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

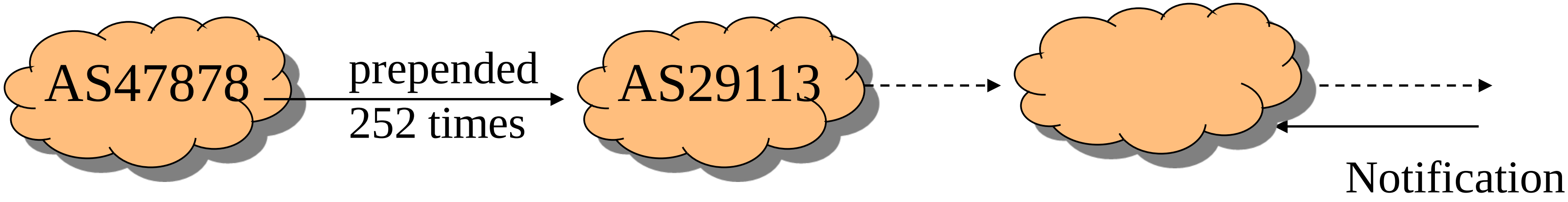
- Announced a single prefix to a single provider
  - huge increase in the global rate of updates
  - 10x increase in global instability for an hour
  - 1 misconfiguration tickled 2 bugs (2 vendors)

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

Challenges of router bugs

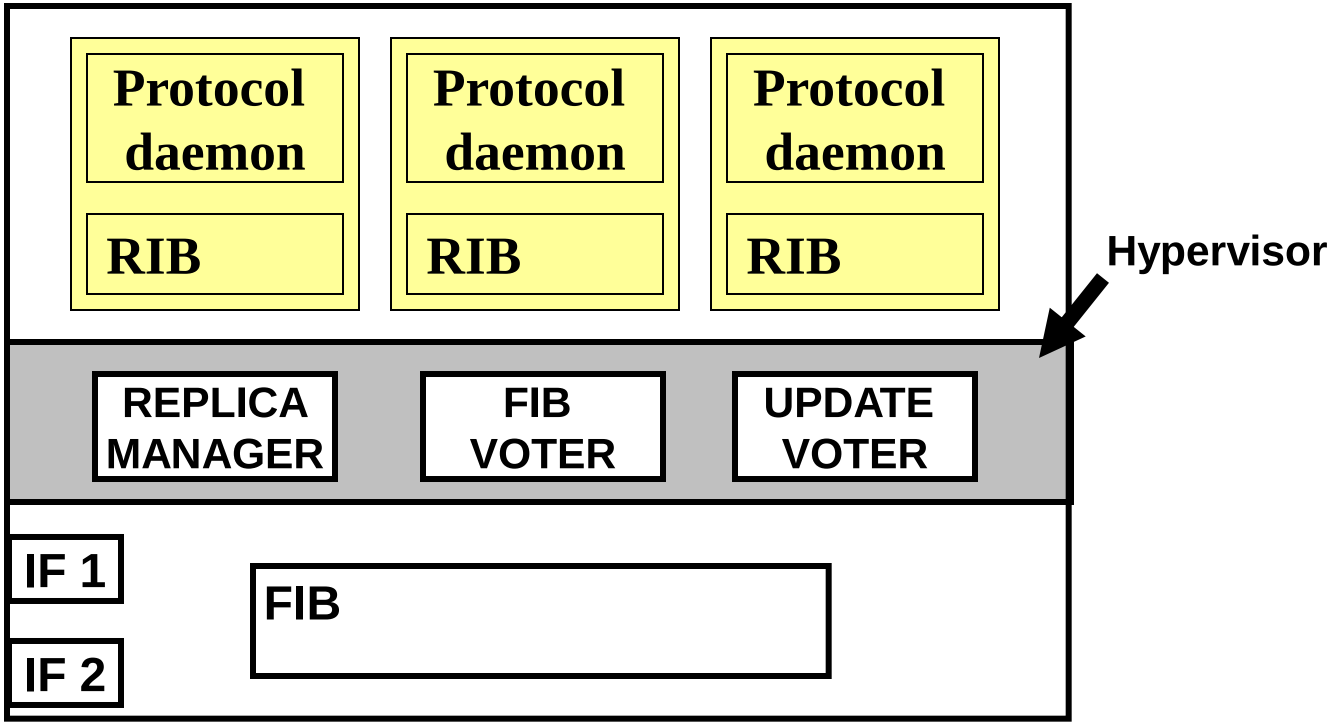
- Bugs different from traditional failures
  - Violate protocol, cause cascading outages, need to wait for vendor to repair, some exploitable by attackers
- Problem is getting worse
  - Increasing demands on routing, vendors allowing 3rd party development, other sources of outage becoming less common

Building bug-tolerant routers

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

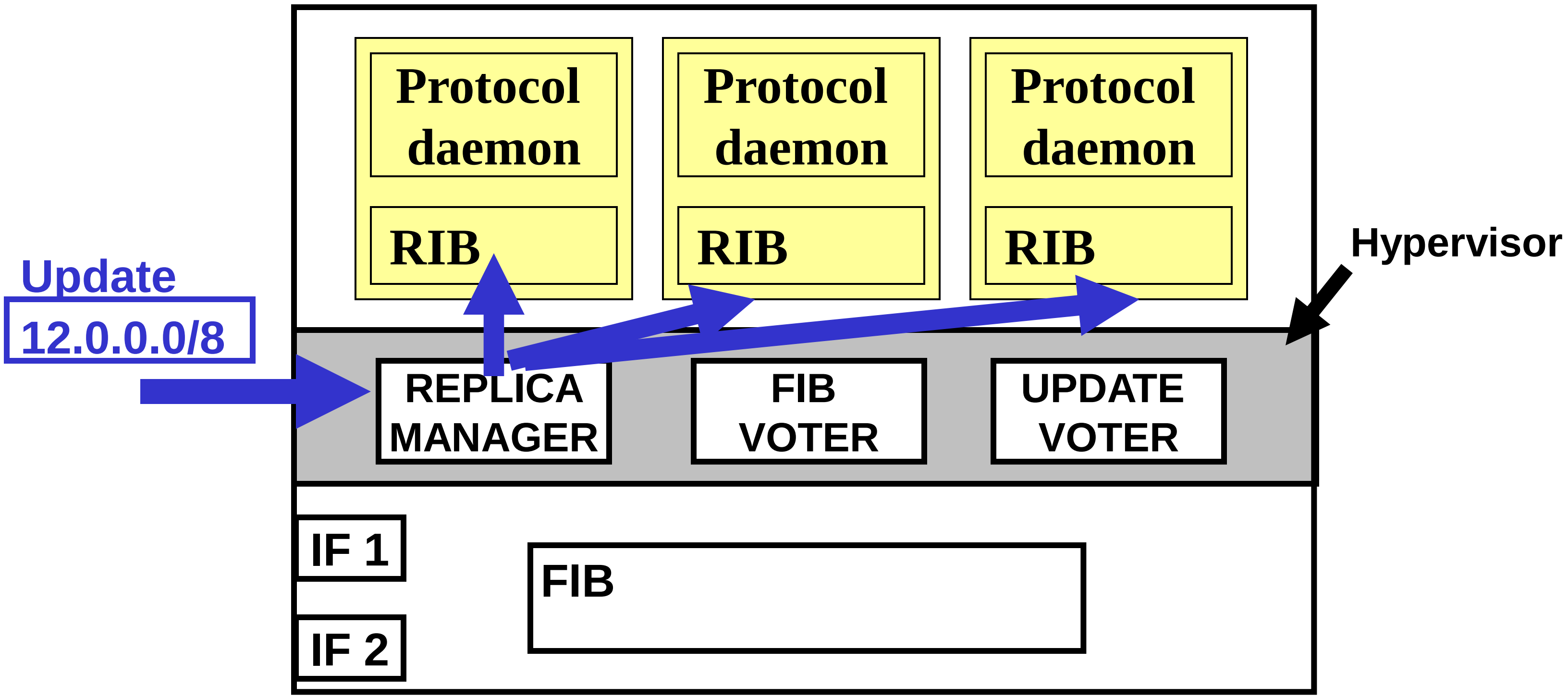
- Instances “vote” on FIB contents, update messages sent to peers

Bug-tolerant Router Architecture



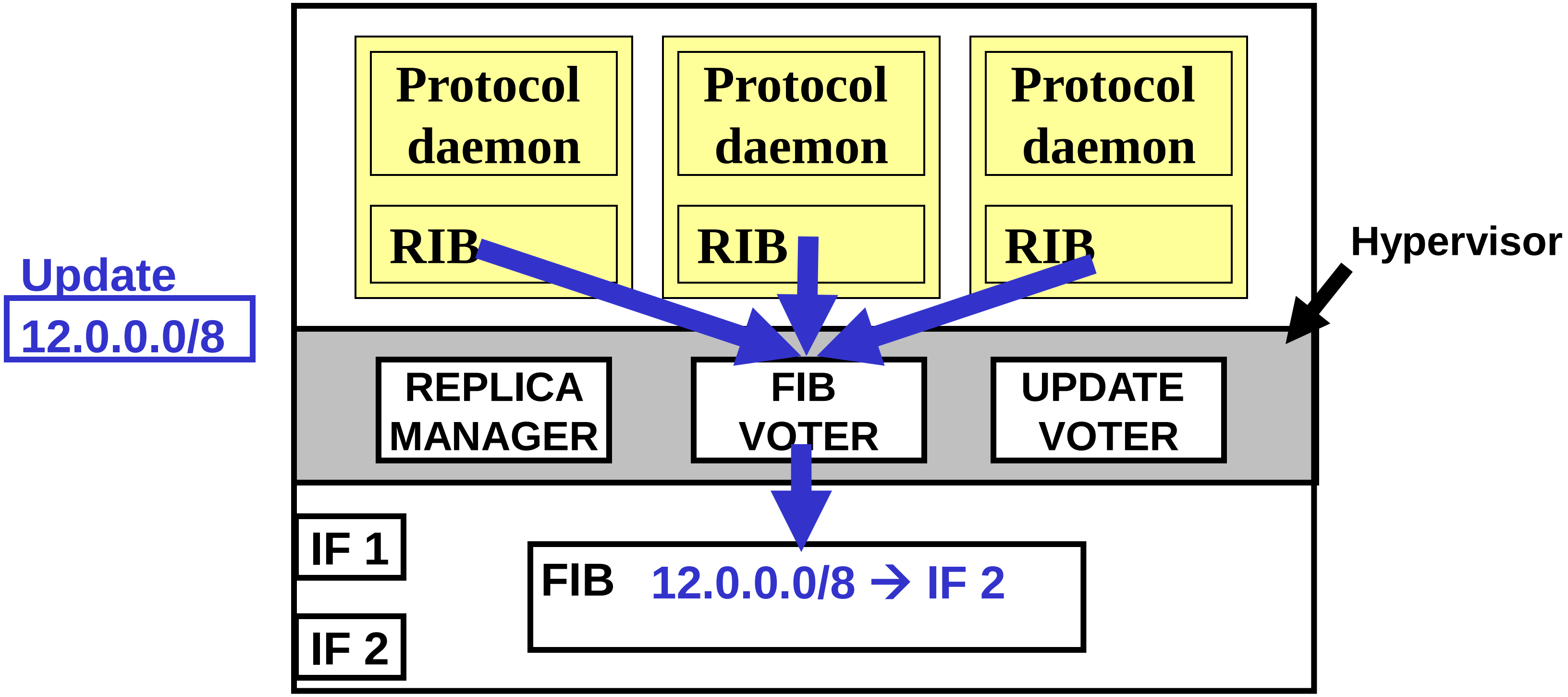
- Hypervisor:
  - Distributes received messages
  - Votes on updates (to FIB or to peer)
  - Maintains replicas (hiding churn)

Bug-tolerant Router Architecture



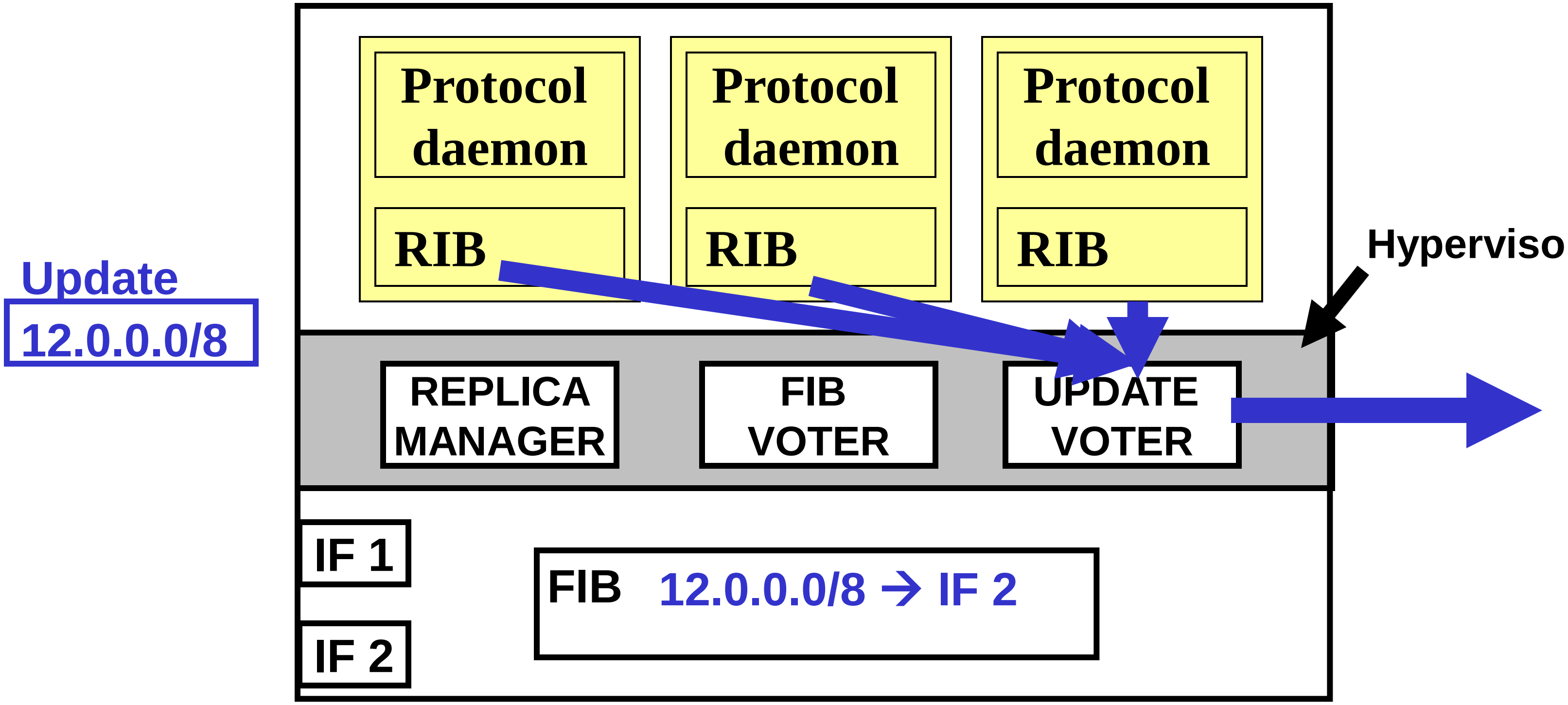
- Hypervisor:
  - Distributes received messages
  - Votes on updates (to FIB or to peer)
  - Maintains replicas (hiding churn)

Bug-tolerant Router Architecture



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

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?

Type of diversity	Examples
Execution Environment	Operating system, memory layout
Software Diversity	Version (0.98 vs 0.99), implementation (Quagga vs XORP)
Data Diversity	Configuration, timing of updates/connections

- 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

Diversity Mechanism	Bugs avoided (est.)
Timing/order of messages	39%
Configuration	25% (avoided), 54% (less severe)
Timing/order of connections	12%

  - 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

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

* 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