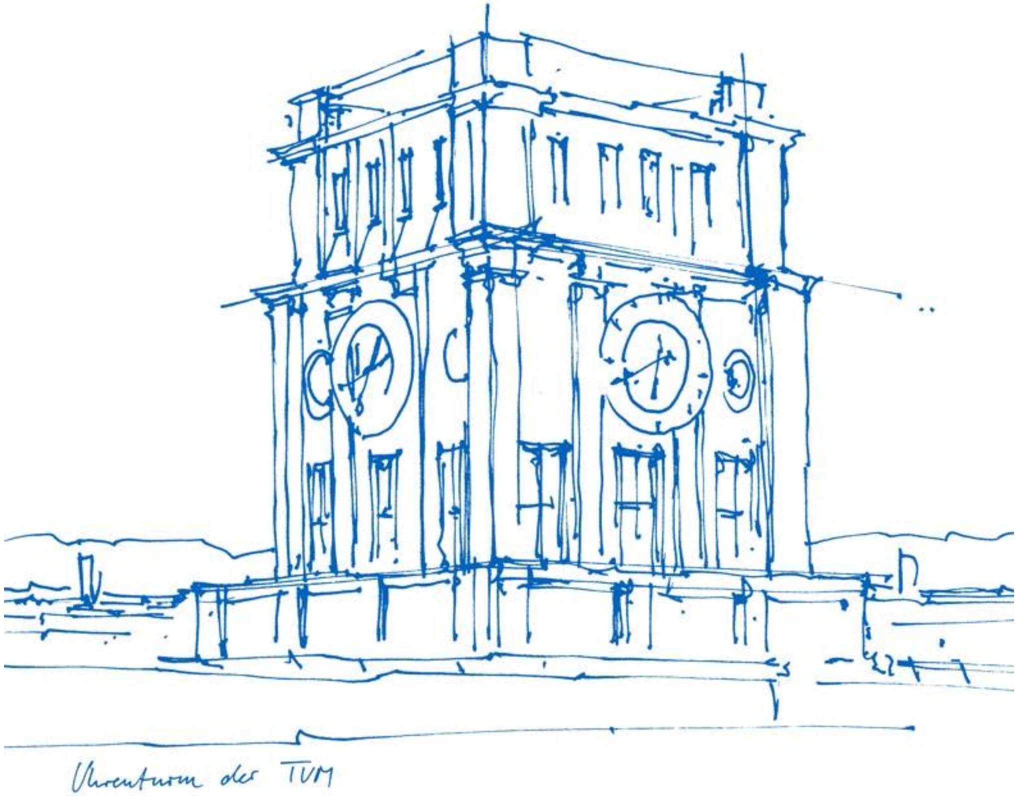
Chair of Communication Networks
Department of Electrical and Computer Engineering
Technical University of Munich

Software Dependability:
A case study on Software Defined Networks

Carmen Mas Machuca, Petra Vizarreta
Chair of Communication Networks,
Technical University of Munich, Germany



Ubiquity and magnitude of software failures

- Software bugs contribute more than 35% of critical network outages [Google2016]
- Bugs caused more than 33% of customer impacting incidents [Microsoft2017]

28.02.2017 09:37 (PST

S3 Service Disruption

the Northern Virginia

(US-EAST-1) Region

https://aws.amazon.co

message/41926/

22.06.19 03:00 to 22.06.20

A widespread BGP routin

Internet services and a p

https://bgr.co

2019-google-amazon-re

26.03.20 16:14 to 27.03.

Cloud IAM experience

across many services

(resulting in continue

subset of services) fo

https://sta

19.05.20 13:30 to 16:30 (UTC).

A **bug** caused high resource utilization in the internal cluster service that is responsible for receiving and executing service management operations in the East US region. The bug was encountered in **all** the service instances of the region leading to **failures** and **timeouts** for management operations

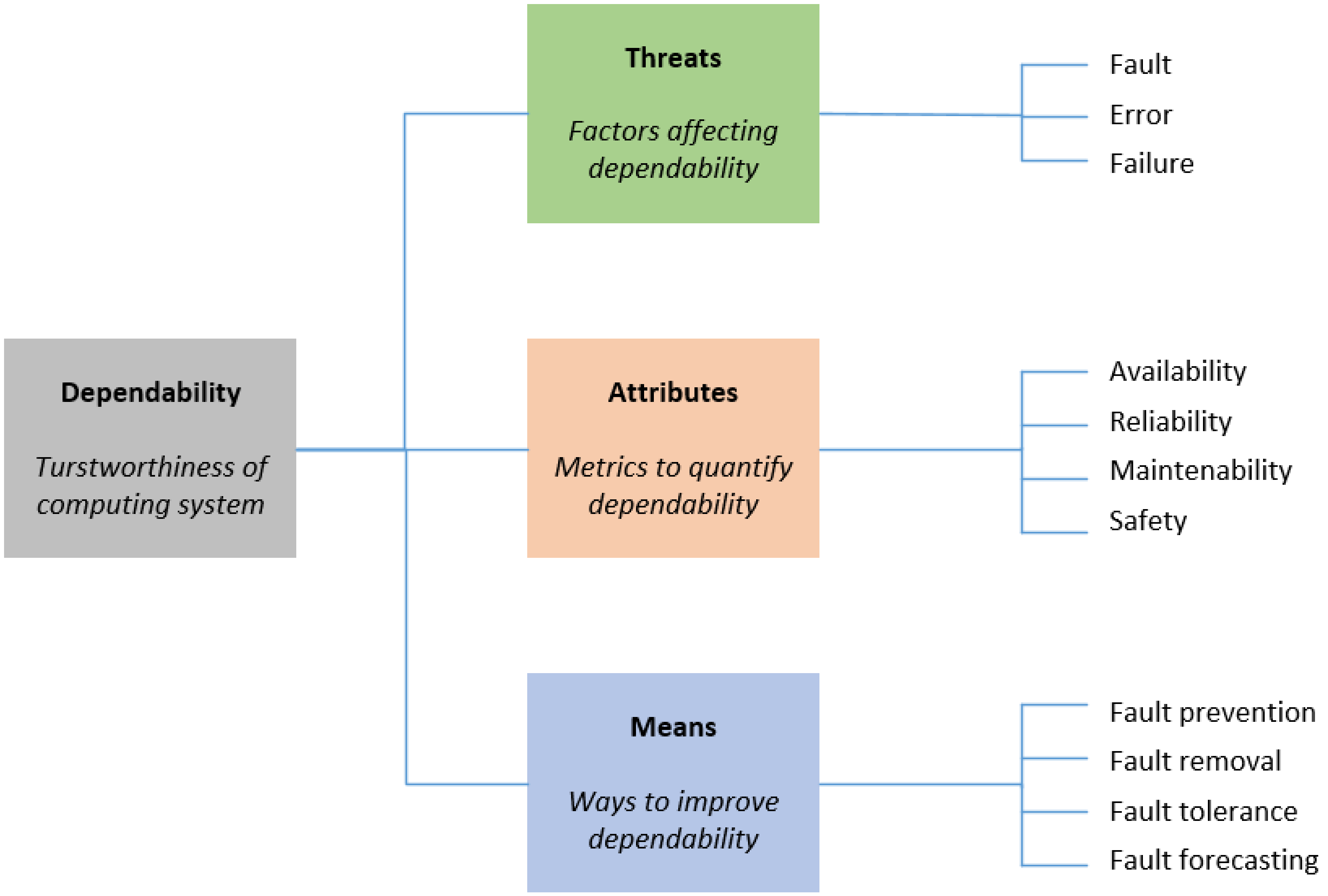
https://status.azure.com/en-us/status/history

PST: Pacific Standard Time
CET: Central European Time

Outline

- Terms and Taxonomy
- Software Dependability Problem
- Addressed questions applied to SDN:
  - How reliable a controller is?→ Steady-state availability
  - How often does software fail?→ Bug forecasting and Software Maturity evaluation
  - What is the impact? → User-perceived service
- Conclusions

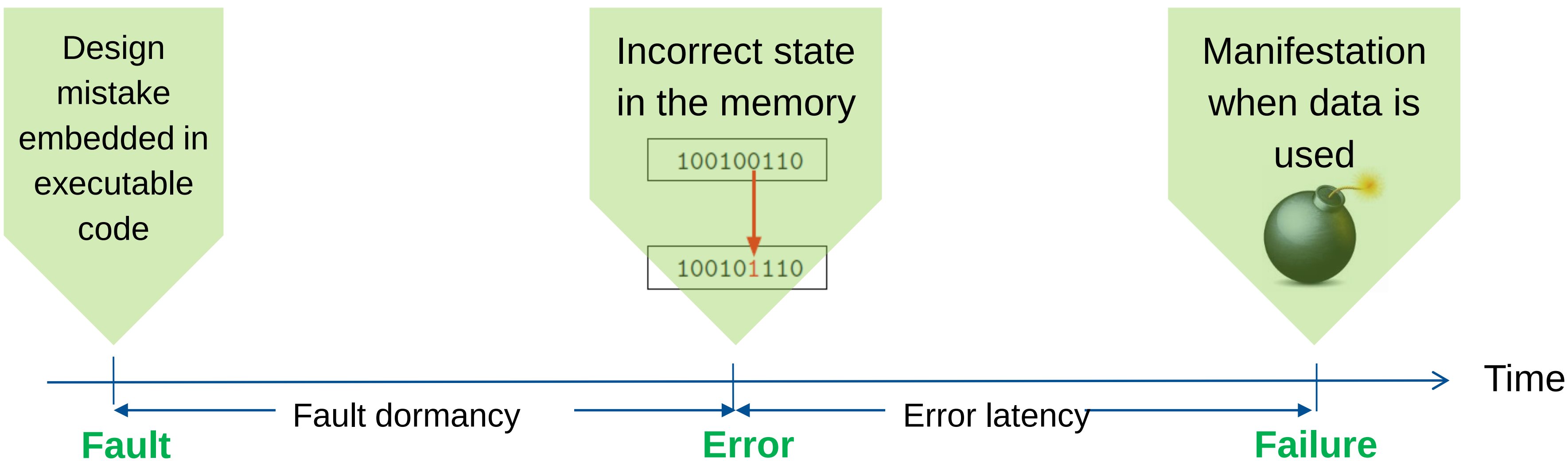
Terms and Taxonomy



Source: IFIP WG10.4 Dependable Computing and Fault Tolerance https://www.dependability.org/wg10.4/

Terms and Taxonomy

- **Fault**: Adjudged or hypothesized cause of an error.
- **Error**: Part of a system state which is liable to lead to failure.
- **Failure**: Deviation of the delivered service according to its specification.



- <u>Active</u>: it produces an error
- <u>Dormant</u>: it has not produced an error

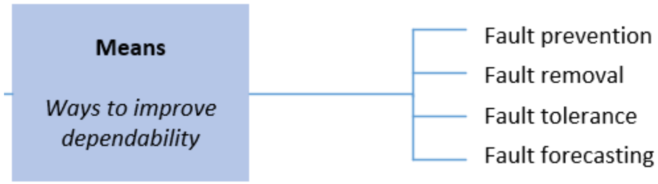
- <u>Detected</u>: it has manifestated as failure
- <u>Latent</u>: it has not been detected

Terms and Taxonomy

- **Availability**: The ability of an item to perform its required function, under environmental and operational conditions at a stated instant of time.
- **Reliability**: The ability of an item to perform its required function, under environmental and operational conditions, for a stated period of time.
- **Maintenability**: the probability of performing a successful repair and maintenance action within a given time.
- **Safety**: Ability of an item to provide its required function without the occurrence of catastrophic consequences on the user(s) and the environment.

Source: ISO 8402 and British Standard BS 4778

Terms and Taxonomy



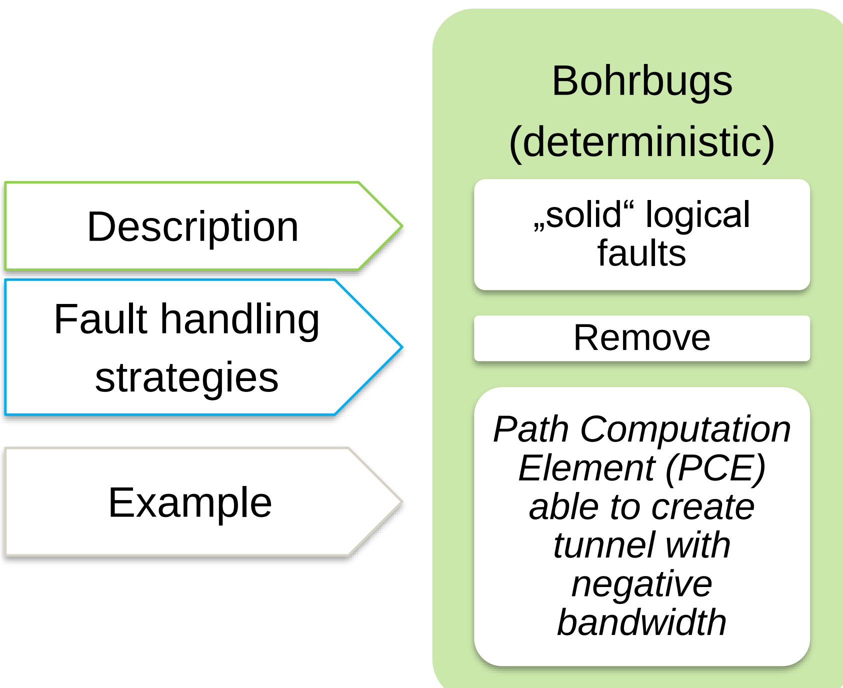
- **Fault prevention** is attained by quality control techniques employed during the design and manufacturing of hardware and software.
- **Fault removal** is performed both during the development phase (verification, diagnosis, and correction), and during the operational life of a system (either corrective or preventive maintenance).
- **Fault tolerance** is intended to preserve the delivery of correct service in the presence of active faults.
- **Fault forecasting** is conducted by performing an evaluation of the system behaviour with respect to fault occurrence or activation: either qualitative (identify, classify, rank the failure modes), or quantitative (probabilities to which some of the attributes are satisfied).

Source: “Fundamental Concepts of Dependability” A. Avizienis et al.

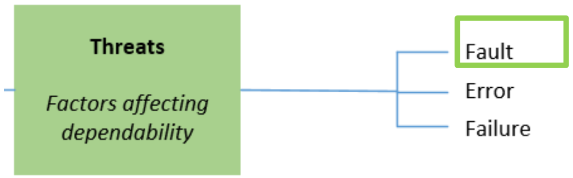
Terms and Taxonomy

- Software fault = bug

- Types of software faults:



Terms and Taxonomy



- Software fault = bug

- Types of software faults:

	Bohrbugs (deterministic)	Mandelbugs (non-deterministic)
Description	„solid“ logical faults	„relative“ logical faults
Fault handling strategies	Remove	Retry, replicate
Example	*Path Computation Element (PCE) able to create tunnel with negative bandwidth*	*Distributed database locking in ONOS*

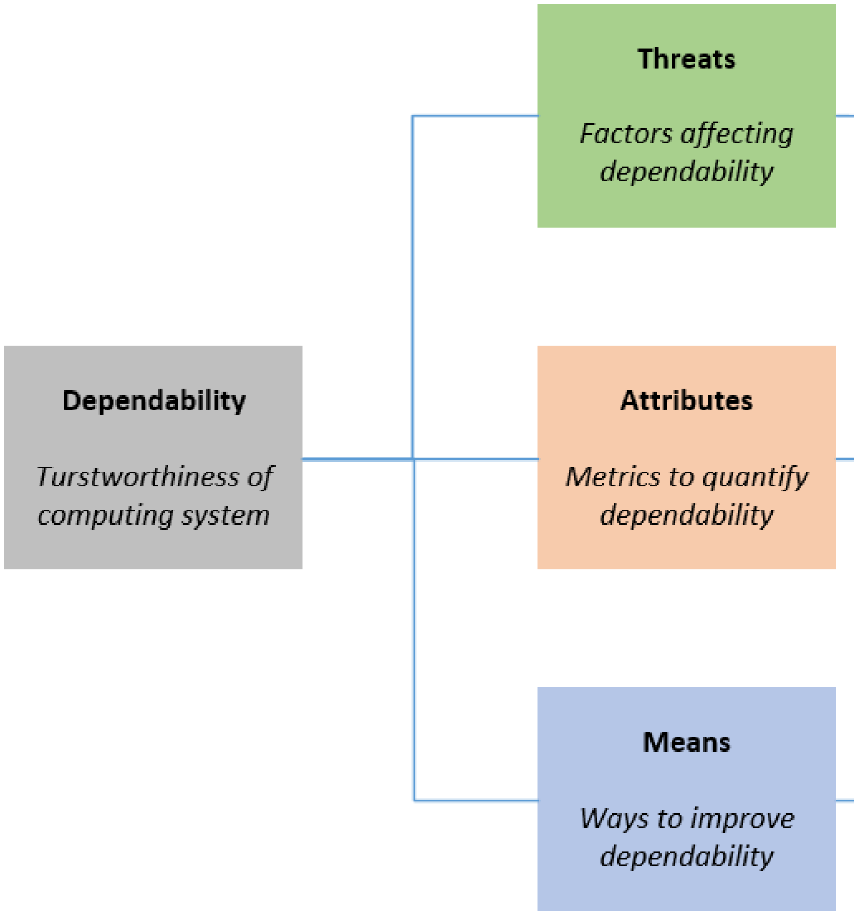
Terms and Taxonomy

- Software fault = bug

- Types of software faults:

	Bohrbugs (deterministic)	Mandelbugs (non-deterministic)	Aging-related bugs
Description	„solid“ logical faults	„relative“ logical faults	Degradation with time
Fault handling strategies	Remove	Retry, replicate	Rejuvenate
Example	*Path Computation Element (PCE) able to create tunnel with negative bandwidth*	*Distributed database locking in ONOS*	*Flows still reported in oper data store after they have been deleted from both config and network*

Limitations of the State of the Art



- **Threat** analysis focus on independent component failures
  - Focused on hardware failures
  - Software related failures *neglected* or *oversimplified (e.g., as single failure mode)*
- **Attributes**, e.g.,
  - *reliability*, does not precisely describe software behaviour
    - *Reliability growth* due to *maturity*
    - *Reliability degradation* due to *aging*
- **Means** focus on structural protection
  - Fault *prevention*, *removal* and *forecasting* have been overlooked

Source: IFIP WG10.4 Dependable Computing and Fault Tolerance https://www.dependability.org/wg10.4/

Software Dependability Problem

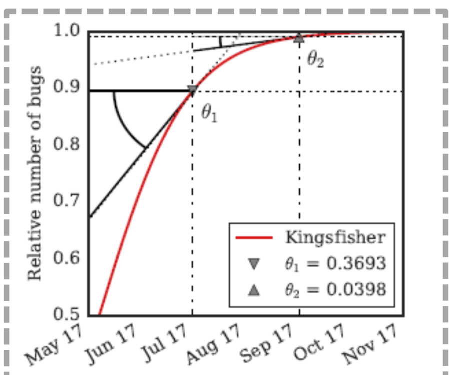
- Softwarized networks
- Open source code

Target: Realistic and practical dependability assurance framework

Proposed methodology based on Statistical inference techniques and stochastic dependability models

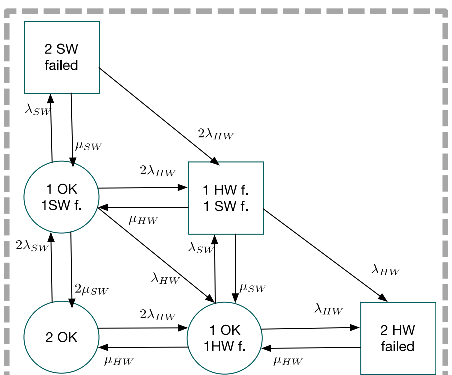
How often does SW fail?

Failure forecasting and Software Maturity



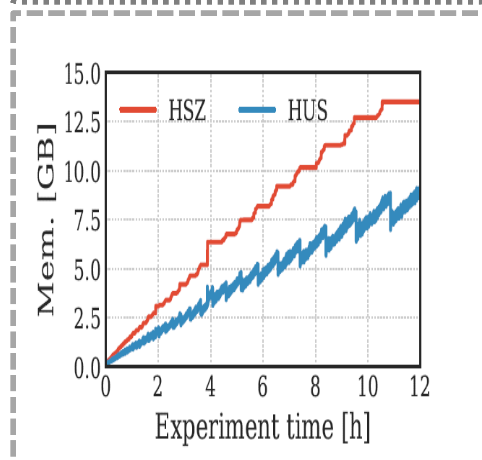
How often is the controller available?

Steady-state availability



Do Softwarized networks age?

Proposed framework



Software Dependability Problem

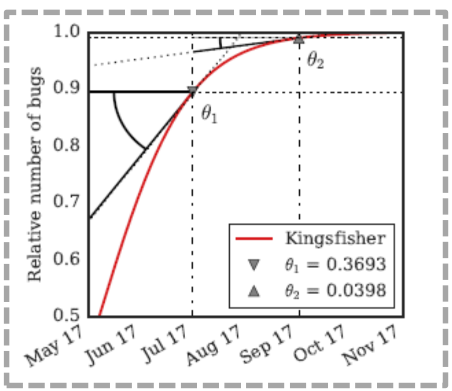
- Softwarized networks
- Open source code

Target: Realistic and practical dependability assurance framework

Proposed methodology based on Statistical inference techniques and stochastic dependability models

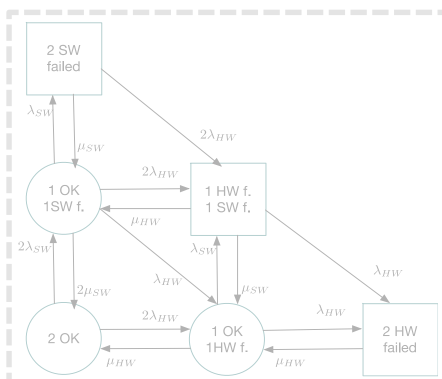
How often does SW fail?

Failure forecasting and Software Maturity



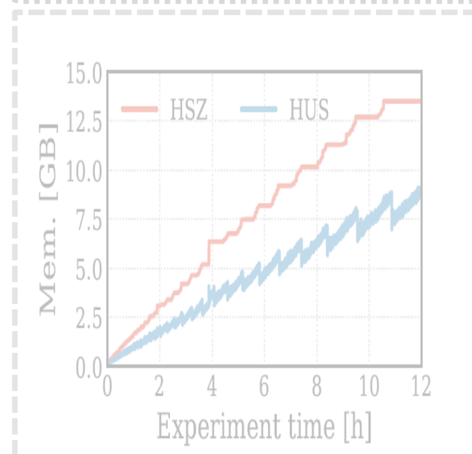
How often is the controller available?

Steady-state availability

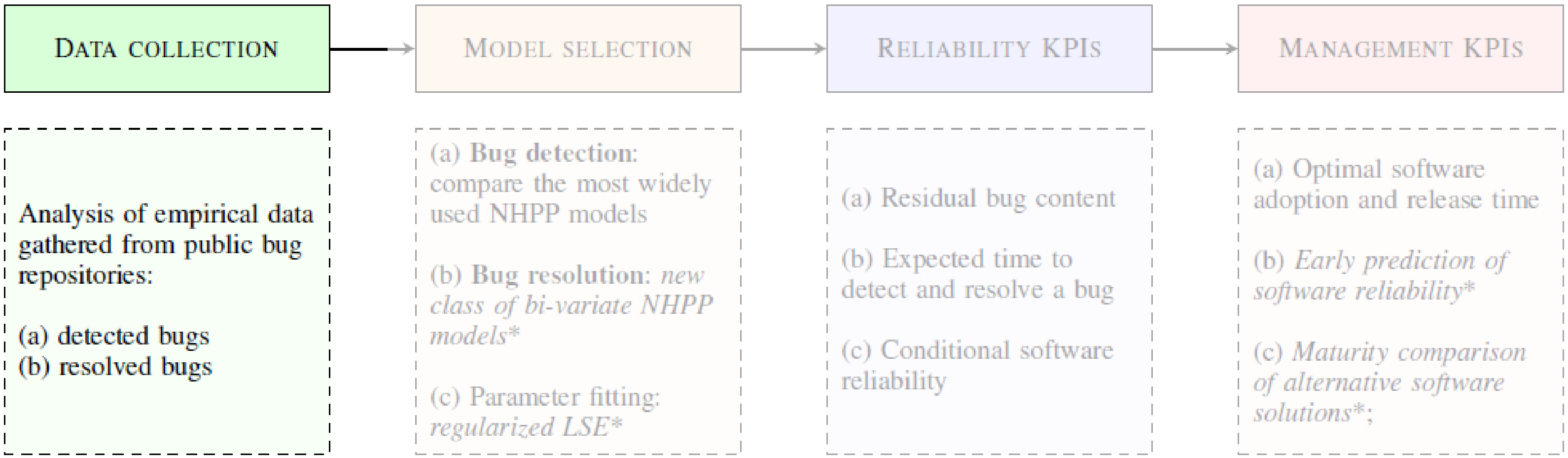


Do Softwarized networks age?

Proposed framework

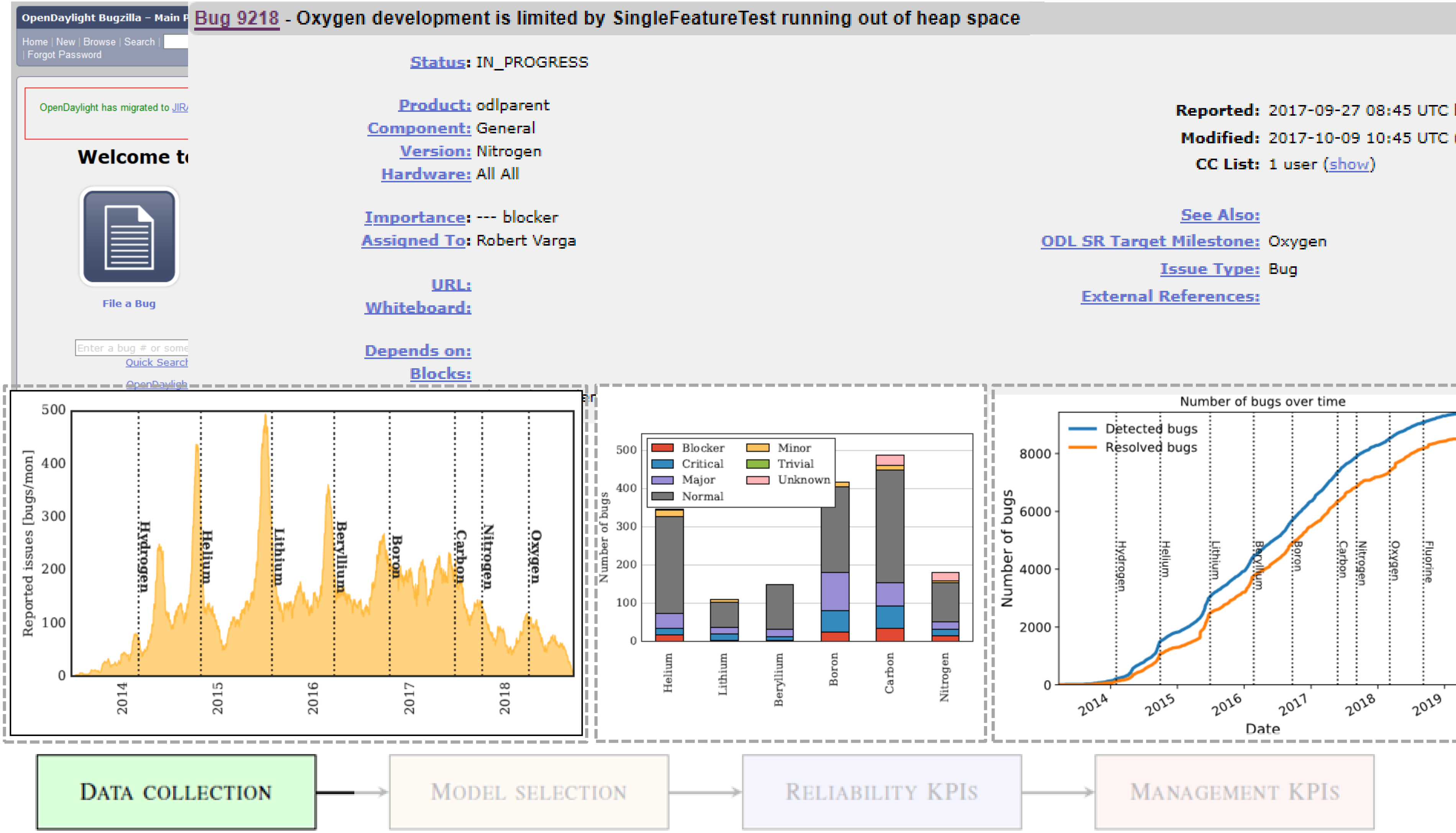


Failure Forecasting and Software Maturity



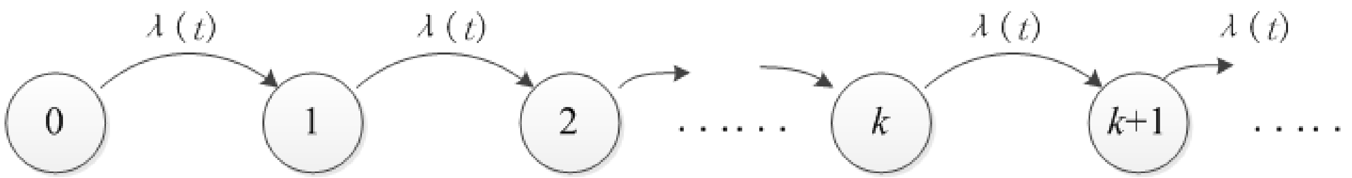
Vizarreta et al., *Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models*. TNSM, June 2018

Failure Forecasting and Software Maturity



Software Reliability Growth Models: *Theory*

Bug 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 of detecting a single bug (manifested SW fault) by time *t*

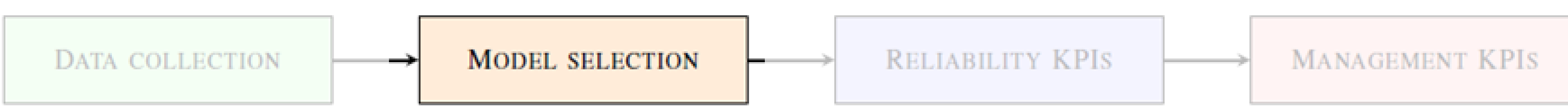
$$p = F(t)$$

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

- The cumulative number of detected bugs

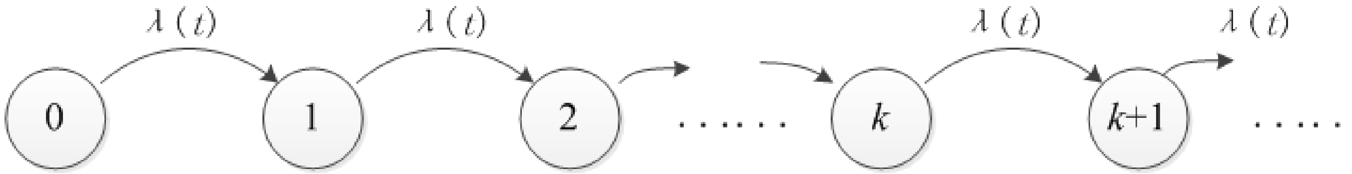
$$P(N(t) = k) = \frac{[aF(t)]^k}{k!} e^{-aF(t)}$$

- Expected number of detected bugs by time *t*



Software Reliability Growth Models: *Model selection*

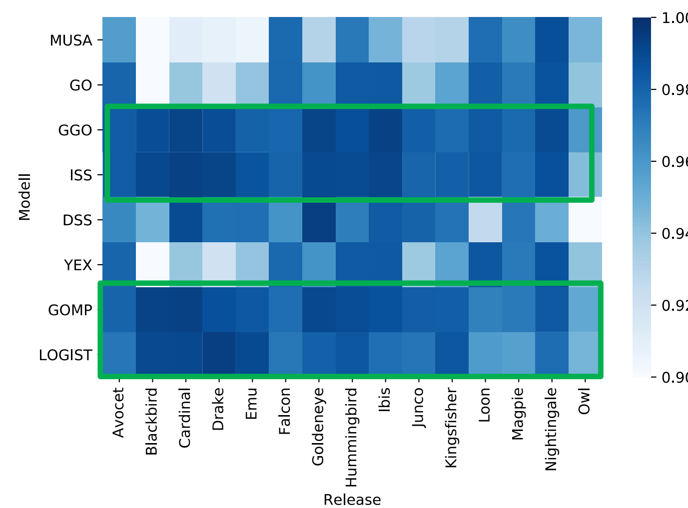
Bug detection as Non-Homogeneous Poisson Process (NHPP)

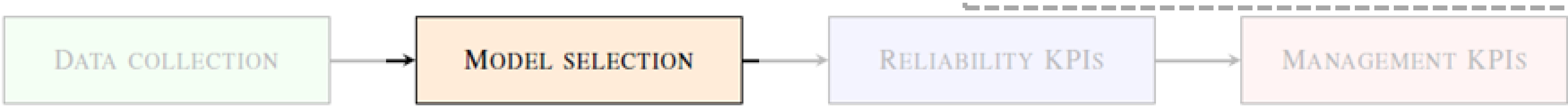


The eight most widely used NHPP models for modelling of the bug detection process are:

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	$m_{logist}(t) = \frac{a}{1+ke^{-bt}}$

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





P. Vizarreta, et a., An Empirical Study of Software Reliability in SDN Controllers, CNSM 2017

Software Reliability Growth Models: *Model Selection*

Bug resolution (R) is a combination of two processes: bug detection (D) and bug correction (C)

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

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

- Closed form solution exist only in trivial cases

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

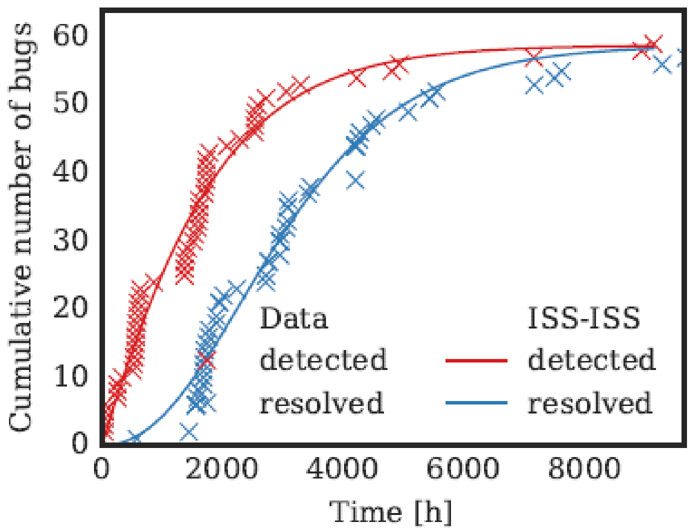
- PCA: Piecewise Constant Approximation is used for fitting instead

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

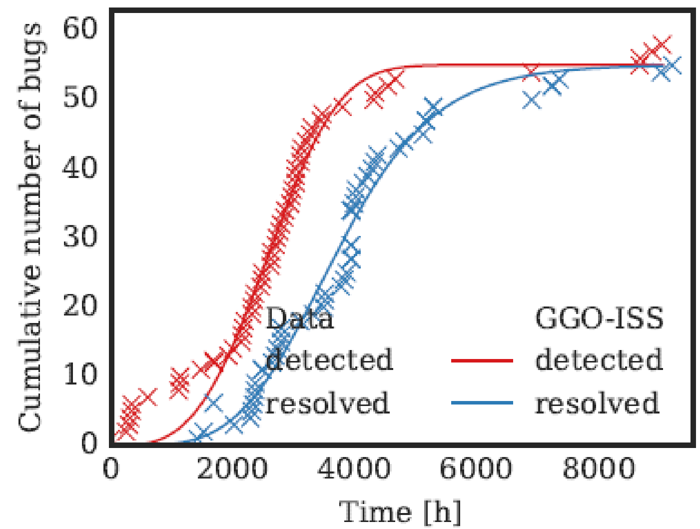
$$F_R(t) = \lim_{\Delta x \to 0} \widetilde{F}_R(t)$$

Data collection → **Model selection** → Reliability KPIs → Management KPIs

Best model selection

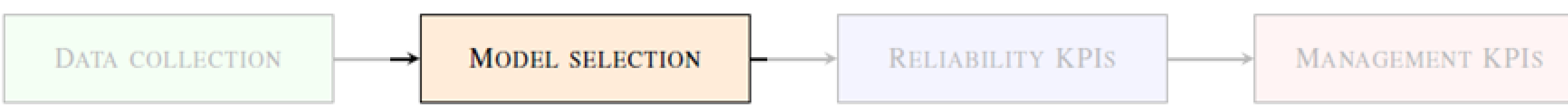


Avocet



Blackbird

The best fitting models for detected and resolved bugs may be different.



Reliability KPIs

Bug detection

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 detected bugs

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

Residual bug content

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

Conditional software reliability

Similarly for **Bug resolution**

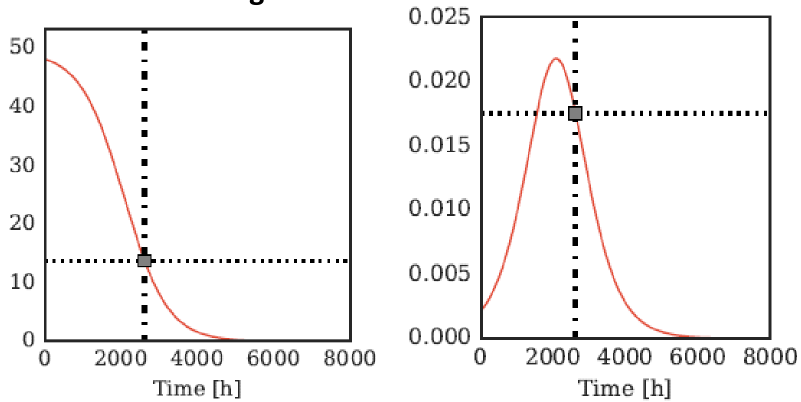
Data collection → Model selection → **Reliability KPIs** → Management KPIs

Management KPIs

Based on the selected model

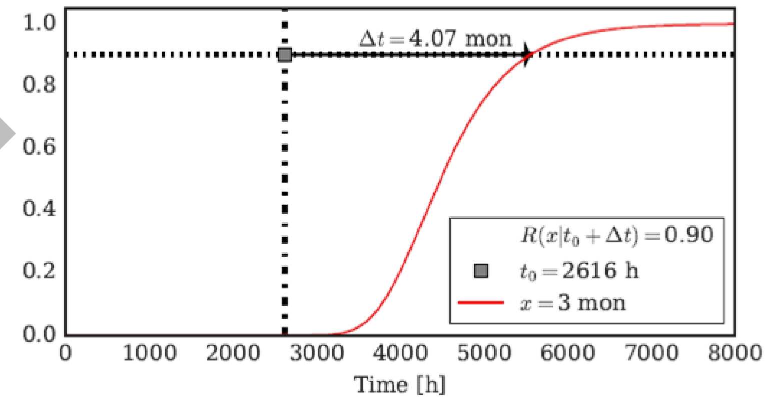
14 critical residual bugs

0.0175 bug/h ~ 2.38 days/bug



(a) Residual bug content $r(t)$ (b) Failure intensity $\lambda(t)$

Release adoption must be postponed 4 months for reliability of 0.9



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

ONOS Junco relase: 28.02.2017

3 months=2160 hours

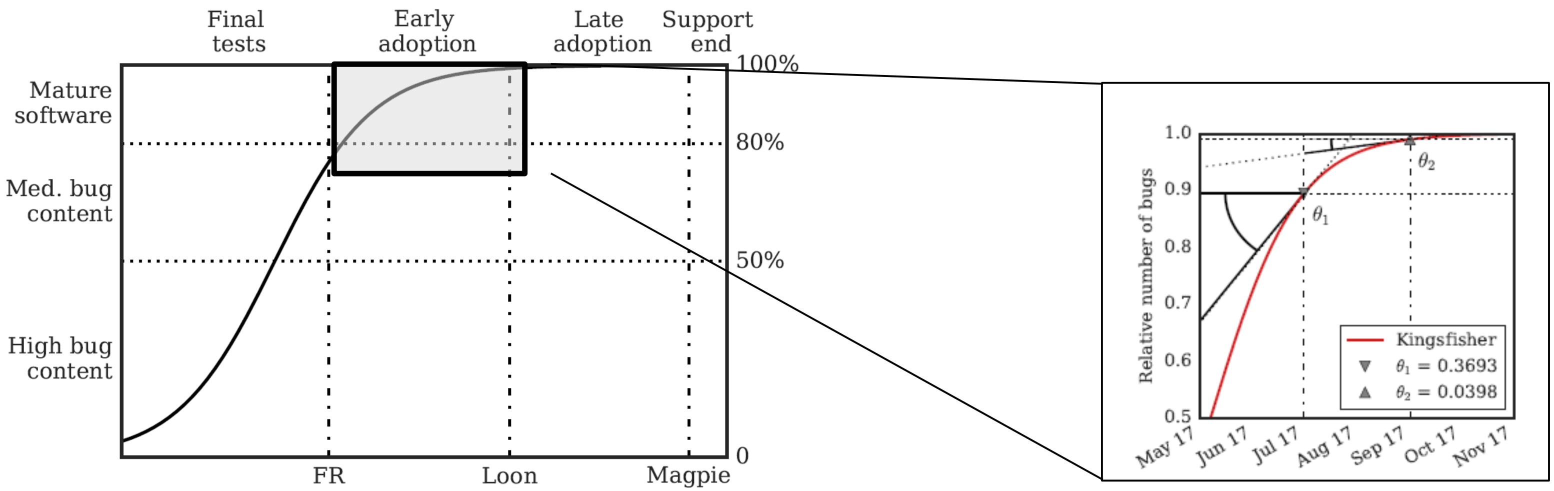
ONOS Kingsfisher release: 31.05.2017

DATA COLLECTION → MODEL SELECTION → RELIABILITY KPIS → **MANAGEMENT KPIS**

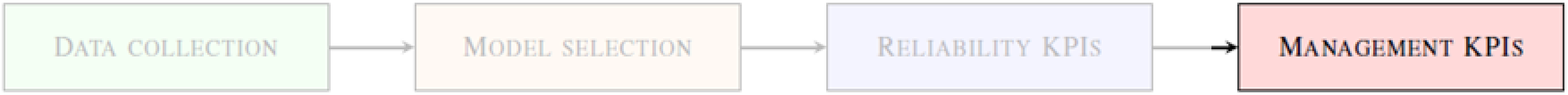
Management KPIs

Software Maturity Metric

- defined as the scaled gradient of the cumulative number of bugs, i.e., $\frac{\lambda(t)}{m_{max}}$.
- measures how far is the software from the stable region at any given moment.

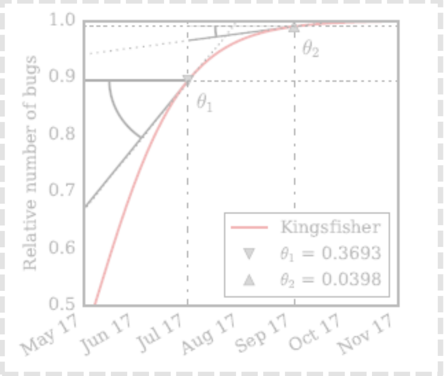


ONOS Kingsfisher final release (FR): June 2017
ONOS Loon release: September 2017

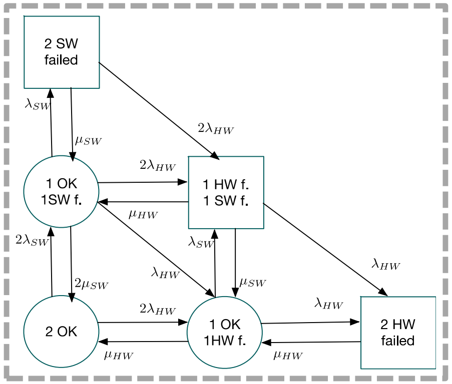


Software Dependability Problem

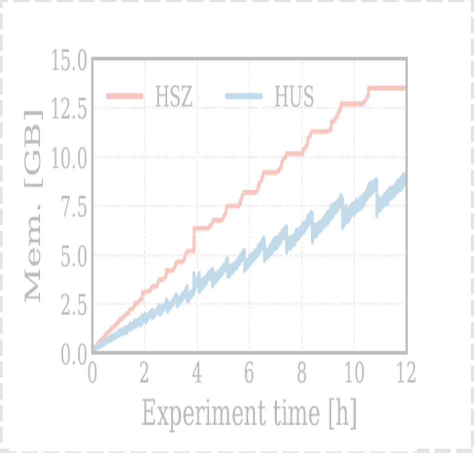
How often does SW fail?
Failure forecasting and Software Maturity



How often is the controller available?
Steady-state availability



Do Softwarized networks age?
Proposed framework



Steady State Availability

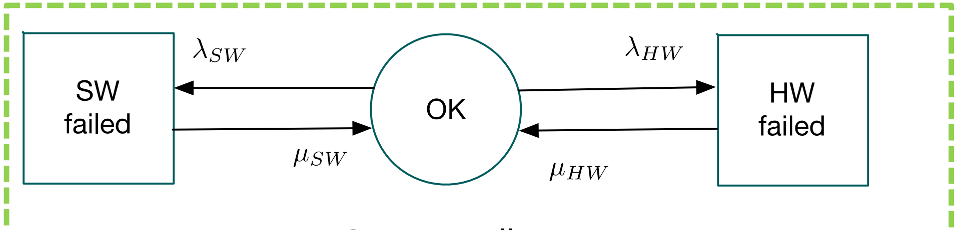
Homogeneous Markov Chains

- Single failure modes
- Usual assumptions
    - $\lambda_{HW} < \lambda_{SW}$
    - $\mu_{HW} > \mu_{SW}$
    - $\mu_x >> \lambda_x$
- Failure shadows

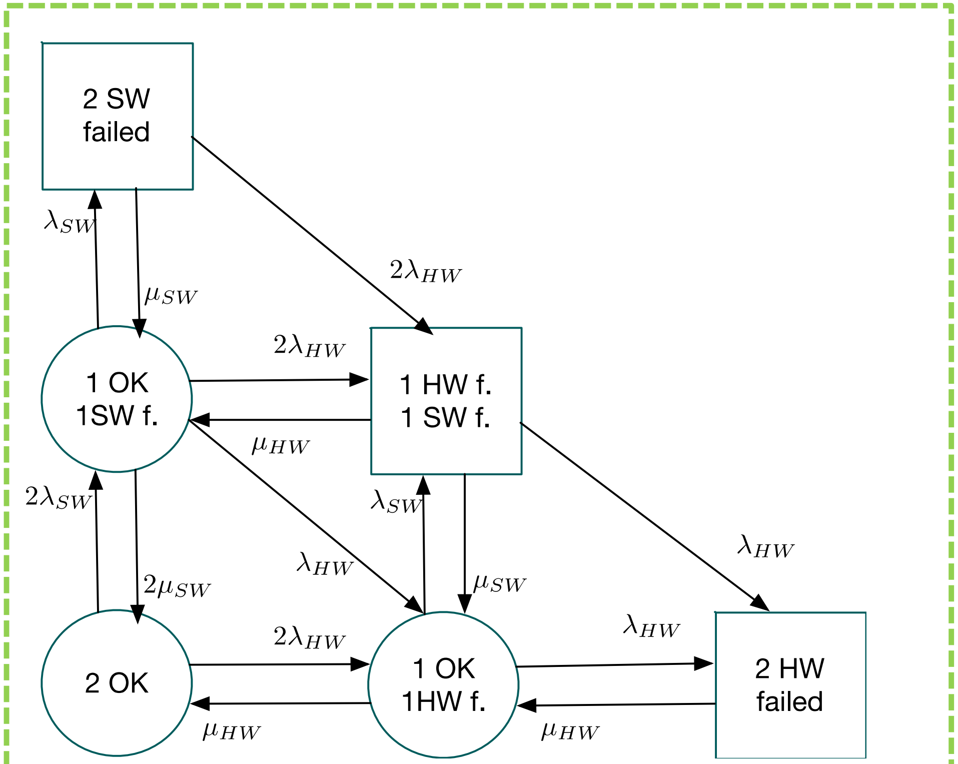
$$\sum_{\forall i} \hat{p}_i = 1$$

$$\hat{P} = \hat{P}.\mathcal{T}$$

$$A = \sum_{i \epsilon \Omega_W} \hat{p}_i$$



One controller

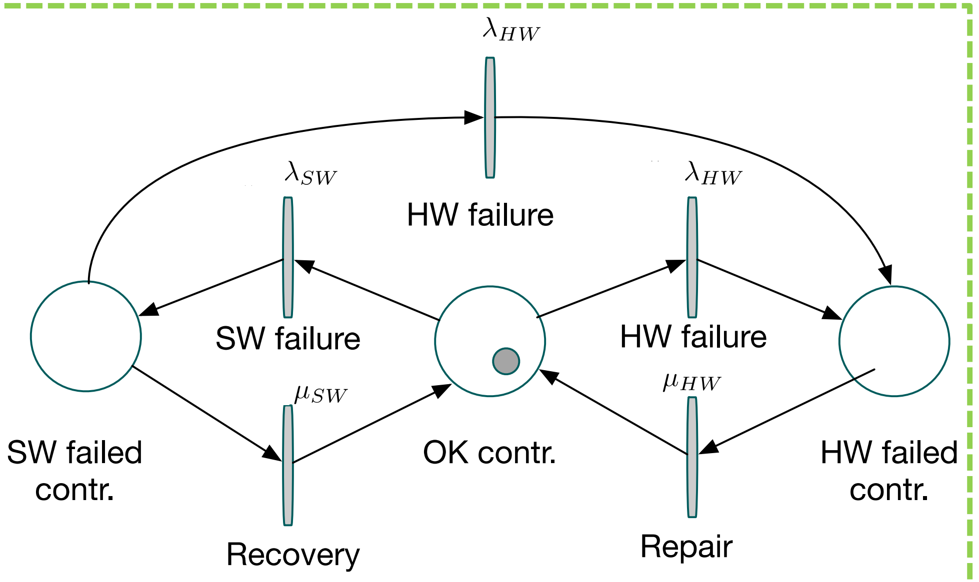


Two independent controllers with a common HW repair facility

Steady State Availability

Stochastic Petri Nets/ Stochastic Activity Networks (SANs)

- Single failure modes
- Usual assumptions
  - $\lambda_{HW} < \lambda_{SW}$
  - $\mu_{HW} > \mu_{SW}$

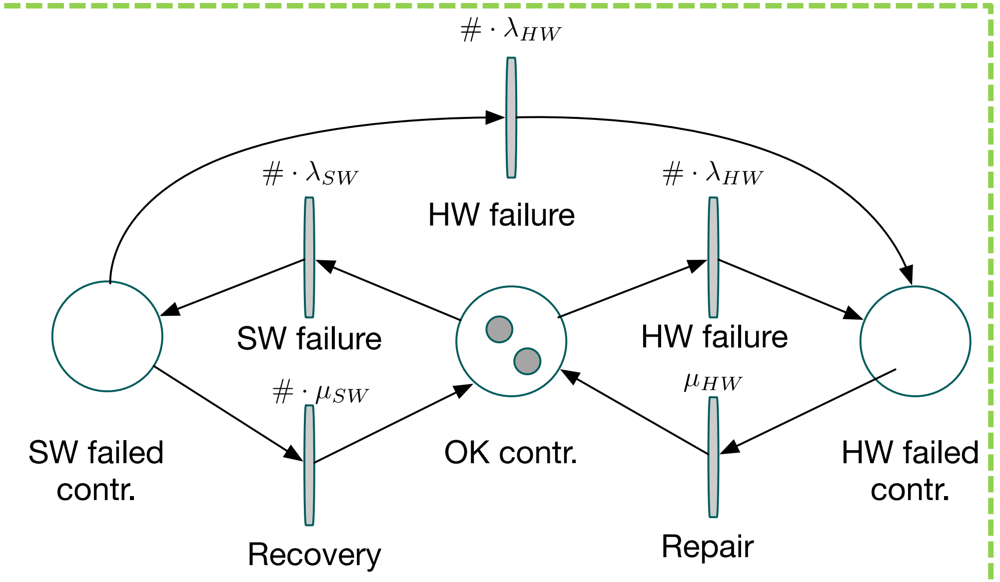


Stochastic Petri Net / SAN model for one controller

Steady State Availability

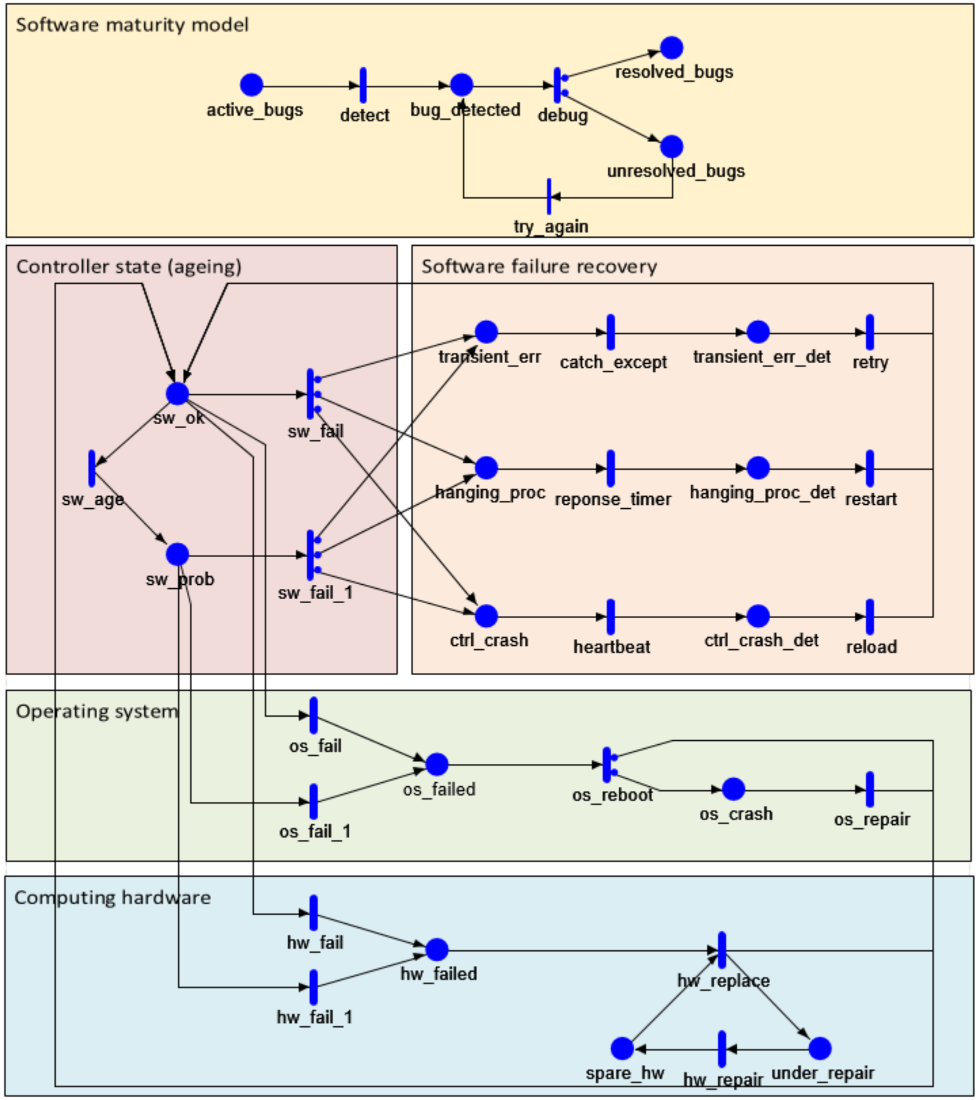
Stochastic Petri Nets/ Stochastic Activity Networks (SANs)

- Single failure modes
- Usual assumptions
  - $\lambda_{HW} < \lambda_{SW}$
  - $\mu_{HW} > \mu_{SW}$



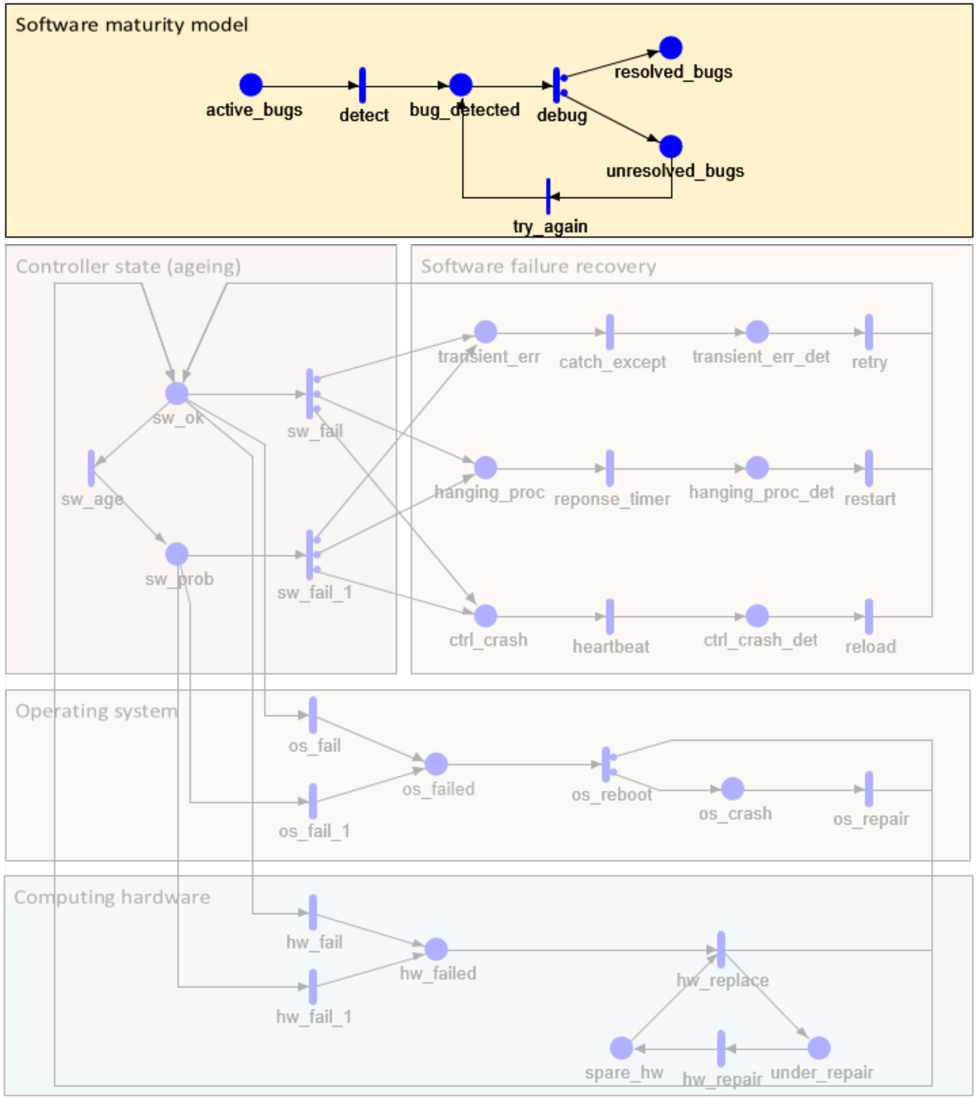
Stochastic Petri Net / SAN model with an arbitrary number # of controllers

SRN: A Case Study on SDN Controllers



Stojsavljevic, Petra; Heegaard, Poul; Helvik, Bjarne; Kellerer, Wolfgang; Mas Machuca, Carmen: Characterization of Failure Dynamics in SDN Controllers. RNDM, 2017

SRN: A Case Study on SDN Controllers



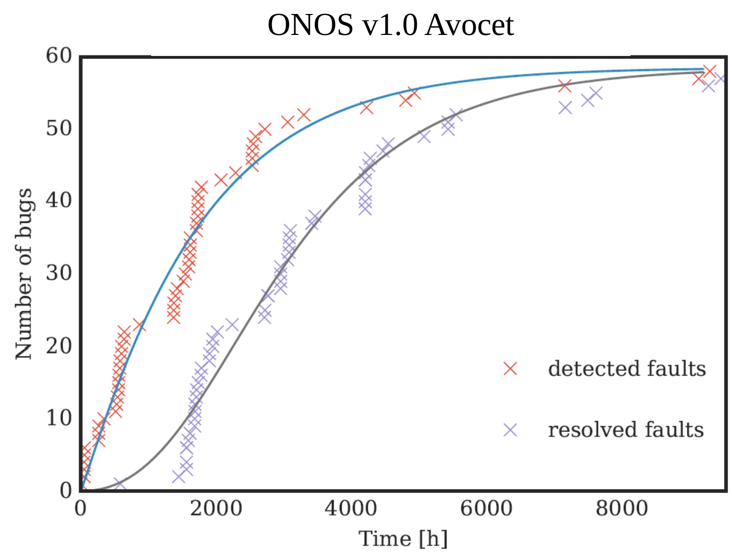
1. Software reliability growth

long term variations of software reliability

Software maturity model

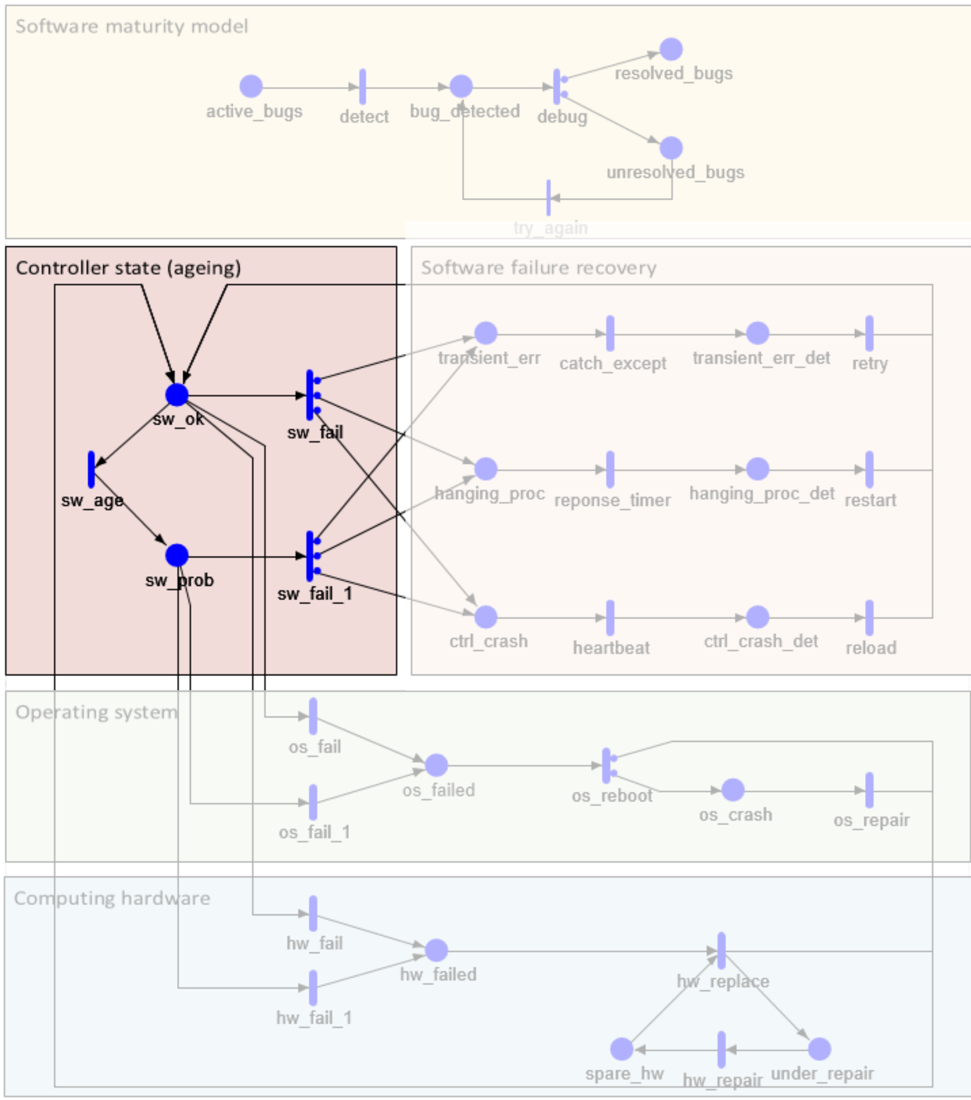
Parameter	Description	Baseline value
N_{bugs}	Initial number of active bugs	60
p_{debug}	Debugging success rate	0.99
$\lambda^{-1}_{bug\_detect}$	Bug detection rate	60 days
μ^{-1}_{debug}	Debug rate	60 days

- Model: Jelinski-Moranda with imperfect debugging



Stojsavljevic, Petra; Heegaard, Poul; Helvik, Bjarne; Kellerer, Wolfgang; Mas Machuca, Carmen: Characterization of Failure Dynamics in SDN Controllers. RNDM, 2017

SRN: A Case Study on SDN Controllers



2. Software aging

short term variations of software reliability

Parameter	Description	Baseline value
$\varphi^{-1}_{sw\_fail}$	Baseline software failure rate	7 days
$\lambda^{-1}_{sw\_age}$	Rate of software ageing	1 day
$\lambda^{-1}_{age\_fail}$	Ageing failure rate	7 days
p_{retry}(ok/prob)	Failures recovered by retry	0.15, 0.15
$p_{restart}$ (ok/prob)	Failures recovered by restart	0.15, 0.70
p_{reload}(ok/prob)	Failures requiring reload	0.70, 0.15

Failure frequency rate depends on controller state:

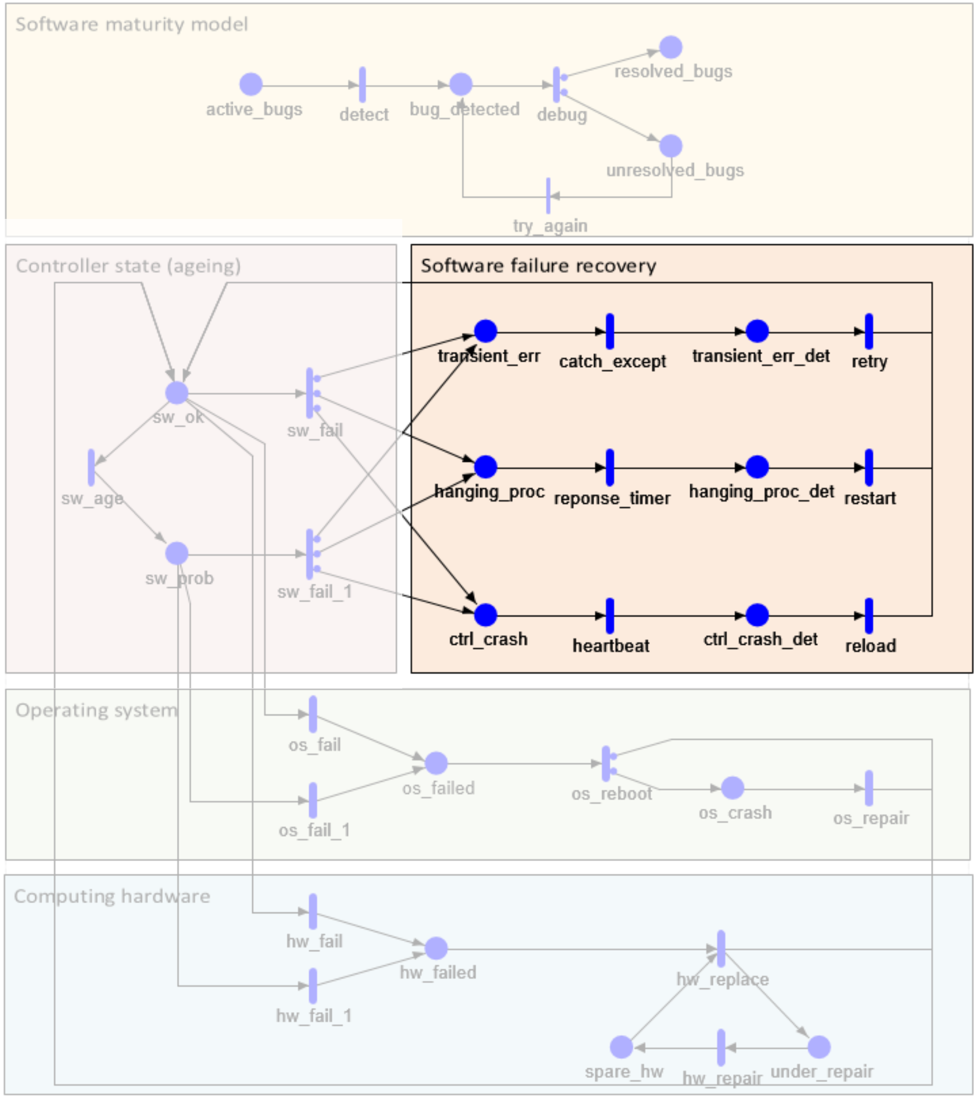
Aging rate

λ_{sw_age}

- highly robust state ***sw_ok***
- vulnerable state ***sw_prob***

Stojsavljevic, Petra; Heegaard, Poul; Helvik, Bjarne; Kellerer, Wolfgang; Mas Machuca, Carmen: Characterization of Failure Dynamics in SDN Controllers. RNDM, 2017

SRN: A Case Study on SDN Controllers



3. Nature of failures

Parameter	Description	Baseline value
μ_{catch}^{-1}	Catch the exception	1 msec
$\mu_{timeout}^{-1}$	Detect hanging process	1 sec
$\mu_{heartbeat}^{-1}$	Detecting controller crash	10 sec
μ_{retry}^{-1}	Retry the operation	0.5 sec
$\mu_{proc\_restart}^{-1}$	Process restart	5 min
μ_{reload}^{-1}	Restart controller and reload	30 min

Transient failures

- detected by catch-except routine
- mitigated by retrying the operation

Hanging failures

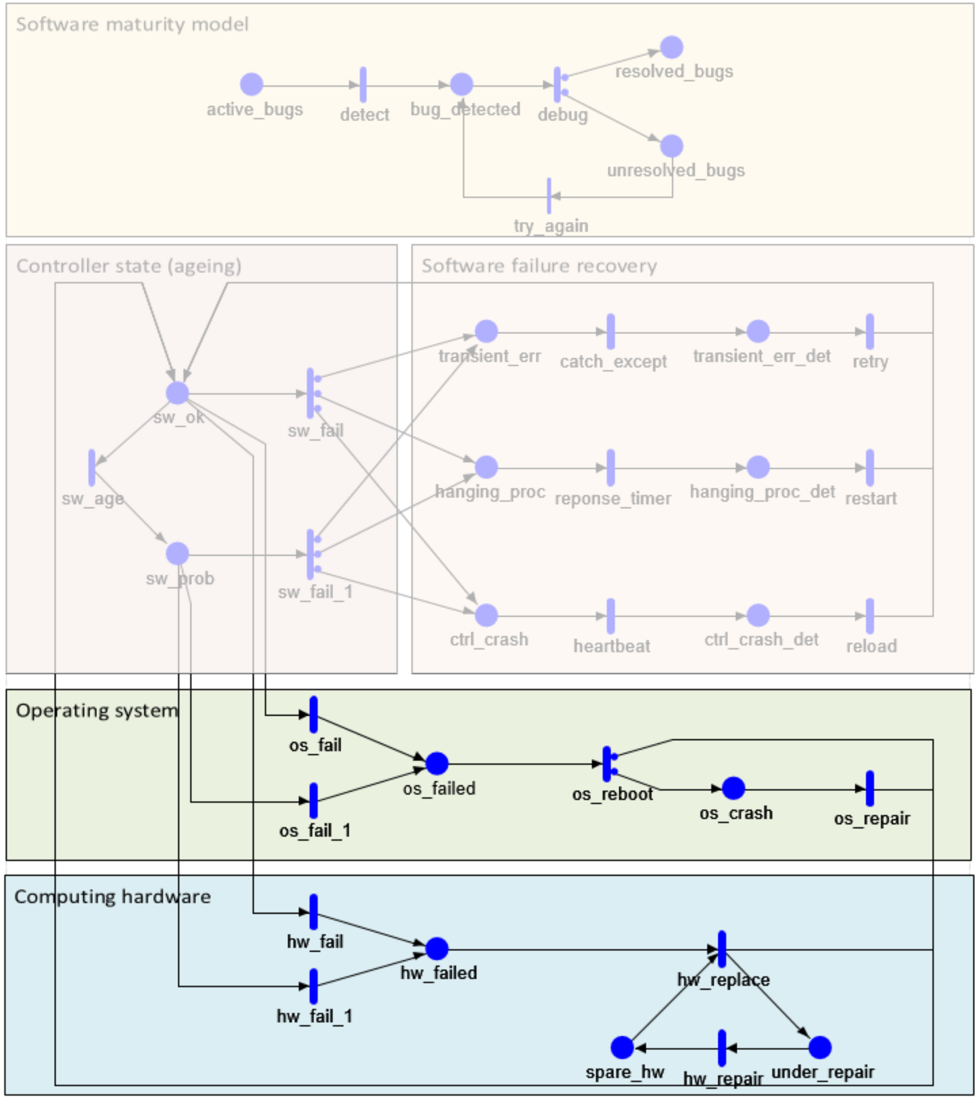
- detected by response timers
- mitigated by bundle restart

Crash failures

- detected by heartbeat messages
- controller software reloaded from the last checkpointed (saved) state

Stojsavljevic, Petra; Heegaard, Poul; Helvik, Bjarne; Kellerer, Wolfgang; Mas Machuca, Carmen: Characterization of Failure Dynamics in SDN Controllers. RNDM, 2017

SRN: A Case Study on SDN Controllers



4. Operating system

Failures of operating system (OS)

Parameter	Description	Baseline value
$\lambda^{-1}_{os\_fail}$	Mean time between OS failures	60 days
p_{os_reboot}	Success of OS reboot	0.9
$\mu^{-1}_{os\_reboot}$	OS reboot time	10 min
$\mu^{-1}_{os\_repair}$	OS repair time	1 h

5. General purpose Hardware

Failures of computing hardware (HW)

Parameter	Description	Baseline value
$\lambda^{-1}_{hw\_fail}$	Mean time between HW failures	6 months
$\mu^{-1}_{hw\_replace}$	HW replace time	2 hours
$\mu^{-1}_{hw\_repair}$	HW repair time	24 hours
N_{spare_hw}	Spare computing hardware	1

Stojsavljevic, Petra; Heegaard, Poul; Helvik, Bjarne; Kellerer, Wolfgang; Mas Machuca, Carmen: Characterization of Failure Dynamics in SDN Controllers. RNDM, 2017

Evaluation of SDN controller

Steady state availability

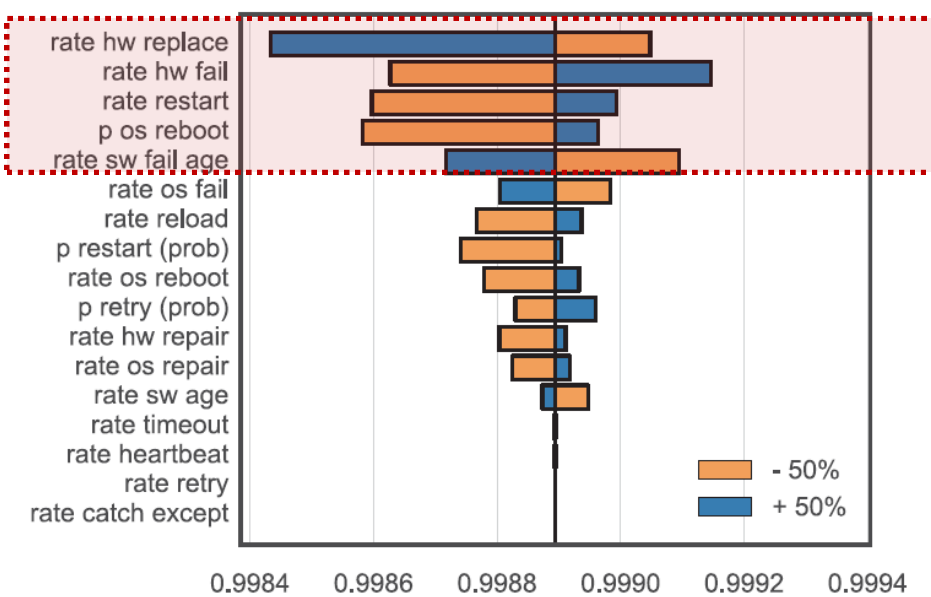
- At least two controllers are needed to achieve "3-nines" availability

Component	Controller	SW	OS	HW
Availability	0.99889	0.99956	0.99981	0.99951

[Ros14] assumed much higher availability of SDN controller A > 0.999975

Further study on clustering: imperfect failover and state synchronisation

- Identification of the most critical parameters (local sensitivity analysis)



Critical parameters

a) External failure rates (well studied and documented)

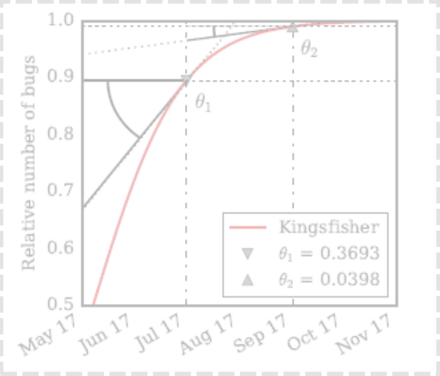
b) <u>Software aging rate</u> (uncertain, load dependant)

A comprehensive study on software aging

Software Dependability Problem

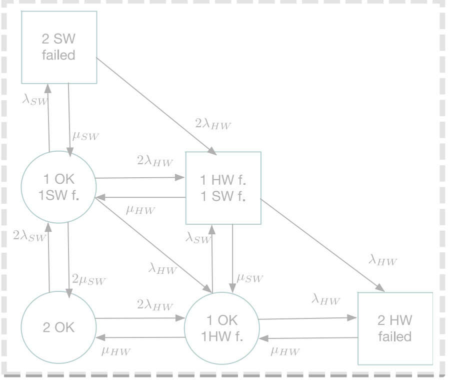
How often does SW fail?

Failure forecasting and Software Maturity



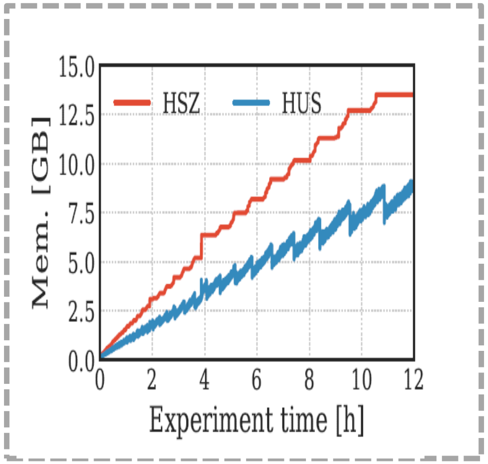
How often is the controller available?

Steady-state availability



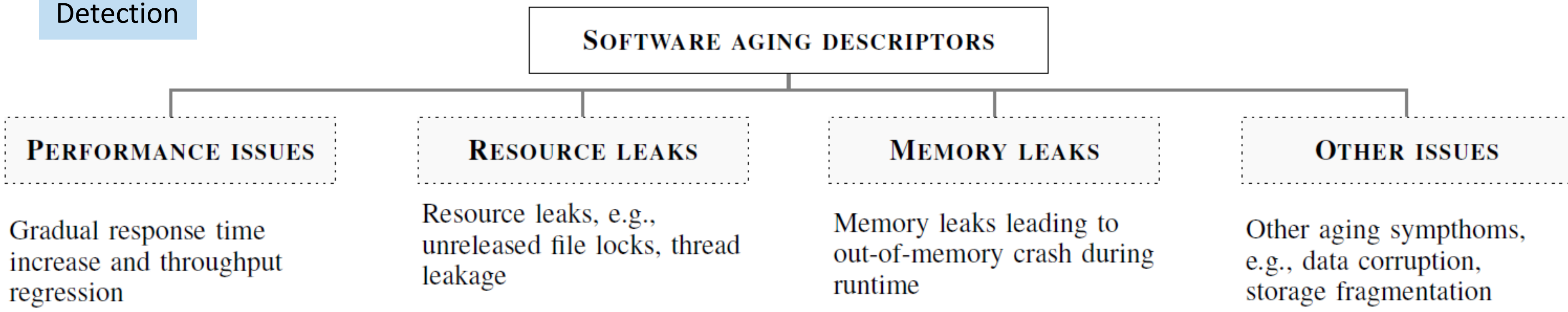
Do Softwarized networks age?

Proposed framework



Software Aging

Detection



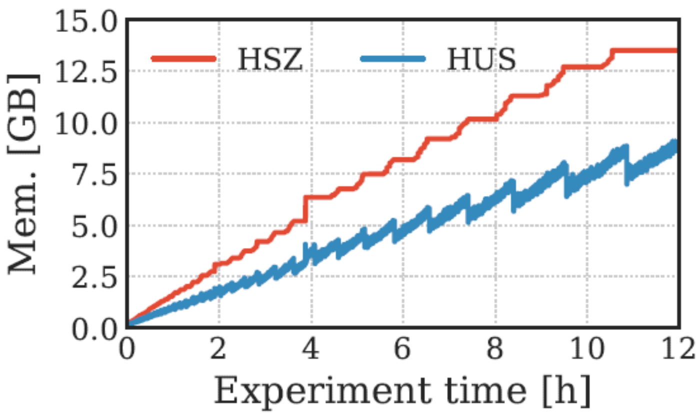
Not all are due to bugs (undesired behaviour of the code that should be corrected), but rather a deliberate design decision

E.g., In ONOS, when flow rules are added and removed, they are not deleted from the controller datastore; Instead, they are replaced with thumbstones (placeholders), to ensure stability of Gossip protocol. This also affects other eventually consistent network state primitives which rely on Gossip

Software Aging

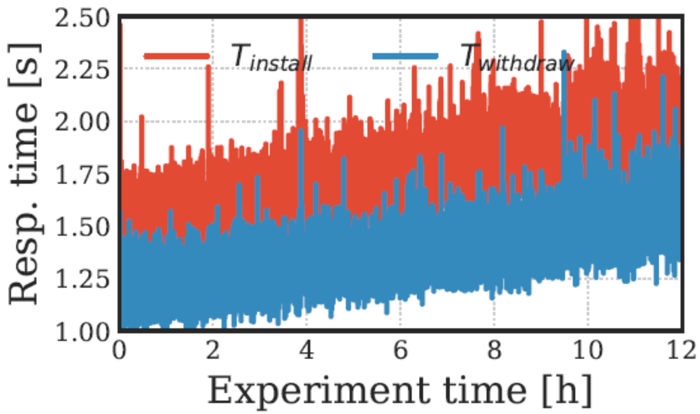
Impact evaluation

Aging observed at system level:



- Allocated heap (HSZ) and used heap memory (HUS) continuously grow
- System crashes after HSZ exhausts all 14 GB of available memory
- Crash happens after 18h at 300 intent/s

Aging observed at application level:



- ONOS response time increases linearly at constant workload
- Response time increases 50% for intent installation and withdrawal after the first day of operation

Software Aging

Prevention

Design of software rejuvenation policies

Rejuvenation schedule

Time-based

Rejuvenation carried out periodically:

→ Fixed time intervals

→ Count-based: counting critical requests

→ Workload-based: at low workload periods

Threshold-based

Executed before given resource utilization or performance degradation reach critical threshold:

→ Hard threshold

→ Projected utilization

Rejuvenation level

Application level

Application-level rejuvenation, mitigating a particular class of ageing effects:

→ Reinitialization of data structures, e.g., data store

→ Restarting network and/or database connections

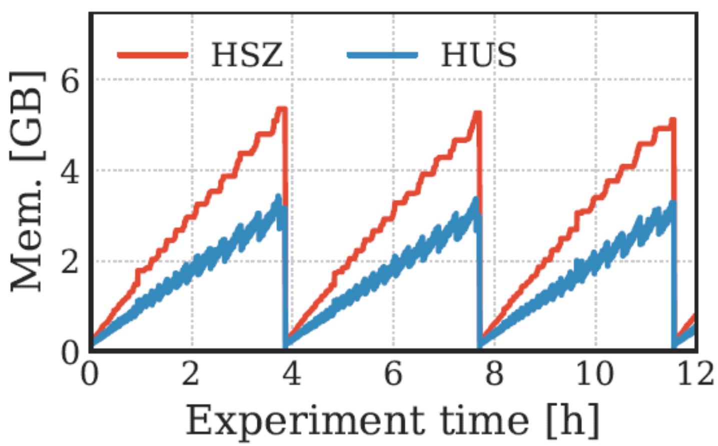
System level

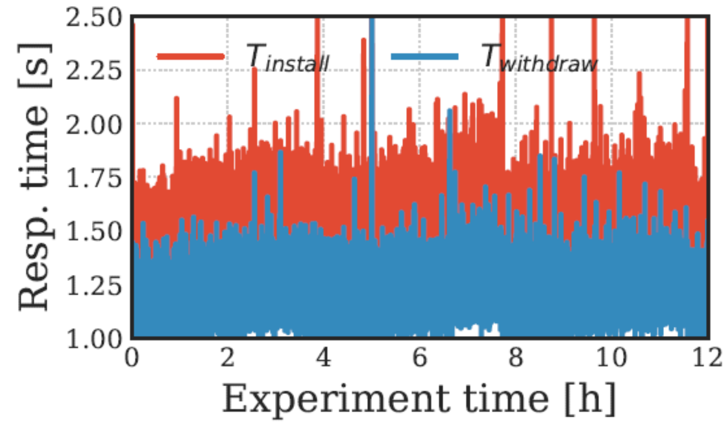
Rejuvenation of system environment:

→ Application reload

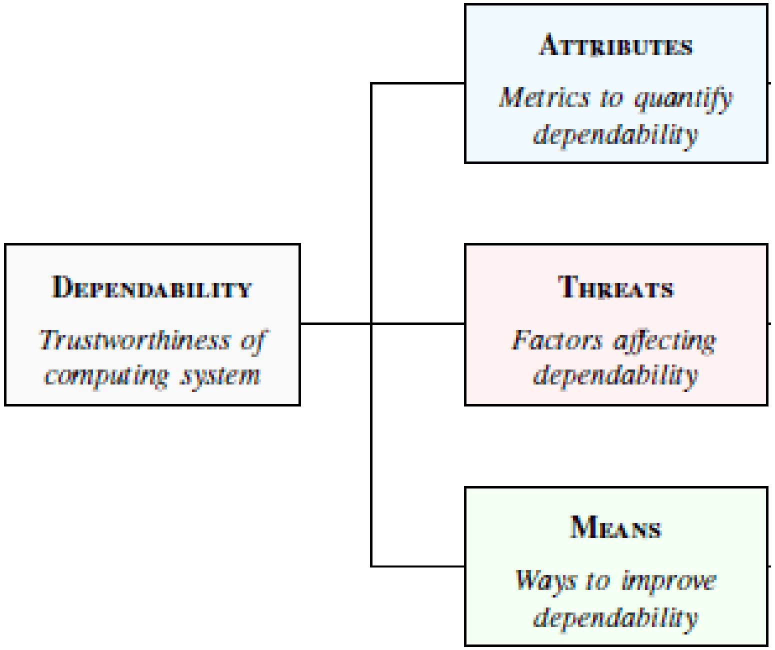
→ Restart of VMs or docker contatiners

→ Hardware reboot





Summary



More **metrics** are required to quatify the software dependability:

- Temporal reliability variations due to maturity and aging
- User-perceived service availability

Improved **threat** analysis to identify and classify software threats

Improved **threat** models and characterization

Software-aware **means**:

- (In)efficiency of software redundancy
- Network software rejuvenation

Questions?

Own related Publications

[J1] Vizarreta, Petra; Trivedi, Kishor; Helvik, Bjarne; Heegaard, Poul; Blenk, Andreas; Kellerer, Wolfgang; Mas Machuca, Carmen**,** ***Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models***. Transactions on Network and Service Management (TNSM), June 2018

[J2] Vizarreta, Petra; Van Bemten, Amaury; Sakic, Ermin; Abbasi, Khawar; Petroulakis, Nikolaos; Kellerer, Wolfgang; Mas Machuca, Carmen ***Incentives for softwarization of wind park communication networks****,* IEEE Communication Magazine, 2019

[J3] Vizarreta, Petra; Trivedi, Kishor; Mendiratta, Veena; Kellerer, Wolfgang; Mas Machuca, Carmen, ***DASON: Dependability Assurance Framework for Imperfect Distributed SDN Implementations***, Transactions on Network and Service Management (TNSM), Volume: 17, Issue: 2, June 2020

[J4] Vizarreta, Petra; Sieber, Christian; Blenk, Andreas; Van Bemten, Amaury; Ramachandra, Vinod; Kellerer, Wolfgang; Mas-Machuca, Carmen; Trivedi, Kishor., ***ARES: A Framework for Management of Software Ageing and Rejuvenation in SDN*** , Transactions on Network and Service Management (TNSM), October 2020

[C1] Stojsavljevic, Petra; Trivedi, Kishor; Helvik, Bjarne; Heegaard, Poul; Kellerer, Wolfgang; Mas Machuca, Carmen**,** ***An Empirical Study of Software Reliability in SDN Controllers***. CNSM, Tokyo, Japan, 2017

[C2] Vizarreta, Petra; Sakic, Ermin; Kellerer, Wolfgang; Mas Machuca, Carmen ***Mining Software Repositories for Predictive Modelling of Defects in SDN Controller***, In Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM), April 2019

[C3] tojsavljevic, Petra; Heegaard, Poul; Helvik, Bjarne; Kellerer, Wolfgang; Mas Machuca, Carmen, ***Characterization of Failure Dynamics in SDN Controllers***. In Proc. of IEEE Int. Workshop on Reliable Networks Design and Modeling, Alghero, Italy, 2017

.

References (I)

SDN controllers

[Ros14] F. J. Ros and P. M. Ruiz, “Five nines of southbound reliability in software-defined networks,” in Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014, pp. 31–36.

[Onos17] ON.Lab, “ONOS: Open Neetwork Operating System,” http://onosproject.org/, 2017.

[Odl17] Linux Foundation, “Opendaylight.” [Online]. Available: https://www.opendaylight.org/

Modelling approach

[SAN01] W. H. Sanders and J. F. Meyer, “Stochastic activity networks: Formal definitions and concepts,” in Lectures on Formal Methods and PerformanceAnalysis. Springer, 2001, pp. 315–343.

[Möb00] D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders, “Möbius: An extensible tool for performance and dependability modeling,” in International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Springer, 2000, pp. 332–336.

Reliability growth and ageing

[JM72] Z. Jelinski and P. B. Moranda, “Software reliability research,” Statistical Computer Performance Evaluation, pp. 465–484, 1972.

[Huang95] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, “Software rejuvenation: Analysis, module and applications,” in Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on. IEEE, 1995, pp. 381–390.

References (II)

Fault mitigation

[Qin05] F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, “Rx: treating bugs as allergies—a safe method to survive software failures,” in Acm sigops operating systems review, vol. 39, no. 5. ACM, 2005, pp. 235–248.

[Gro07] M. Grottke and K. S. Trivedi, “Fighting bugs: Remove, retry, replicate, and rejuvenate,” Computer, vol. 40, no. 2, 2007.

[Tri10] K. S. Trivedi, M. Grottke, and E. Andrade, “Software fault mitigation and availability assurance techniques,” International Journal of System Assurance Engineering and Management, vol. 1, no. 4, pp. 340–350, 2010.

Model parameters

[Kim09] D. S. Kim, F. Machida, and K. S. Trivedi, “Availability modeling and analysis of a virtualized system,” in Dependable Computing, 2009. PRDC’09. 15th IEEE Pacific Rim International Symposium on. IEEE, 2009, pp. 365–371.

[Nec16] G. Nencioni, B. E. Helvik, A. J. Gonzalez, P. E. Heegaard, and A. Kamisinski, “Availability modelling of software-defined backbone networks,” in *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 2016, pp. 105–112.