Refactoring Router Software to Minimize Disruption

Eric Keller

Advisor: Jennifer Rexford

Princeton University

Final Public Oral - 8/26/2011
User’s Perspective of The Internet
The Actual Internet

- Real infrastructure with real problems
Change Happens

- Network operators need to make changes
  - Install, maintain, upgrade equipment
  - Manage resource (e.g., bandwidth)
Change is Painful

- Network operators need to deal with change
  - Install, maintain, upgrade equipment
  - Manage resource (e.g., bandwidth)
Change is Painful -- Loops

- Network operators need to deal with change
  - Install, maintain, upgrade equipment
  - Manage resource (e.g., bandwidth)
Change is Painful -- Blackholes

- Network operators need to deal with change
  - Install, maintain, upgrade equipment
  - Manage resource (e.g., bandwidth)
Change is Painful -- Bugs

- Single update partially brought down Internet
  - 8/27/10: House of Cards
  - 5/3/09: AfNOG Takes Byte Out of Internet
  - 2/16/09: Reckless Driving on the Internet

[Renesys]
Degree of the Problem Today

• Tireless effort of network operators
  – Majority of the cost of a network is management [yankee02]
Degree of the Problem Today

• Tireless effort of network operators
  – Majority of the cost of a network is management [yankee02]

• Skype call quality is an order of magnitude worse (than public phone network) [CCR07]
  – Change as much to blame as congestion
  – 20% of unintelligible periods lasted for 4 minutes
The problem will get worse
The problem will get worse

• More devices and traffic
  – Means more equipment, more networks, more change
The problem will get worse

- More devices and traffic
- More demanding applications
  e-mail ➔ social media ➔ streaming (live) video

[Images of logos for Outlook, Facebook, Gmail, FourSquare, and Netflix]
The problem will get worse

- More devices and traffic
- More demanding applications
  e-mail → social media → streaming (live) video
- More critical applications
  business software → smart power grid → healthcare
Minimizing the Disruption

Goal: Make change pain free (minimize disruption)
Goal: Make change pain free (minimize disruption)

Approach: Refactoring router software
Change without...

Unnecessary reconvergence events
Change without...

Unnecessary reconvergence events

Triggering bugs
Change without...

Unnecessary reconvergence events

Coordination with neighbor

Triggering bugs
Change without...

- Unnecessary reconvergence events
- Coordination with neighbor
- Triggering bugs
- An Internet-wide Upgrade

Internet 2.0
Refactor Router Software?
Hiding Router Bugs

Physical network
Router software

Bug Tolerant Router

Bad message
Decouple Physical and Logical

IP network

Physical network

VROOM
Unbind Links from Routers

Internet (network of networks)  Router Grafting
IP network
## Outline of Contributions

<table>
<thead>
<tr>
<th>Bug Tolerant Router</th>
<th>VROOM</th>
<th>Router Grafting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide (router bugs)</td>
<td>Decouple (logical and physical)</td>
<td>Break binding (link to router)</td>
</tr>
<tr>
<td>Software and Data Diversity</td>
<td>Virtual Router Migration</td>
<td>Seamless Edge Link Migration</td>
</tr>
<tr>
<td>[CoNext09]</td>
<td>[SIGCOMM08]</td>
<td>[NSDI10]</td>
</tr>
</tbody>
</table>
Part I:
Hiding Router Software Bugs with the Bug Tolerant Router

With Minlan Yu, Matthew Caesar, Jennifer Rexford

[CoNext 2009]
## Outline of Contributions

<table>
<thead>
<tr>
<th>Bug Tolerant Router</th>
<th>VROOM</th>
<th>Router Grafting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide (router bugs)</td>
<td>Decouple (logical and physical)</td>
<td>Break binding (link to router)</td>
</tr>
<tr>
<td>Software and Data Diversity</td>
<td>Virtual Router Migration</td>
<td>Seamless Edge Link Migration</td>
</tr>
<tr>
<td>[CoNext09]</td>
<td>[SIGCOMM08]</td>
<td>[NSDI10]</td>
</tr>
</tbody>
</table>
Dealing with Router Bugs

- Cascading failures
- Effects only seen after serious outage
Dealing with Router Bugs

• Cascading failures
• Effects only seen after serious outage

Stop their effects *before* they spread
Avoiding Bugs via Diversity

• Run *multiple, diverse* instances of router software

• Instances “vote” on routing updates

• Software and Data Diversity used in other fields
Approach is a Good Fit for Routing

- Easy to vote on standardized output
  - Control plane: IETF-standardized routing protocols
  - Data plane: forwarding-table entries

- Easy to recover from errors via bootstrap
  - Routing has limited dependency on history
  - Don’t need much information to bootstrap instance

- Diversity is effective in avoiding router bugs
  - Based on our studies on router bugs and code
Approach is **Necessary** for Routing

- Easy to vote on standardized output
  - Control plane: IETF-standardized routing protocols
  - Data plane: forwarding-table entries
- Easy to recover from errors via bootstrap
  - Routing has limited dependency on history
  - Don’t need much information to bootstrap instance
- Diversity is effective in avoiding router bugs
  - Based on our studies on router bugs and code

60% of bugs do not crash the router
## Achieving Diversity

<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>Data Diversity</td>
<td>Configuration, timing of updates/connections</td>
</tr>
<tr>
<td>Software Diversity</td>
<td>Version (0.98 vs 0.99), implementation (Quagga vs XORP)</td>
</tr>
</tbody>
</table>
# Achieving Diversity

<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>Data Diversity</td>
<td>Configuration, timing of updates/connections</td>
</tr>
<tr>
<td>Software Diversity</td>
<td>Version (0.98 vs 0.99), implementation (Quagga vs XORP)</td>
</tr>
</tbody>
</table>
## Effectiveness of 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%</td>
</tr>
<tr>
<td>Timing/order of connections</td>
<td>12%</td>
</tr>
</tbody>
</table>

Selected two from each to reproduce and avoid
Effectiveness of Software Diversity

• Sanity check on implementation diversity
  – Picked 10 bugs from XORP, 10 bugs from Quagga
  – None were present in the other implementation

• Static code analysis on version diversity
  – Overlap decreases quickly between versions
  – 75% of bugs in Quagga 0.99.1 are fixed in 0.99.9
  – 30% of bugs in Quagga 0.99.9 are newly introduced

• Vendors can also achieve software diversity
  – Different code versions
  – Code from acquired companies, open-source
BTR Architecture Challenge #1

• Making replication transparent
  – Interoperate with existing routers
  – Duplicate network state to routing instances
  – Present a common configuration interface
Architecture

- Protocol daemon
- Routing table
- Protocol daemon
- Routing table
- Protocol daemon
- Routing table

“Hypervisor”

- REPLICAMANAGER
- FIBVOTER
- UPDATEVOTER

- Interface 1
- Forwarding table (FIB)
- Interface 2
Replicate Incoming Routing Messages

No protocol parsing – operates at socket level
Vote on Forwarding Table Updates

Transparent by intercepting calls to “Netlink”
Vote on Control-Plane Messages

Transparent by intercepting socket system calls
Hide Replica Failure

Kill/Bootstrap instances
BTR Architecture Challenge #2

• Making replication transparent
  – Interoperate with existing routers
  – Duplicate network state to routing instances
  – Present a common configuration interface

• Handling transient, real-time nature of routers
  – React quickly to network events
  – But not over-react to transient inconsistency
Simple Voting Mechanisms

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

• **Continuous Majority**: handling transience
  – Voting rerun when any instance sends an update
  – Output when majority agree (among responders)
Evaluation with Prototype

• Prototype
  – Built on Linux with XORP, Quagga, and BIRD
  – No modification of routing software
  – Simple hypervisor (hundreds of lines of code)

• Evaluation
  – Which algorithm is best?
  – What is the processing overhead?

• Setup
  – Inject bugs with some frequency and duration
  – Evaluated in 3GHz Intel Xeon
  – BGP trace from Route Views on March, 2007
Which algorithm is best?

Depends… there are tradeoffs

<table>
<thead>
<tr>
<th>Voting algorithm</th>
<th>Avg wait time (sec)</th>
<th>Fault rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single router</td>
<td>-</td>
<td>0.06600%</td>
</tr>
<tr>
<td>Master-slave</td>
<td>0.020</td>
<td>0.00060%</td>
</tr>
<tr>
<td>Continuous-majority</td>
<td>0.035</td>
<td>0.00001%</td>
</tr>
</tbody>
</table>
What is the processing overhead?

• Overhead of hypervisor
  – 1 instance ==> 0.1%
  – 5 routing instances ==> 4.6%
  – 5 instances in heavy load ==> 23%
  – Always, less than 1 second

• Little effect on network-wide convergence
  – ISP networks from Rocketfuel, and cliques
  – Found no significant change in convergence
    (beyond the pass through time)
Bug tolerant router Summary

- Router bugs are serious
  - Cause outages, misbehaviors, vulnerabilities
- Software and data diversity (SDD) is effective
- Design and prototype of bug-tolerant router
  - Works with Quagga, XORP, and BIRD software
  - Low overhead, and small code base
Part II:
Decoupling the Logical from Physical with VROOM

With Yi Wang, Brian Biskeborn, Kobus van der Merwe, Jennifer Rexford

[SIGCOMM 08]
## Outline of Contributions

<table>
<thead>
<tr>
<th>Bug Tolerant Router</th>
<th>VROOM</th>
<th>Router Grafting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide (router bugs)</td>
<td>Decouple (logical and physical)</td>
<td>Break binding (link to router)</td>
</tr>
<tr>
<td>Software and Data Diversity</td>
<td>Virtual Router Migration</td>
<td>Seamless Edge Link Migration</td>
</tr>
<tr>
<td>[CoNext09]</td>
<td>[SIGCOMM08]</td>
<td>[NSDI10]</td>
</tr>
</tbody>
</table>
(Recall) Change Happens

- Network operators need to make changes
  - Install, maintain, upgrade equipment
  - Manage resource (e.g., bandwidth)
(Recall) Change is Painful

- Network operators need to deal with change
  - Install, maintain, upgrade equipment
  - Manage resource (e.g., bandwidth)
The Two Notions of “Router”

IP-layer logical functionality & the physical equipment

Logical (IP layer)

Physical
Root of many network adaptation challenges

The Tight Coupling of Physical & Logical

Logical (IP layer)

Physical
Decouple the “Physical” and “Logical”

• Whenever physical changes are the goal, e.g.,
  – Replace a hardware component
  – Change the physical location of a router

• A router’s logical configuration stays intact
  – Routing protocol configuration
  – Protocol adjacencies (sessions)
VROOM: Breaking the Coupling

Re-map the logical node to another physical node

Logical (IP layer)

Physical
VROOM: Breaking the Coupling

Re-map the logical node to another physical node

VROOM enables this re-mapping of logical to physical through **virtual router migration**
Example: Planned Maintenance

- NO reconfiguration of VRs, NO disruption
Example: Planned Maintenance

- NO reconfiguration of VRs, NO disruption
Example: Planned Maintenance

• NO reconfiguration of VRs, NO disruption
Virtual Router Migration: the Challenges

1. Migrate an entire virtual router instance
   • All control plane & data plane processes / states
Virtual Router Migration: the Challenges

1. Migrate an entire virtual router instance
2. Minimize disruption
   - Data plane: millions of packets/second on a 10Gbps link
   - Control plane: less strict (with routing message retransmission)
Virtual Router Migration: the Challenges

1. Migrate an entire virtual router instance
2. Minimize disruption
3. Link migration
Virtual Router Migration: the Challenges

1. Migrate an entire virtual router instance
2. Minimize disruption
3. Link migration
VROOM’s Migration Process

- Key idea: separate the migration of control and data planes
  
  1. Migrate the control plane
  2. Clone the data plane
  3. Migrate the links
Control-Plane Migration

Leverage virtual server migration techniques

• Router image
  – Binaries, configuration files, etc.
Control-Plane Migration

Leverage virtual server migration techniques

- **Router image**
- **Memory**
  - 1\(^{st}\) stage: iterative pre-copy
  - 2\(^{nd}\) stage: stall-and-copy (when the control plane is “frozen”)
Leverage virtual server migration techniques

• Router image

• Memory
Data-Plane Cloning

- Clone the data plane by repopulation
  - Enable migration across different data planes
Remote Control Plane

- **Data-plane cloning takes time**
  - Installing 250k routes takes over 20 seconds*

- **The control & old data planes need to be kept “online”**

- **Solution: redirect routing messages through tunnels**

Remote Control Plane

• Data-plane cloning takes time
  – Installing 250k routes takes over 20 seconds*

• The control & old data planes need to be kept “online”

• Solution: redirect routing messages through tunnels

Double Data Planes

- At the end of data-plane cloning, both data planes are ready to forward traffic
Asynchronous Link Migration

- With the double data planes, links can be migrated independently
Prototype Implementation

• Control plane: OpenVZ + Quagga

• Data plane: two prototypes
  – Software-based data plane (SD): Linux kernel
  – Hardware-based data plane (HD): NetFPGA

• Why two prototypes?
  – To validate the data-plane hypervisor design (e.g., migration between SD and HD)
Evaluation

• Impact on data traffic
  – SD: Slight delay increase due to CPU contention
  – HD: no delay increase or packet loss

• Impact on routing protocols
  – Average control-plane downtime: 3.56 seconds
    (performance lower bound)
  – OSPF and BGP adjacencies stay up
VROOM Summary

• Simple abstraction
• No modifications to router software (other than virtualization)
• No impact on data traffic
• No visible impact on routing protocols
Part III: Seamless Edge Link Migration with Router Grafting

With Jennifer Rexford, Kobus van der Merwe

[NSDI 2010]
## Outline of Contributions

<table>
<thead>
<tr>
<th>Bug Tolerant Router</th>
<th>VROOM</th>
<th>Router Grafting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide (router bugs)</td>
<td>Decouple (logical and physical)</td>
<td>Break binding (link to router)</td>
</tr>
<tr>
<td>Software and Data Diversity</td>
<td>Virtual Router Migration</td>
<td>Seamless Edge Link Migration</td>
</tr>
<tr>
<td>[CoNext09]</td>
<td>[SIGCOMM08]</td>
<td>[NSDI10]</td>
</tr>
</tbody>
</table>
(Recall) Change Happens

- Network operators need to make changes
  - Install, maintain, upgrade equipment
  - Manage resource (e.g., bandwidth)
(Recall) Change is Painful

• Network operators need to deal with change
  – Install, maintain, upgrade equipment
  – Manage resource (e.g., bandwidth)
Understanding the Disruption (today)

1) Reconfigure old router, remove old link
2) Add new link, configure new router
3) Establish new BGP session (exchange routes)
<table>
<thead>
<tr>
<th>Understanding the Disruption (today)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Reconfigure old router, remove old link</td>
</tr>
<tr>
<td>2) Add new link, configure new router</td>
</tr>
<tr>
<td>3) Establish new BGP session (exchange routes)</td>
</tr>
</tbody>
</table>

| Downtime (Minutes) | 83 |
Breaking the Link to Router Binding
Router Grafting enables this breaking apart a router (splitting/merging).
Not Just State Transfer

Migrate session

AS100

AS200

AS300

AS400
Not Just State Transfer

Migrate session

The topology changes
(Need to re-run decision processes)
Challenge: Protocol Layers

A

B

C

Physical Link

Migrate Link

Migrate State

BGP
TCP
IP

BGP
TCP
IP

Exchange routes
Deliver reliable stream
Send packets
Physical Link

A

B

Exchange routes
Deliver reliable stream
Send packets
Physical Link
Migrate Link

BGP
TCP
IP

BGP
TCP
IP

Migrate State

C
Physical Link

• Unplugging cable would be disruptive
Physical Link

- Unplugging cable would be disruptive
- Links are not physical wires
  - Switchover in nanoseconds
IP

- Exchange routes
- Deliver reliable stream
- Send packets

Physical Link

Migrate Link

Migrate State
Changing IP Address

- IP address is an identifier in BGP
- Changing it would require neighbor to reconfigure
  - Not transparent
  - Also has impact on TCP (later)
Re-assign IP Address

- IP address not used for global reachability
  - Can move with BGP session
  - Neighbor doesn’t have to reconfigure

neighboring network

network making change
TCP

A

B

C

Physical Link

Migrate State

Migrate Link

Exchange routes

Deliver reliable stream

Send packets

TCP

BGP

IP
Dealing with TCP

• TCP sessions are long running in BGP
  – Killing it implicitly signals the router is down

• BGP graceful restart and TCP migrate are possible workarounds
  – Requires the neighbor router to support it
  – Not available on all routers
Migrating TCP Transparently

- Capitalize on IP address not changing
  - To keep it completely transparent

- Transfer the TCP session state
  - Sequence numbers
  - Packet input/output queue (packets not read/ack’d)
BGP

Exchange routes
Deliver reliable stream
Send packets
Physical Link
Migrate Link
Migrate State
BGP: What (not) to Migrate

• Requirements
  – Want data packets to be delivered
  – Want routing adjacencies to remain up

• Need
  – Configuration
  – Routing information

• Do not need (but can have)
  – State machine
  – Statistics
  – Timers

• Keeps code modifications to a minimum
Routing Information

Migrate-from router send the migrate-to router:

- The routes it learned
  - Instead of making remote end-point re-announce

- The routes it advertised
  - So able to send just an incremental update
Migration in The Background

- Migration takes a while
  - A lot of routing state to transfer
  - A lot of processing is needed
- Routing changes can happen at any time
- Migrate in the background
Prototype

• Added grafting into Quagga
  – Import/export routes, new ‘inactive’ state
  – Routing data and decision process well separated

• Graft daemon to control process

• SockMi for TCP migration

Graftable Router

Modified Quagga

graft daemon

SockMi.ko

Emulated link migration

Handler Comm

click.ko

Unmod. Router

Quagga

Linux kernel 2.6.19.7

Linux kernel 2.6.19.7-click

Linux kernel 2.6.19.7
# Evaluation

**Mechanism:**

- Impact on migrating routers
- Disruption to network operation

**Application:**

- Traffic engineering
Impact on Migrating Routers

- How long migration takes
  - Includes export, transmit, import, lookup, decision
  - CPU Utilization roughly 25%

Migration Time (seconds) vs. Routing Table Size (# prefixes)

- Between Routers
  - 0.9s (20k)
  - 6.9s (200k)
Disruption to Network Operation

- **Data traffic affected by not having a link**
  - nanoseconds

- **Routing protocols affected by unresponsiveness**
  - Set old router to “inactive”, migrate link, migrate TCP, set new router to “active”
  - milliseconds
Traffic Engineering (traditional)

* adjust routing protocol parameters based on traffic
Traffic Engineering w/ Grafting

* Rehome customer to change where traffic enters/exits
Traffic Engineering Evaluation

• Setup
  – Internet2 topology and 1 week of traffic data
  – Simple greedy heuristic

• Findings
  – Network can handle more traffic (18.8%)
  – Don’t need frequent re-optimization (2-4/day)
  – Don’t need to move many links (<10%)
Router Grafting Summary

• Enables moving a single link with…
  – Minimal code change
  – No impact on data traffic
  – No visible impact on routing protocol adjacencies
  – Minimal overhead on rest of network

• Applying to traffic engineering…
  – Enables changing ingress/egress points
  – Networks can handle more traffic
Part IV: A Unified Architecture
Grafting

BGP1 (Router Grafting)

Seamless Link Migration
Grafting + BTR

- BGP1 (Router Grafting)
- BGP2 (Router Grafting)
- Control-plane hypervisor (BTR)

Multiple Diverse Instances
Grafting + BTR + VROOM

- Virtual Router
  - BGP1 (Router Grafting)
  - BGP2 (Router Grafting)
- Control-plane hypervisor (BTR)
- Data-plane Hypervisor (VROOM)

Virtual Router Migration

- BGP1 (Router Grafting)
- BGP2 (Router Grafting)
- Control-plane hypervisor (BTR)
Summary of Contributions

• New refactoring concepts
  – Hide (router bugs) with *Bug Tolerant Router*
  – Decouple (logical and physical) with *VROOM*
  – Break binding (link to router) with *Router Grafting*

• Working prototype and evaluation for each

• Incrementally deployable solution
  – “Refactored router” can be drop in replacement
  – Transparent to neighboring routers
Final Thoughts
Questions