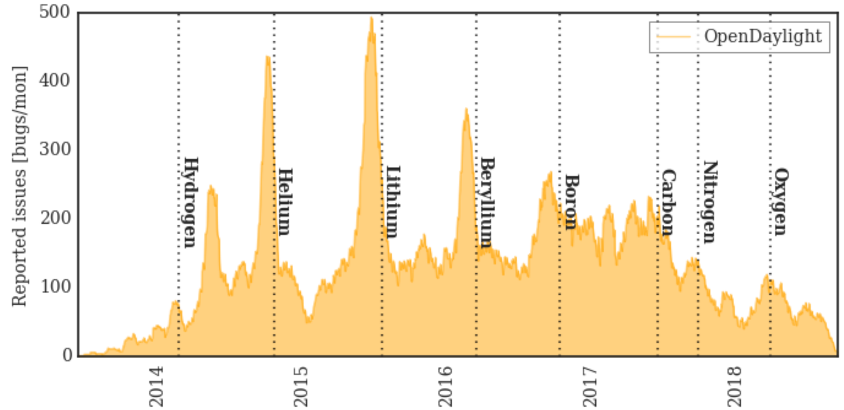
- "Linux of the networks"
- Data center applications, network virtualization and co-existence with legacy networks
- Cisco, Ericsson, HP, IBM, Juniper

Controller	ONOS	OpenDaylight
Project start	December 2014	February 2013
Current release	Quail (rel.17)	Fluorine (rel.9)
Commits	13k	99k
Lines of Code (LOC)	863,144	3,920,926
Bugs	2,193	9,394
Fault density [bug/kLOC]	2.5	2.4

*Data retrieved on March 15, 2019

Open source SDN orchestration platforms



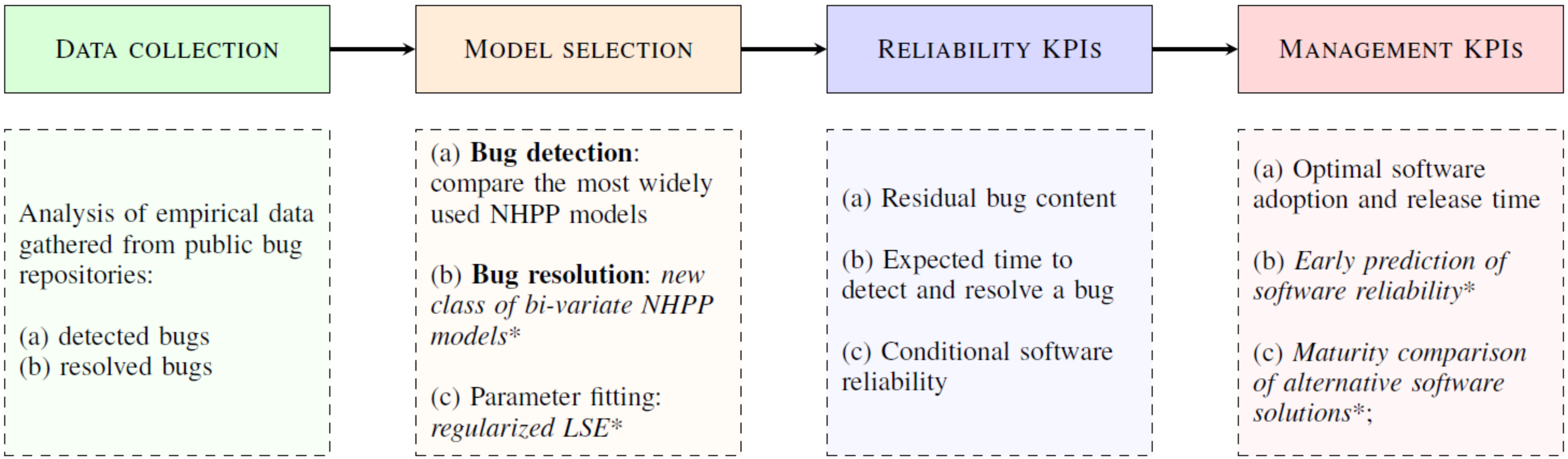


*Data retrieved on March 15, 2019

The only constant is change! ***Fast pace of software evolution***

Assessing the Software Maturity with SRGM

- Workflow
  1. Collecting the data from public bug repositories → ONOS and ODL maintain public Jira trackers
  2. Selection and parametrization of the best SRGM to describe bug maifestation process
  3. Evaluation of reliability KPIs → residual bug content, failure rate and interval reliability
  4. Release management decisions → optimal software release and adoption time



Source: Vizarreta et al., *Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models.* Transactions on Network and Service Management (TNSM), June 2018

Software Reliability Growth Models (SRGM)

Fault detection as Non-Homogeneous Poisson Process (NHPP)

- Initial number of bugs *N* is Poisson random variable with *E[N] = a*

$$P(N = n) = \frac{a^n}{n!} e^{-a}$$

- Probability that the single bug (sw fault) is manifested by the time *t*

$$p = F(t)$$

- Assuming time to discover every bug is i.i.d. we have Bernoulli trials

$$P(N(t) = k | N = n) = \binom{n}{k} p^k (1 - p)^{n-k}$$

- Using the theorem of total probability we derive distribution of cumulative number of detected bugs

$$P(N(t) = k) = \frac{[aF(t)]^k}{k!} e^{-aF(t)}$$

- Expected number of detected bugs by time *t*

$$m(t) = E[N(t)] = aF(t)$$

Software reliability growth models (SRGM)

Fault detection process

Fault detection as Non-Homogeneous Poisson Process (NHPP)

NHPP model is completely described by its mean value function *m(t)*

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

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

Expected time between failures

$$r(t) = E[a - N(t)] = a - m(t)$$

Residual bug content

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

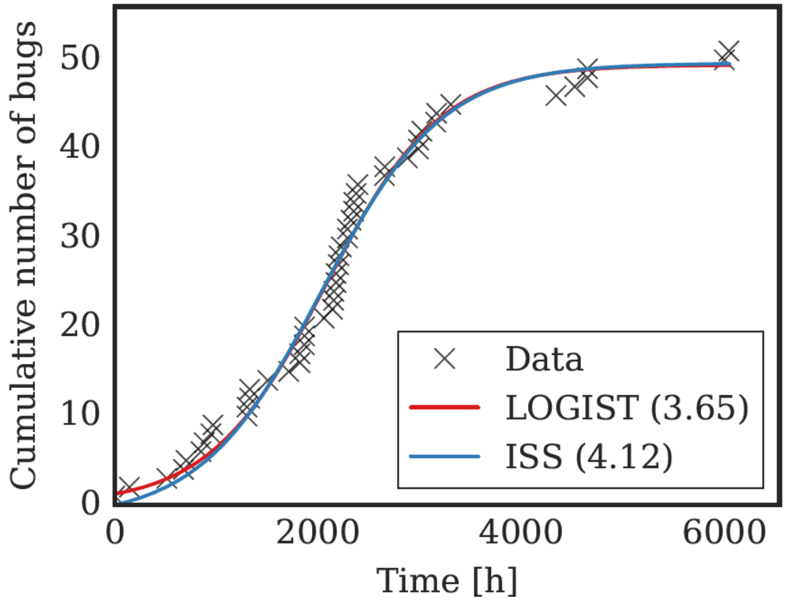
Conditional software reliability

Commonly used Non-Homogeneous Poisson Process (NHPP) [Lyu95]

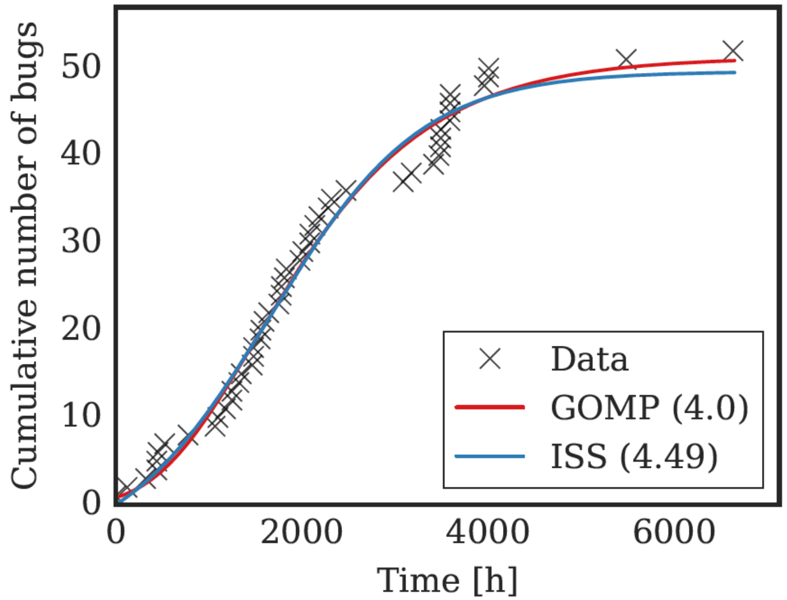
Model	Shape	Mean value function
Musa-Okumoto logarithmic	Concave	$m_{mo}(t) = a\ln(1+bt)$
Goel-Okumoto exponential	Concave	$m_{go}(t) = a(1-e^{-bt})$
Generalized Goel-Okumoto	S-shaped	$m_{ggo}(t) = a(1-e^{-bt^c})$
Ohba's inflection S-shaped	S-shaped	$m_{iss}(t) = a\frac{1-e^{-bt}}{1+\phi e^{-bt}}$
Yamada delayed S-shaped	S-shaped	$m_{dss}(t) = a(1-(1+bt)e^{-bt})$
Yamada exponential	Concave	$m_{yex}(t) = a(1-e^{-r(1-e^{-bt})})$
Gompertz	S-shaped	$m_{gomp}(t) = ak^{b^t}$
Logistic	S-shaped	$m_{logist}(t) = \frac{a}{1+ke^{-bt}}$

Selecting the Best SRGM for ONOS

Kingsfisher



Junco



Best fit across all releases are:

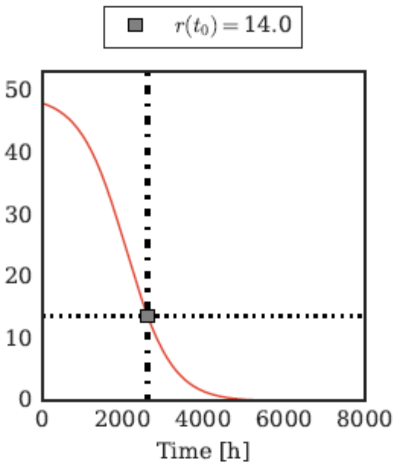
3-parameter S-Shaped Models: ISS, GGO, GOMP, LOGIST

Evaluation and Forecasting of Software Reliability Metrics

On the official release date of Kingsfisher

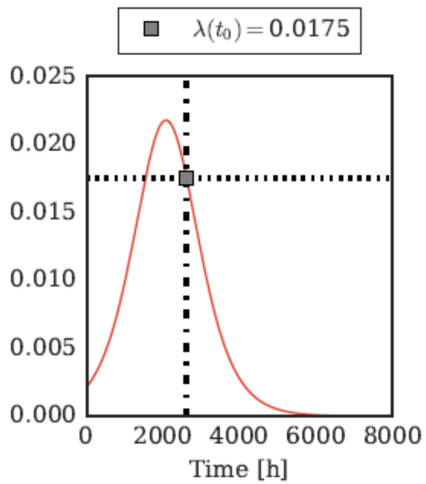
a. Residual bug content ~14 critical bugs
b. Expected failure rates ~2 days between bugs
c. Risk of having a critical outage in 3-month maintenance period

14 critical residual bugs



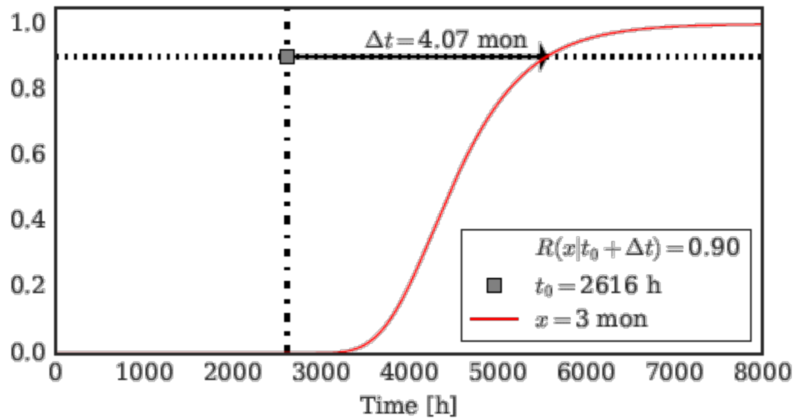
(a) Residual bug content $r(t)$

0.0175 bug/h ~ 2.38 days/bug



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

Release doption must be postponed 4 months for reliability of 0.9

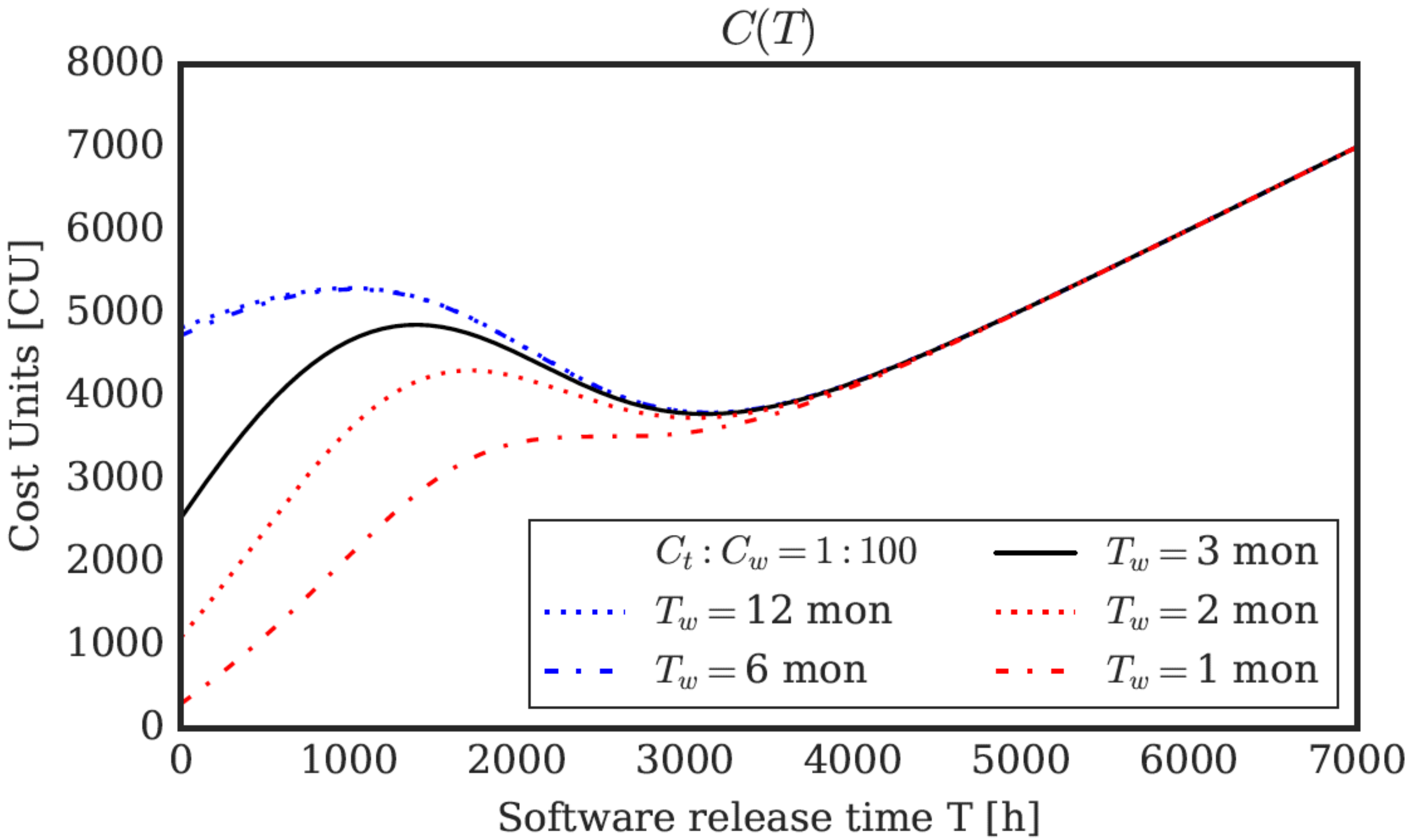


(c) Software reliability $R(x|t)$

Source: Vizarreta et al., *Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models.* Transactions on Network and Service Management (TNSM), June 2018

Management KPIs: Optimal Software Release Time

Kingsfisher: Cost-based Release Criteria



Management KPIs: Software Maturity Metric

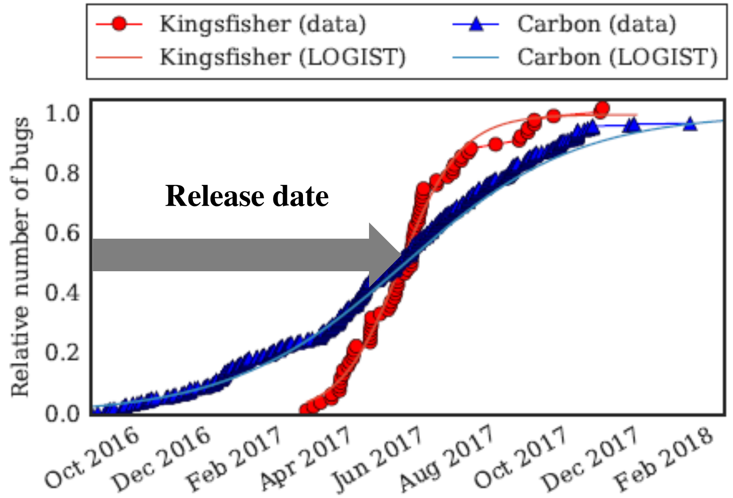
MEASURING RELIABILITY GROWTH

a) Maturity of ONOS v.s. ODL

Simultaneous released

Kingsfisher (ONOS v1.11)

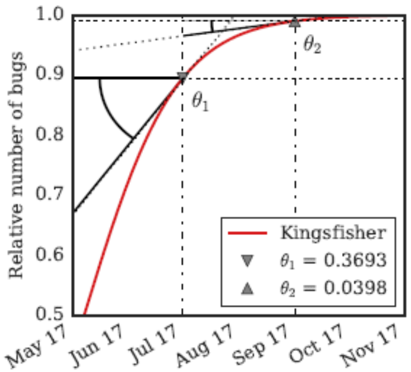
Carbon (ODL v0.7)



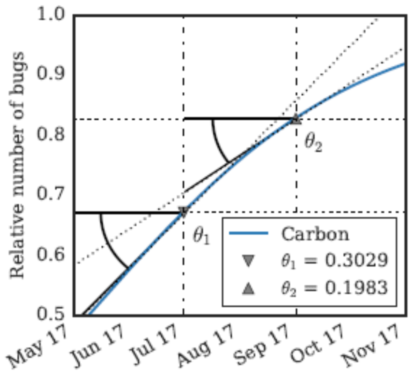
(a) Emprical and fitted data for the two controllers.

a) Software maturity Kingsfisher

b) Software maturity Carbon



(b) Software maturity of Kingsfisher: one (θ_1) and three months (θ_2) after the software release.



(c) Software maturity of Carbon: one (θ_1) and three months (θ_2) after the software release.

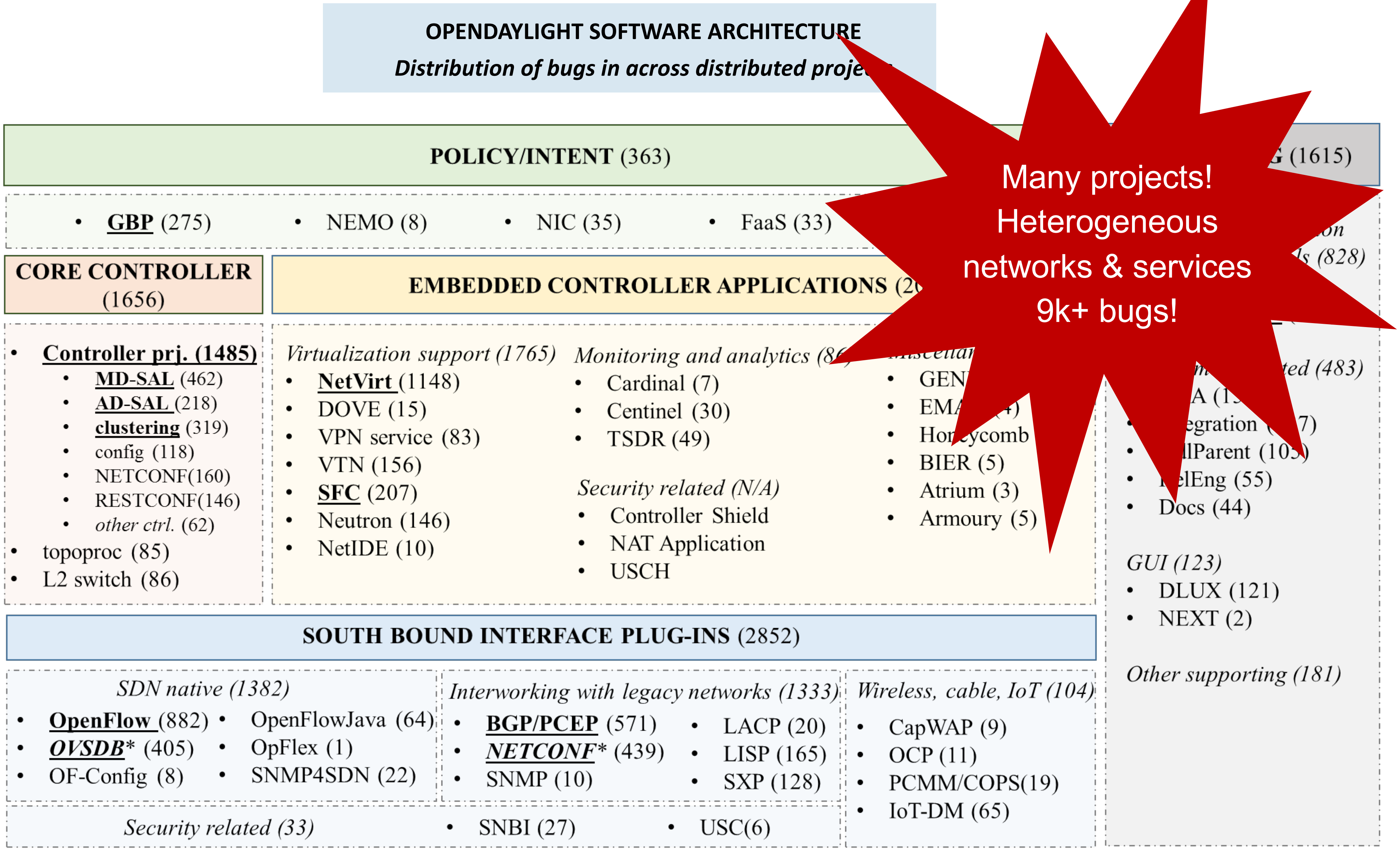
Source: Vizarreta et al., *Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models.* Transactions on Network and Service Management (TNSM), June 2018

Discussion

- Threats to validity
  1. ***Accuracy and completeness of data sets***
     - Fault report entries not complete
     - Creation time in future
  2. ***Inherent model assumptions of NHPP models***
     - Independent times between consecutive fault reports
     - Every bug contributes the same to the overall fault manifestation rate
     - Calendar time v.s. actual test effort (CPU time and men-hours)

- Applicability of NHPP models
  - Successfully applied to several large open source projects Mozilla Firefox, Eclipse IDE, Apache Server [Rossi20, Rah2009, Zhou2005, Ullah2013]
  - Identification of the most vulnerable software components
  - Early prediction of software reliability

Diversity brings complexity! ODL Architecture



Source: Vizarreta et al., *Mining Software Repositories for Predictive Modelling of Defects in SDN Controller*, submitted to IFIP/IEEE International Symposium on Integrated Network Management (IM), April 2019

Per-project Software Maturity

HIGH-FIDELITY MODELS

- TTF: Ohba's Inflection S-Shaped (ISS) NHPP
- TTR: log-normal distribution

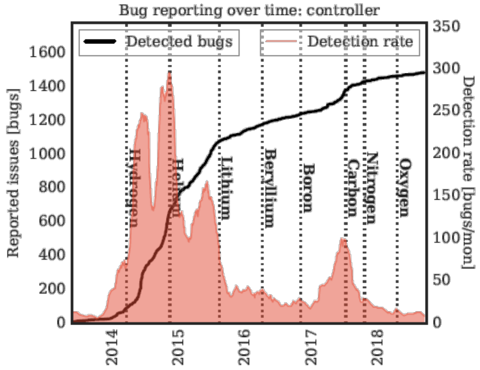
- Applications
  1. Allocation of test effort

$$\max \sum_{p\in\text{Projects}} \left[N_p^{bugs}(t_0) - N_p^{bugs}(t_0+t_p)\right]$$
$$\text{s.t.} \sum_{p\in\text{Projects}} t_p \le T_{\text{budget}}$$

  2. Software release management/adoption policy

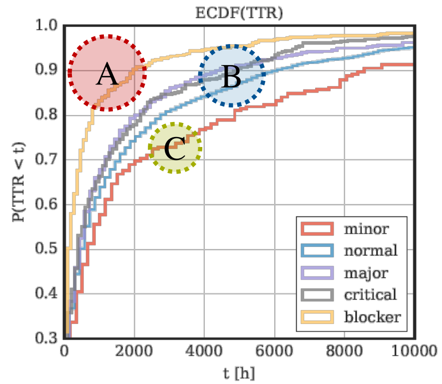
$$\min T_R$$
$$\text{s.t.} \max_{p\in\text{Projects}} \lambda_p(T_R) \le \lambda_0$$

A) TTF DEPENDS ON TIME



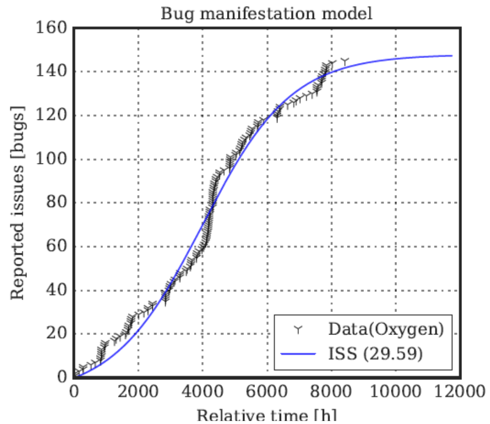
(a) $\text{TTF}^{-1} = f(t)$

B) TTR DEPENDS ON BUG CRITICALITY



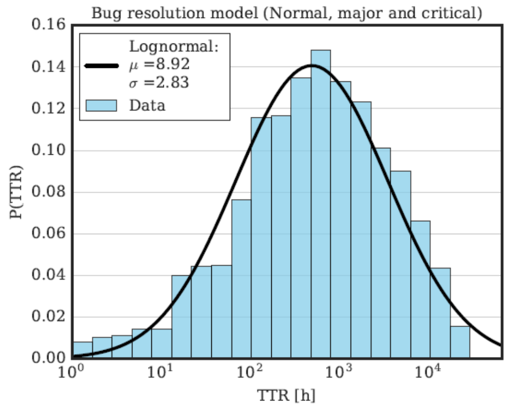
(b) $\text{TTR} = f(\text{severity})$

C) MODELLING TTF WITH ISS



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

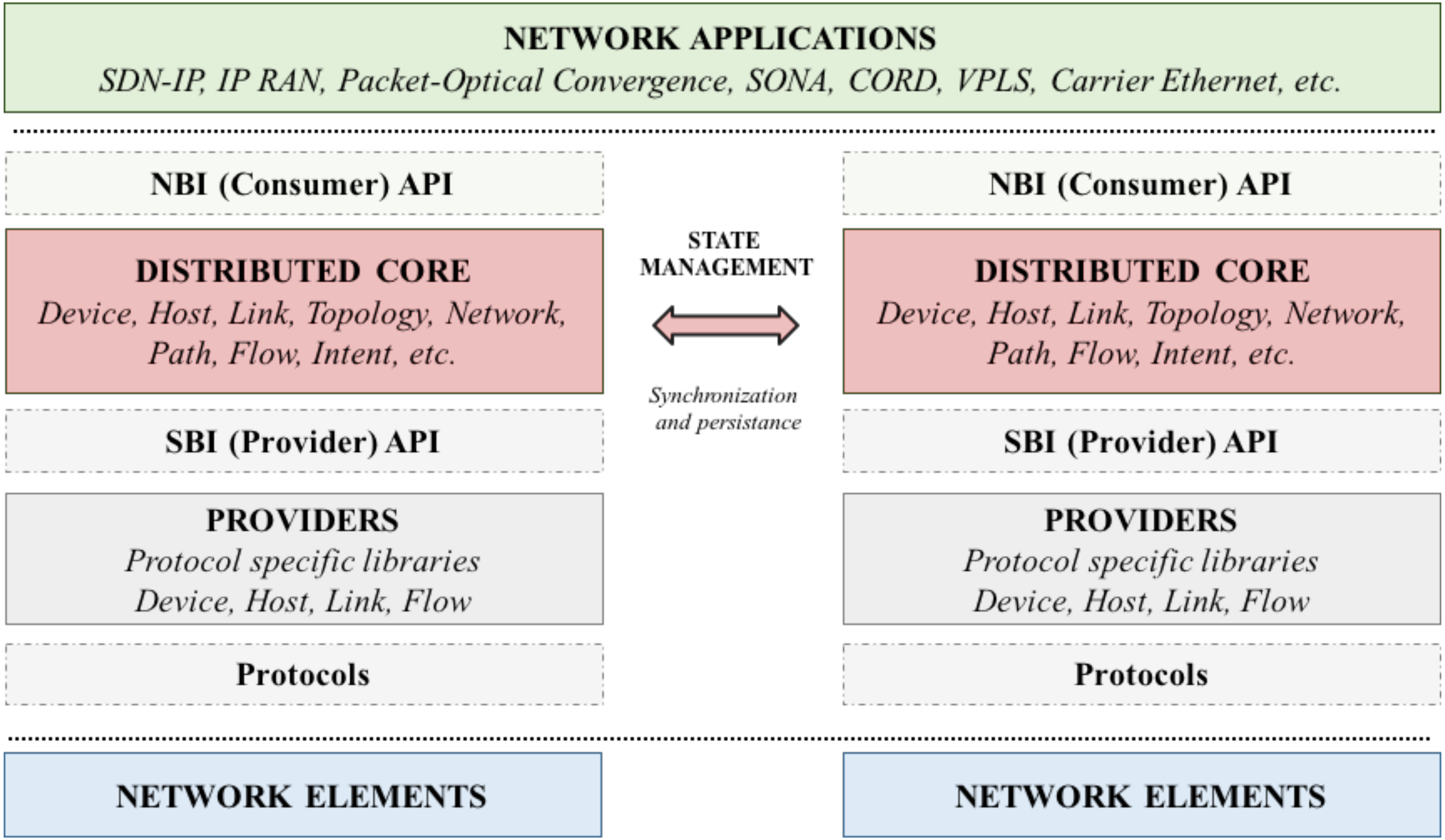
D) TTR FOLLOWS LOG-NORMAL DISTRIBUTION



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

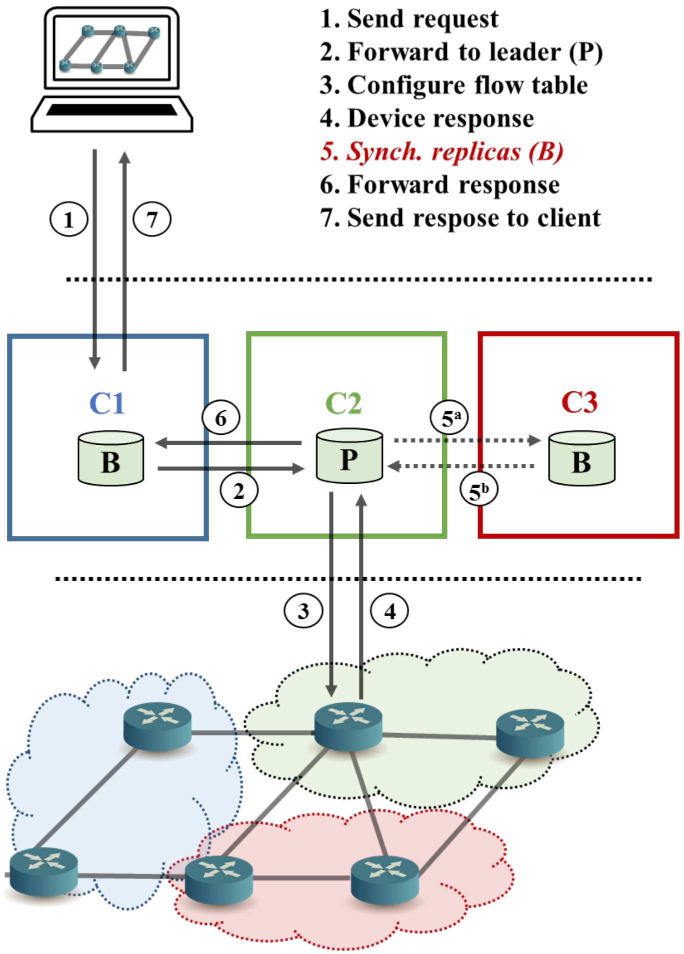
Source: Vizarreta et al., *Mining Software Repositories for Predictive Modelling of Defects in SDN Controller*, submitted to IFIP/IEEE International Symposium on Integrated Network Management (IM), April 2019

Diversity brings complexity! ONOS Architecture



Replicating problems! *Fallacies of distributed SDN implementations*

LOGICALLY CENTRALIZED - PHYSICALLY DISTRIBUTED



BUGS IN DISTRIBUTED IMPLEMENTATION

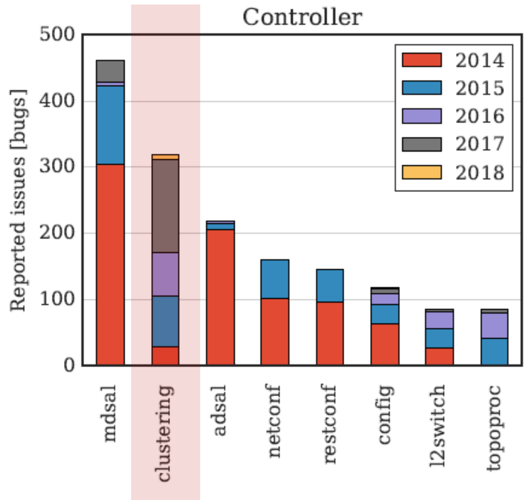


TABLE I: Distribution of software defects by category: distributed protocols (*DP*), scalability and performance (*SP*), high-availability (*HA*) and operational (*OP*) issues.

Category	ONOS	ODL	Total
DP	97 (44%)	119 (36%)	216 (40%)
SP	39 (18%)	52 (16%)	91 (17%)
HA	42 (19%)	76 (23%)	118 (21%)
OP	30 (14%)	46 (14%)	76 (14%)
Other	13 (6%)	42 (13%)	55 (9%)
Total	221 (40%)	335 (60%)	**556**

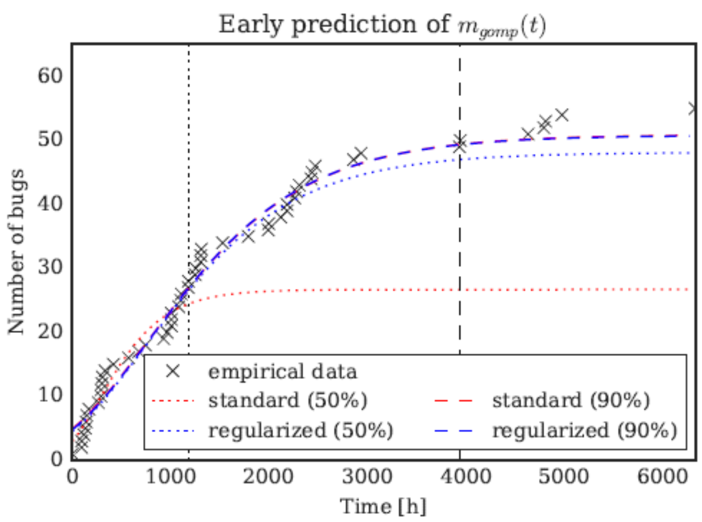
Source: Vizarreta et al., *DASON: Dependability Assurance Framework for Imperfect Distributed SDN Implementations*, TNSM, under revision, 2019

Early Prediction of Software Reliability Metrics

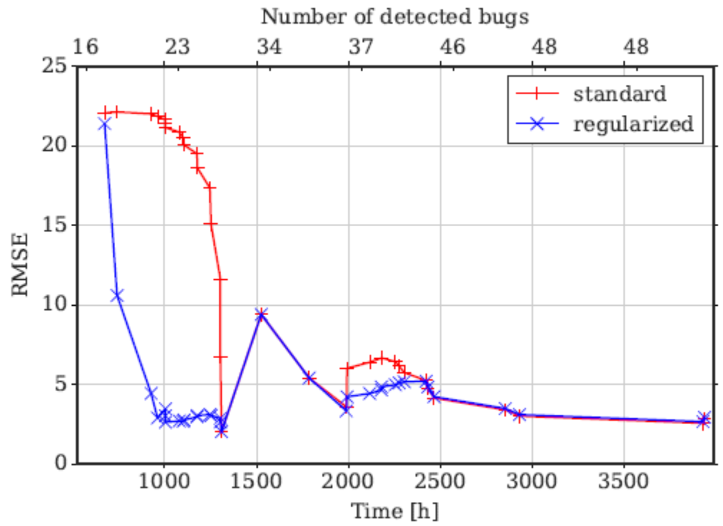
$$m_\xi^i \leftarrow \omega\, \xi^i + (1-\omega)\, m_\xi^{i-1}$$

4) IMPROVING THE ACCURACY OF ERALY PREDICTABILITY OF SRGM

- Large number of samples (outages) needed to estimate model parameters
  - Too late if network control software is already deployed!
- Regularize model parameter search space for early prediction
  - Exponentially weighted moving average



(a) Early prediction of mean value function $m_{gomp}(t)$, when limited number of samples are available.



(b) Evolution of Root Mean Square Error (RMSE) with the number of training samples.

Source: Vizarreta et al., *Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models.* Transactions on Network and Service Management (TNSM), June 2018

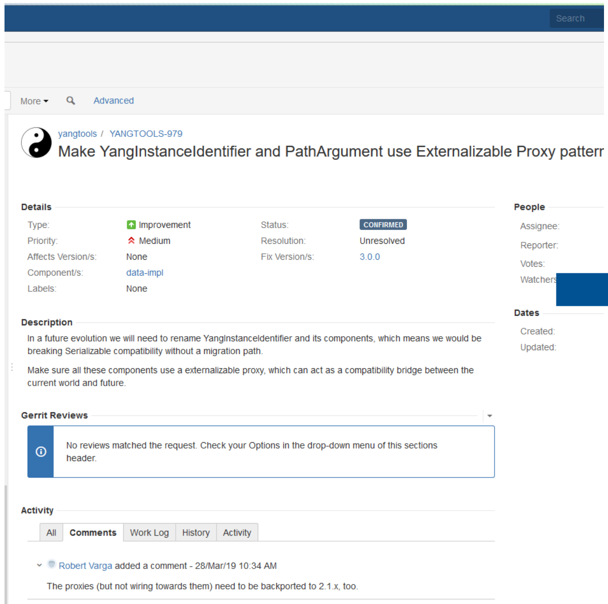
Potential applications of machine learning in software QA

“Software is eating the world, but A.I. is going to eat software.”

AUTOMATIED PROBLEM INFERENCE WITH NLP

Processing large amounts of information with Natural Language Processing (NLP) from:

1. Public bug repositories and issue trackers (10k+)
2. Software logging statements [7,8]
3. Code features and descriptors [9]



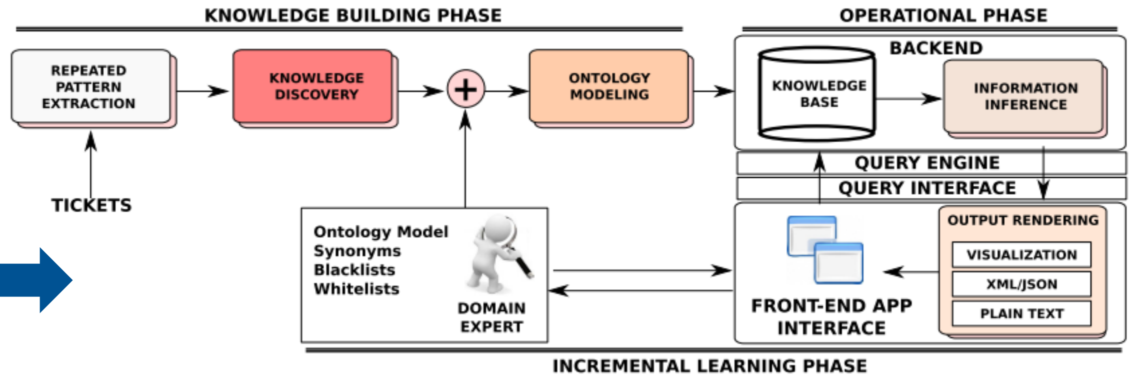


Figure 4: NetSieve Architecture: The first phase builds a domain-specific knowledge base using existing tickets. The second phase uses the knowledge base to make problem inference. The third phase leverages human guidance to improve the inference accuracy.

Source: Potharaju R. et al., *Juggling the Jigsaw : Towards Automated Problem Inference from Network Trouble Tickets, USENIX NSDI, 2013* ([6])

Questions?

References

[1] Govidan et al, "Evolve or die: High-availability design principles drawn from googles network infrastructure." ACM SIGCOMM Conference. ACM, 2016.

[2] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, A. Blenk, W. Kellerer, C. Mas Machuca, "Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models". Transactions on Network and Service Management (TNSM), June 2018

[3] P. Vizarreta, P. Heegaard, B. Helvik, W. Kellerer, C. Mas Machuca, "Characterization of Failure Dynamics in SDN Controllers". In Proc. of IEEE International Workshop on Reliable Networks Design and Modeling, Alghero, Italy, 2017

[4]] P. Vizarreta, V. Mendiratta, L. Jagadeesan, W. Kellerer, C. Mas Machuca, K. Trivedi, "DASON: Dependability Assessment Framework for Imperfect Distributed SDN Implementations", under revision, 2019

[5] P. Vizarreta, E. Sakic, W. Kellerer, C. Mas Machuca, "Mining Software Repositories for Predictive Modelling of Defects in SDN Controller", submitted to IFIP/IEEE International Symposium on Integrated Network Management (IM), April 2019

[6] R. Potharaju et al., "Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets", USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)

[7] P. He, et al., Towards automated log parsing for large-scale log data analysis, IEEE Transactions on Dependable and Secure Computing, 2018

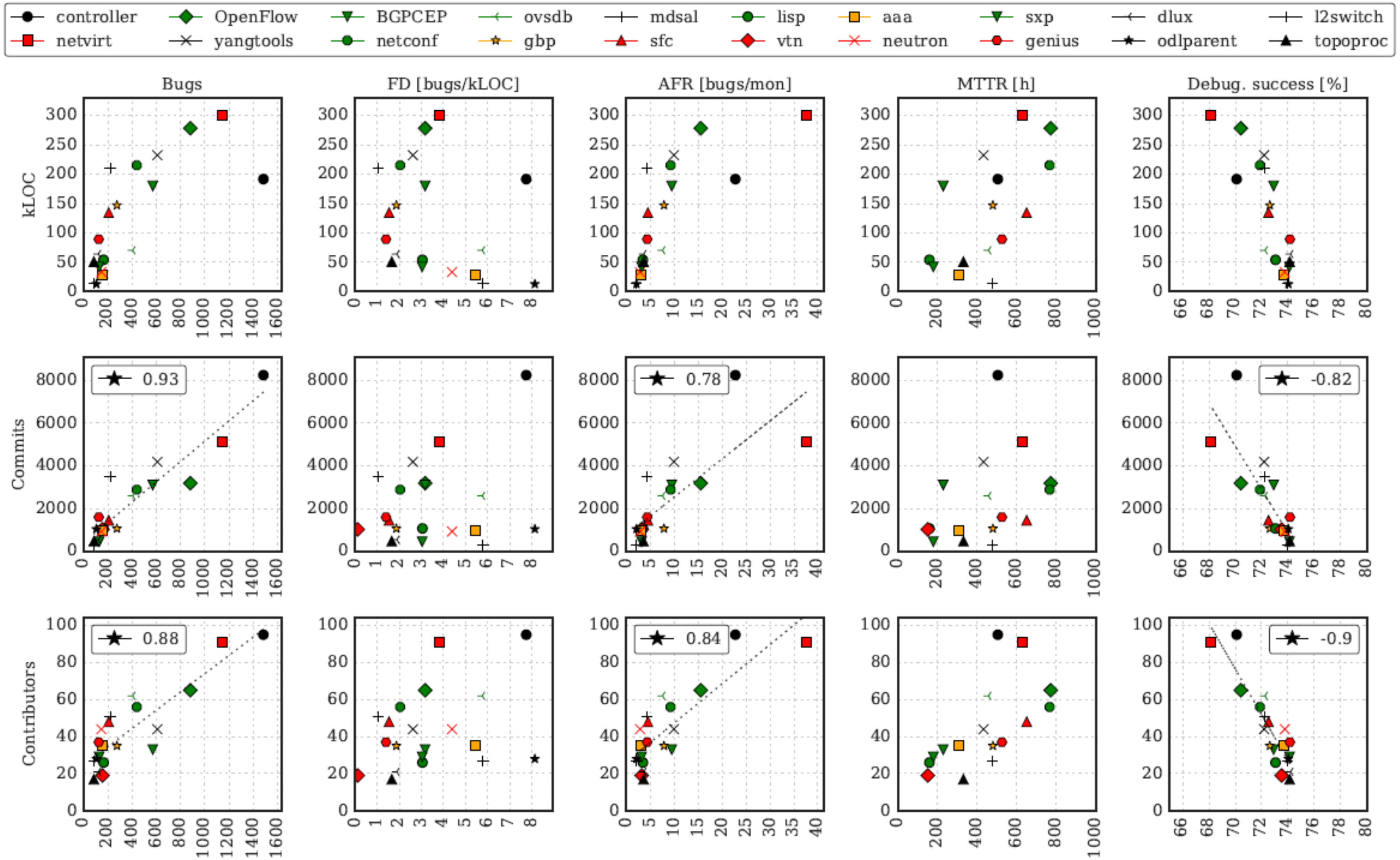
[8] P. He, et al., Characterizing the natural language descriptions in software logging statements, ACM International Conference on Information and Knowledge Management, 2017

[9] F. Qin, et al., “Studying aging-related bug prediction using cross-project models,” IEEE Transactions on Reliability, 2018

[10] M.R. Lyu Handbook of software reliability engineering. CA: IEEE computer society press; 1996

Learning from mistakes: *Early-prediction models*

CORRELATION BETWEEN CODE INTERNALS AND SOFTWARE DEFECTS



Empirical fault density
3 bugs/kLOC (Java)

Project activity
1 bug : 5 commits

Community size
15 bugs : contributor

Source: Vizarreta et al., *Mining Software Repositories for Predictive Modelling of Defects in SDN Controller*, submitted to IFIP/IEEE International Symposium on Integrated Network Management (IM), April 2019

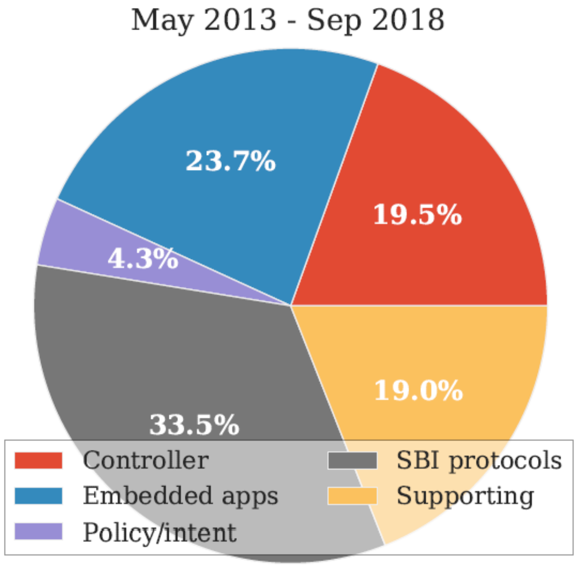
Diversity brings complexity! *Heterogenity of supported networks*

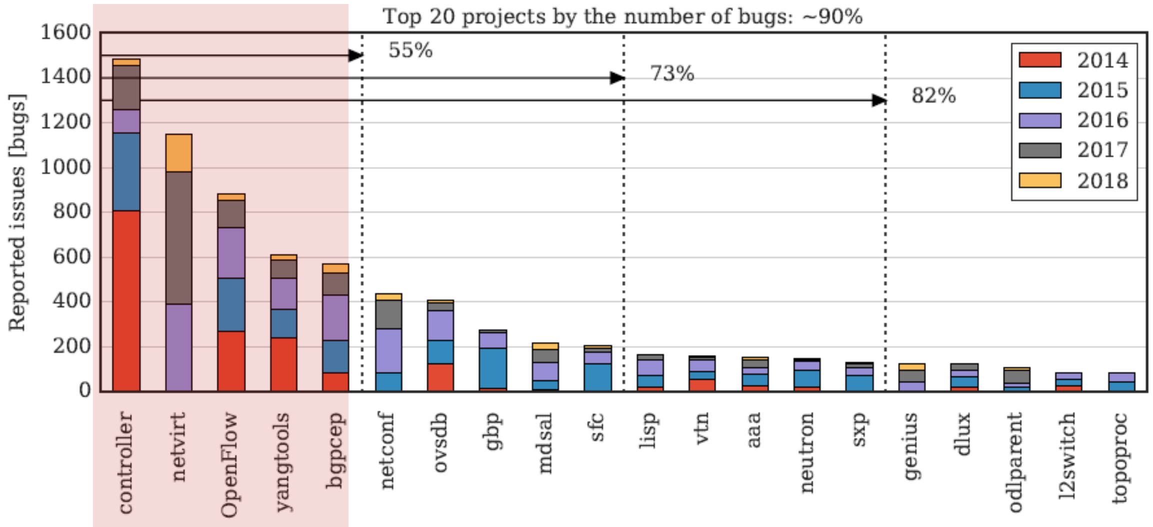
OPENDAYLIGHT SOFTWARE ARCHITECTURE

Distribution of bugs in across distributed projects

ISSUES RELATED TO SUPPORT OF DIFFERENT NETWORKS PREVAIL

TOP 5 PROJECTS CONTRIBUTE TO >50% BUGS





Source: Vizarreta et al., *Mining Software Repositories for Predictive Modelling of Defects in SDN Controller*, submitted to IFIP/IEEE International Symposium on Integrated Network Management (IM), April 2019

Potential applications of machine learning in software QA

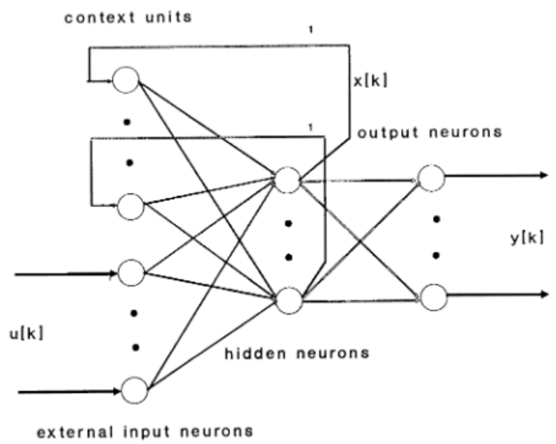
"Software is eating the world, but A.I. is going to eat software."

IMPROVING ACCURACY OF QA FORECASTING WITH ANN

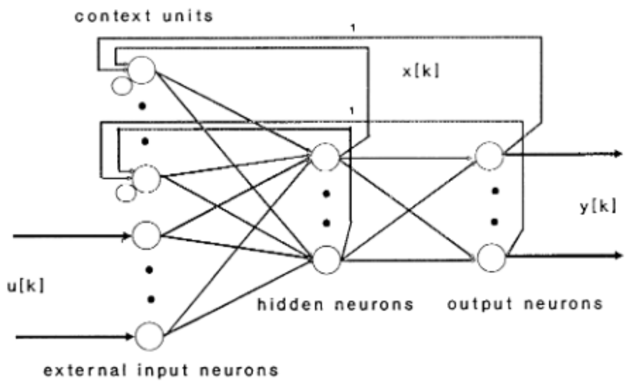
Improving predictive accuracy oftrend analysis with Artificial Neural Networks (ANN) for:

1. Software maturity, i.e., reliability growth analysis [10]
2. Performance degradation trends

ELMAN RECURRENT NETWORKS



JORDAN RECURRENT NETWORKS



Source: D.T. Pham, et al., Training Elman and Jordan networks for system identification using genetic algorithms, Artificial Intelligence in Engineering, 1998