Virtually Eliminating Router Bugs

Eric Keller, Minlan Yu, Matt Caesar, Jennifer Rexford
Princeton University, UIUC

NANOG 46: Philadelphia, PA
Dealing with router bugs

• Internet’s complexity implemented in software running on routers

• Complexity leads to bugs

• String of high profile vulnerabilities, outages
Feb. 16, 2009: SuproNet

- Announced a single prefix to a single provider
  - huge increase in the global rate of updates
  - 10x increase in global instability for an hour
  - 1 misconfiguration tickled 2 bugs (2 vendors)

Source: Renesys

Misconfiguration:
as-path prepend 47868

Did not filter

AS path
Prepending
After: len > 255

MikroTik bug:
no-range check

Cisco bug:
Long AS paths

AS47878 prepended 252 times AS29113
Challenges of router bugs

• Bugs different from traditional failures
  – Violate protocol, cause cascading outages, need to wait for vendor to repair, some exploitable by attackers

• Problem is getting worse
  – Increasing demands on routing, vendors allowing 3rd party development, other sources of outage becoming less common
Building bug-tolerant routers

• Our approach: run multiple, diverse instances of router software in parallel

• Instances “vote” on FIB contents, update messages sent to peers
Bug-tolerant Router Architecture

- Hypervisor:
  - Distributes received messages
  - Votes on updates (to FIB or to peer)
  - Maintains replicas (hiding churn)
• Hypervisor:
  - Distributes received messages
  - Votes on updates (to FIB or to peer)
  - Maintains replicas (hiding churn)
Bug-tolerant Router Architecture

- Hypervisor:
  - Distributes received messages
  - Votes on updates (to FIB or to peer)
  - Maintains replicas (hiding churn)
Bug-tolerant Router Architecture

- Hypervisor:
  - Distributes received messages
  - Votes on updates (to FIB or to peer)
  - Maintains replicas (hiding churn)
Voting Algorithms

• Wait-for-consensus: handling transience
  – Output when a majority of instances agree

• Master-Slave: speeding reaction time
  – Output Master’s answer
  – Slaves used for detection
  – Switch to slave on buggy behavior

• Continuous Majority: hybrid
  – Voting rerun when any instance sends an update
“We did this in the 1970s...”

• Yes, it’s an old idea applied to routing
  – new opportunities: e.g., small dependence on past, ability to correct mistakes
  – new challenges: e.g., transient behavior may legitimately differ, need fast reaction time

• Plus, it’s not just “N-version programming”
  – Can also diversify execution environment
Achieving Diversity

• If not N-version programming...

• Where does diversity come from?

<table>
<thead>
<tr>
<th>Type of diversity</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Environment</td>
<td>Operating system, memory layout</td>
</tr>
<tr>
<td>Software Diversity</td>
<td>Version (0.98 vs 0.99), implementation (Quagga vs XORP)</td>
</tr>
<tr>
<td>Data Diversity</td>
<td>Configuration, timing of updates/connections</td>
</tr>
</tbody>
</table>

• Next: How effective are these?
### Achieving Diversity

- **General Diversity** (e.g., OS, mem space layout)
  - Not studied here

- **Data Diversity**
  - Taxonomized XORP and Quagga bug database

<table>
<thead>
<tr>
<th>Diversity Mechanism</th>
<th>Bugs avoided (est.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing/order of messages</td>
<td>39%</td>
</tr>
<tr>
<td>Configuration</td>
<td>25% (avoided), 54% (less severe)</td>
</tr>
<tr>
<td>Timing/order of connections</td>
<td>12%</td>
</tr>
</tbody>
</table>

- Selected two from each to reproduce and avoid
Achieving Diversity

- Software Diversity
  - Version: static analysis
    - Overlap decreases quickly between versions
    - Only 25% overlap in Quagga 0.99.9 and 0.99.1
    - 30% of bugs in Quagga 0.99.9 not in 0.99.1
  - Implementation: small verification
    - Picked 10 from XORP, 10 from Quagga
    - Setup test to trigger bug
    - None were present in other implementation
Feb 16, 2009: SuproNet

• Recall: 1 misconfig tickled 2 bugs

• Bug 1: MikroTik range-check bug
  – version diversity (fixed in latest version)

• Bug 2: Cisco long AS path bug
  – configuration diversity (an alternate configuration avoids bug)
Is voting really necessary?

- Voting adds code (which adds bugs)...
  - But, it’s relatively simple (functionality and lines of code)
  - Simpler, means easier to verify with static analysis

- We already have a standby...
  - Let’s see how many bugs cause crashes
# Categorizing Faults in Bugzilla DBs

<table>
<thead>
<tr>
<th>Fault (occurrence freq*)</th>
<th>Symptom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash/hang (41%)</td>
<td>Signal sent, non-participation in vote</td>
</tr>
<tr>
<td>Add incorrect Link Attr (11%)</td>
<td>Incorrect during vote</td>
</tr>
<tr>
<td>Prevent Link Startup / Delete existing link (5%)</td>
<td>Socket error, non-participation, only instance to withdraw</td>
</tr>
<tr>
<td>Use wrong link (6%)</td>
<td>Only participant, non-participation, or incorrect in a FIB update</td>
</tr>
<tr>
<td>Add non-existent route (9%)</td>
<td>Only instance to advertise</td>
</tr>
<tr>
<td>Delete existent route (10%)</td>
<td>Only instance to withdraw</td>
</tr>
<tr>
<td>Fail to advertise route (11%)</td>
<td>Non-participation in update</td>
</tr>
<tr>
<td>Incorrect policy (6%)</td>
<td>Incorrect attr, only instance to advertise or withdraw</td>
</tr>
<tr>
<td>Incorrect logging (1%)</td>
<td>Not handled</td>
</tr>
</tbody>
</table>

* XORP and Quagga bugzilla databases
Is voting really necessary?

- Voting adds code (which adds bugs)...
  - But, it’s relatively simple (functionality and lines of code)
  - Simpler, means easier to verify with static analysis

- We already have a standby...
  - Only 41% cause crash/hang
  - Rest are byzantine
Is this even possible?

- Routers already at high CPU utilization...
- Use a better processor (small part of cost)
- Ride multi-core trend
- Utilize existing physical redundancy
  - Standby route processor and routers
- Run instances in background
  - Used to check, not active in each update
Diverse Replication

• It is effective
  – Both software and data diversity are effective

• It is necessary
  – Only 41% of bugs cause a crash/hang
  – Rest cause byzantine faults

• It is possible
  – Use better (multi-core) CPUs
  – Run in background
  – Existing redundancy
Prototype

• Based on Linux with open source routing software (XORP and Quagga)
  – Details can be read about in our paper

• No router software modification

• Detect and recover from faults

• Low complexity
Other Deployment Scenarios

- Server-based read-only operation
  - Routers run on server to cross-check
  - Migrate router process upon fault

- Network-wide deployment
  - Parallel networks instead of parallel instances (enables protocol diversity)

- Process-level deployment
  - Reduce overhead by sharing RIB
Conclusions

• Our design has several benefits
  – First step in building bug-tolerant networks
  – Diverse replication both viable and effective
  – Prototype shows improved robustness to bugs with tolerable additional delay

• Next step?
  – Looking for a place to deploy... anyone?
  – Automate diversity mechanisms
Questions

• Read more at: 
  http://verb.cs.princeton.edu