Software Dependability:
A case study on Software Defined Networks

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

©2020 Technical University of Munich
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 in the Northern Virginia (US-EAST-1) Region
https://aws.amazon.com/message/41926/

22.06.19 03:00 to 22.06.2019 05:42 (PST).
A widespread BGP routing leak affected a number of Internet services and a portion of traffic to Cloudflare.

26.03.20 16:14 to 27.03.2020 16:14 (PST).
Cloud IAM experienced a disruption across many services (resulting in continued failures and timeouts for a subset of services) for 3.5 hours.
https://status.cloud.google.com/incident/zall/20003

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
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

Dependability
- Trustworthiness of computing system

Threats
- Factors affecting dependability
  - Fault
  - Error
  - Failure

Attributes
- Metrics to quantify dependability
  - Availability
  - Reliability
  - Maintainability
  - Safety

Means
- Ways to improve dependability
  - Fault prevention
  - Fault removal
  - Fault tolerance
  - Fault forecasting

Source: IFIP WG10.4 Dependable Computing and Fault Tolerance https://www.dependability.org/wg10.4/
C. Mas Machuca (TUM) | Software Dependability
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.

**Fault dormancy**
- **Active**: it produces an error
- **Dormant**: it has not produced an error

**Error latency**
- **Detected**: it has manifestated as failure
- **Latent**: 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

C. Mas Machuca (TUM) | *Software Dependability*
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).
Software fault = bug

Types of software faults:

- Bohrbugs (deterministic): „solid“ logical faults
  - Remove

- Mandelbugs (non-deterministic): "relative" logical faults
  - Path Computation Element (PCE) able to create tunnel with negative bandwidth
  - Rejuvenate
  - Degradation with time
  - Ageing-related bugs

Fault handling strategies:

- Description
- Example

Terms and Taxonomy
Software fault = bug

Types of software faults:

**Bohrbugs (deterministic)**
- "solid“ logical faults
- Remove
- Path Computation Element (PCE) able to create tunnel with negative bandwidth

**Mandelbugs (non-deterministic)**
- "relative“ logical faults
- Retry, replicate
- Distributed database locking in ONOS

Terms and Taxonomy
Terms and Taxonomy

- **Software fault = bug**

- **Types of software faults:**

<table>
<thead>
<tr>
<th>Threats</th>
<th>Faults affecting dependability</th>
<th>Error</th>
<th>Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bohrbugs</strong> (deterministic)</td>
<td>&quot;solid&quot; logical faults</td>
<td>Remove</td>
<td>Path Computation Element (PCE) able to create tunnel with negative bandwidth</td>
</tr>
<tr>
<td><strong>Mandelbugs</strong> (non-deterministic)</td>
<td>&quot;relative&quot; logical faults</td>
<td>Retry, replicate</td>
<td>Distributed database locking in ONOS</td>
</tr>
<tr>
<td><strong>Aging-related bugs</strong></td>
<td>Degradation with time</td>
<td>Rejuvenate</td>
<td>Flows still reported in oper data store after they have been deleted from both config and network</td>
</tr>
</tbody>
</table>
Limitations of the State of the Art

**Threats**
- Factors affecting dependability

**Attributes**
- reliability, does not precisely describe software behaviour
  - Reliability growth due to maturity
  - Reliability degradation due to aging

**Means**
- Ways to improve dependability

---

**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/
C. Mas Machuca (TUM) | Software Dependability
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

Data Collection

- Analysis of empirical data gathered from public bug repositories:
  - (a) detected bugs
  - (b) resolved bugs

Model Selection

- (a) Bug detection: compare the most widely used NHPP models
- (b) Bug resolution: new class of bi-variate NHPP models*
- (c) Parameter fitting: regularized LSE*

Reliability KPIs

- (a) Residual bug content
- (b) Expected time to detect and resolve a bug
- (c) Conditional software reliability

Management KPIs

- (a) Optimal software adoption and release time
- (b) Early prediction of software reliability*
- (c) Maturity comparison of alternative software solutions*

Vizarreta et al., Assessing the Software Maturity of SDN Controllers Using Software Reliability Growth Models. TNSM, June 2018
C. Mas Machuca (TUM) | Software Dependability
Failure Forecasting and Software Maturity

DATA COLLECTION → MODEL SELECTION → RELIABILITY KPIs → MANAGEMENT KPIs
Software Reliability Growth Models: *Theory*

**Bug detection** as Non-Homogeneous Poisson Process (NHPP)

- Initial number of bugs $N$ is Poisson random variable with $E[N] = \alpha$.
- Probability of detecting a single bug (manifested SW fault) by time $t$.
- Assuming time to discover every bug is i.i.d. we have Bernoulli trials.
- The cumulative number of detected bugs.
- Expected number of detected bugs by time $t$.

\[
P(N = n) = \frac{\alpha^n}{n!} e^{-\alpha}
\]

\[
p = F(t)
\]
The eight most widely used NHPP models for modelling of the bug detection process are:

<table>
<thead>
<tr>
<th>Model</th>
<th>Shape</th>
<th>Mean value function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musa-Okumoto logarithmic</td>
<td>Concave</td>
<td>$m_{mo}(t) = a \ln(1 + bt)$</td>
</tr>
<tr>
<td>Goel-Okumoto exponential</td>
<td>Concave</td>
<td>$m_{go}(t) = a(1 - e^{-bt})$</td>
</tr>
<tr>
<td>Generalized Goel-Okumoto</td>
<td>S-shaped</td>
<td>$m_{gg}(t) = a(1 - e^{-bt})$</td>
</tr>
<tr>
<td>Ohba’s inflection S-shaped</td>
<td>S-shaped</td>
<td>$m_{is}(t) = a\frac{1-e^{-bt}}{1+ae^{-bt}}$</td>
</tr>
<tr>
<td>Yamada delayed S-shaped</td>
<td>S-shaped</td>
<td>$m_{ds}(t) = a(1 - (1 + bt)e^{-bt})$</td>
</tr>
<tr>
<td>Yamada exponential</td>
<td>Concave</td>
<td>$m_{ye}(t) = a(1 - e^{-\tau(1-e^{-bt})})$</td>
</tr>
<tr>
<td>Gompertz</td>
<td>S-shaped</td>
<td>$m_{gomp}(t) = ak^{b^t}$</td>
</tr>
<tr>
<td>Logistic</td>
<td>S-shaped</td>
<td>$m_{logist}(t) = \frac{a}{1+ke^{-bt}}$</td>
</tr>
</tbody>
</table>

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

\[
m_R^{g_0-g_0}(t) = a \left[ 1 - \frac{b_1 e^{-b_2 t} - b_2 e^{-b_1 t}}{b_1 - b_2} \right]
\]

- PCA: Piecewise Constant Approximation is used for fitting instead

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

\[
F_R(t) = \lim_{\Delta x \to 0} \tilde{F}_R(t)
\]
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 $m(t)$

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

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

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

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

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

Release adoption must be postponed 4 months for reliability of 0.9

ONOS Junco release: 28.02.2017
3 months = 2160 hours

ONOS Kingsfisher release: 31.05.2017

(a) Residual bug content \( r(t) \)
(b) Failure intensity \( \lambda(t) \)
(c) Software reliability \( R(x|t) \)

3 months = 2160 hours
Management KPIs

Software Maturity Metric

- defined as the scaled gradient of the cumulative number of bugs, i.e., \( \frac{\lambda(t)}{m_{\text{max}}} \).
- 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
  - $\lambda_{HW} < \lambda_{SW}$
  - $\mu_{HW} > \mu_{SW}$
  - $\mu_x \gg \lambda_x$
- Failure shadows

\[
\sum_{i} \hat{p}_i = 1
\]
\[
\hat{p} = \hat{p} \cdot T
\]

\[
A = \sum_{i \in \Omega_W} \hat{p}_i
\]
Steady State Availability

Stochastic Petri Nets/ Stochastic Activity Networks (SANs)

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

- 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
2. Software aging

short term variations of software reliability

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Baseline value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi_{sw_fail}^{-1}$</td>
<td>Baseline software failure rate</td>
<td>7 days</td>
</tr>
<tr>
<td>$\lambda_{sw_age}^{-1}$</td>
<td>Rate of software ageing</td>
<td>1 day</td>
</tr>
<tr>
<td>$\lambda_{age_fail}^{-1}$</td>
<td>Ageing failure rate</td>
<td>7 days</td>
</tr>
<tr>
<td>$p_{retry,(ok/prob)}$</td>
<td>Failures recovered by retry</td>
<td>0.15, 0.15</td>
</tr>
<tr>
<td>$p_{restart,(ok/prob)}$</td>
<td>Failures recovered by restart</td>
<td>0.15, 0.70</td>
</tr>
<tr>
<td>$p_{preload,(ok/prob)}$</td>
<td>Failures requiring reload</td>
<td>0.70, 0.15</td>
</tr>
</tbody>
</table>

Failure frequency rate depends on controller state:

- **highly robust state** `sw_ok`
- **vulnerable state** `sw_prob`
3. Nature of failures

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Baseline value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu^{-1}_{\text{catch}}$</td>
<td>Catch the exception</td>
<td>1 msec</td>
</tr>
<tr>
<td>$\mu^{-1}_{\text{timeout}}$</td>
<td>Detect hanging process</td>
<td>1 sec</td>
</tr>
<tr>
<td>$\mu^{-1}_{\text{heartbeat}}$</td>
<td>Detecting controller crash</td>
<td>10 sec</td>
</tr>
<tr>
<td>$\mu^{-1}_{\text{retry}}$</td>
<td>Retry the operation</td>
<td>0.5 sec</td>
</tr>
</tbody>
</table>
| $\mu^{-1}_{\text{proc.
restart}}$ | Process restart                  | 5 min          |
| $\mu^{-1}_{\text{reload}}$   | 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
SRN: A Case Study on SDN Controllers

4. Operating system

Failures of operating system (OS)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Baseline value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_{os_fail}^{-1}$</td>
<td>Mean time between OS failures</td>
<td>60 days</td>
</tr>
<tr>
<td>$p_{os_reboot}$</td>
<td>Success of OS reboot</td>
<td>0.9</td>
</tr>
<tr>
<td>$L_{os_reboot}^{-1}$</td>
<td>OS reboot time</td>
<td>10 min</td>
</tr>
<tr>
<td>$L_{os_repair}^{-1}$</td>
<td>OS repair time</td>
<td>1 h</td>
</tr>
</tbody>
</table>

5. General purpose Hardware

Failures of computing hardware (HW)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Baseline value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_{hw_fail}^{-1}$</td>
<td>Mean time between HW failures</td>
<td>6 months</td>
</tr>
<tr>
<td>$L_{hw_replace}^{-1}$</td>
<td>HW replace time</td>
<td>2 hours</td>
</tr>
<tr>
<td>$L_{hw_repair}^{-1}$</td>
<td>HW repair time</td>
<td>24 hours</td>
</tr>
<tr>
<td>$N_{spare_hw}$</td>
<td>Spare computing hardware</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Controller</th>
<th>SW</th>
<th>OS</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>0.99889</td>
<td>0.99956</td>
<td>0.99981</td>
<td>0.99951</td>
</tr>
</tbody>
</table>

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

[Ros14] assumed much higher availability of SDN controller 

A > 0.999975

Further study on clustering: imperfect failover and state synchronisation

**Critical parameters**

a) External failure rates (well studied and documented)

b) Software aging rate (uncertain, load dependant)

A comprehensive study on software aging

C. Mas Machuca (TUM) | *Software Dependability*
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*
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
Impact evaluation

Software Aging

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 containers
  - Hardware reboot

Graphs showing memory usage over experiment time for HSZ and HUS, and response time distribution over experiment time with $T_{install}$ and $T_{withdraw}$.
More **metrics** are required to quantify 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

References (I)

SDN controllers


Modelling approach


Reliability growth and ageing


References (II)

Fault mitigation


Model parameters
