Routing in Fat Trees (RIFT) Hackathon
Chaos Monkey Testing of RIFT Implementations

IETF 104, Prague
Bruno Rijsman, brunorijsman@gmail.com, 24-Mar-2019 v3
What is RIFT?

• RIFT = Routing In Fat Trees
• A new link-state routing protocol optimized for “fat tree” topologies
• Main use case is large data center networks
RIFT Hackathon Participants

<table>
<thead>
<tr>
<th>Name</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artur Makutunowicz</td>
<td>LinkedIn</td>
</tr>
<tr>
<td>Bruno Rijsman</td>
<td>Individual</td>
</tr>
<tr>
<td>Pascal Thubert</td>
<td>Cisco</td>
</tr>
<tr>
<td>Tony Przygienda</td>
<td>Juniper Networks</td>
</tr>
</tbody>
</table>
RIFT Hackathon Activities

• Use open source RIFT-Python implementation.
• Generate configuration for large data center topology.
• Run topology in virtual machine (AWS instance).
• Automatically test correct initial convergence.
• Introduce a sequence of random “perturbations” in the topology.
• Automatically test correct initial re-convergence.
• For more details see:
  • https://youtu.be/GqebgPmA4Xc: hackathon instructions video
Run fat tree topology in virtual machine

• Each RIFT-Python router runs in separate network namespace
• Use virtual Ethernet (veth) pairs to connect routers.
Generate topology and scripts

Meta-configuration

nr-pods: 3
nr-leaf-nodes-per-pod: 3
nr-spine-nodes-per-pod: 3
nr-superspine-nodes: 4

Configuration for each RIFT router

leaf-1

spine-1

super-1

...

Scripts to start and stop topology

start.sh

stop.sh

Scripts for “chaos monkey” perturbations

chaos.sh

Scripts to test correct convergence

check.sh
“Chaos monkey” perturbation testing

• Tool generates “chaos script” to randomly break and repair things
• “Chaos script” is topology aware
• Things that are being broken and repaired:
  • Full bi-directional link failures
  • Node failures
  • More things in future: uni-directional link failures, packets drops, packet re-ordering, packet delay, slow CPU, …
• All breakages are repaired at the end of the script
• After chaos script is finished, run check script to check convergence
“Chaos monkey” perturbation testing
Example chaos run

```
root@06786deal16bc:/host# generated/chaos.sh
Break Link if-1001a-if-101a (bi-directional failure)
Break Link if-2b-if-102d (bi-directional failure)
Fix Link if-2b-if-102d
Break Node super-1
Break Link if-1c-if-103c (bi-directional failure)
Break Link if-1002a-if-101b (bi-directional failure)
Fix Link if-1001a-if-101a
Break Link if-1003a-if-103a (bi-directional failure)
Fix Link if-1c-if-103c
Break Node spine-2-2
Fix Node spine-2-2
Break Link if-1001a-if-101a (bi-directional failure)
Fix Link if-1001a-if-101a
Fix Link if-1003a-if-103a
Fix Node super-1
Fix Link if-1002a-if-101b
Break Link if-1c-if-103c (bi-directional failure)
Break Link if-2d-if-104d (bi-directional failure)
Fix Link if-1c-if-103c
Break Node spine-2-1
Break Link if-1c-if-103c (bi-directional failure)
Break Link if-1001b-if-102a (bi-directional failure)
```
Automated convergence testing

• Tool generates “check script” to check correct re-convergence
• “Check script” is topology aware
• Things that are tested:
  • Ping from every leaf to every other leaf
  • Each node is up
  • All adjacencies are up (3-way)
  • North-bound default routes are in RIB and FIB and kernel
  • South-bound specific /32 routes are in RIB and FIB and kernel
  • More things in future
Example check convergence run

```bash
root@06786dea16bc:/host# tools/config_generator.py -n -c meta_topology/clos_2pod_2leaf_2spine_2super.yml

**** Check node leaf-1-1
OK Can Telnet to RIFT process
OK RIFT engine is responsive
OK Interfaces are up
OK North-bound default routes are present
OK South-bound specific routes are present
OK RIB and FIB are consistent
OK FIB and Kernel are consistent

**** Check node leaf-1-2
OK Can Telnet to RIFT process
OK RIFT engine is responsive
OK Interfaces are up
OK North-bound default routes are present
OK South-bound specific routes are present
OK RIB and FIB are consistent
OK FIB and Kernel are consistent

**** Check node spine-1-1
OK Can Telnet to RIFT process
OK RIFT engine is responsive
OK Interfaces are up
OK North-bound default routes are present
OK South-bound specific routes are present
```
Protocol visualization tool to help debug issues

TX TIDE ProtocolPacket(header=PacketHeader(level=1, sender=1, major_version=19, minor_version=0), content=PacketContent)

Transition VALID_REFLECTION [TWO WAY] > start_flooding [THREE WAY]

Transition NEIGHBOR_OFFER [UPDATING_CLIENTS] > update_or_remove_offer [None]

Push TIMER_TICK

Transition TIMER_TICK [THREE WAY] > check_hold_time_expired,SEND_LIE [None]

TX LIE ProtocolPacket(header=PacketHeader(level=1, sender=1, major_version=19, minor_version=0), content=PacketContent)

Transition SEND_LIE [THREE WAY] > send_lie [None]

Push TIMER_TICK

RX LIE ProtocolPacket(header=PacketHeader(level=1, sender=1, major_version=19, minor_version=0), content=PacketContent)

Push LIE_RECEIVED

Transition TIMER_TICK [TWO WAY] > check_hold_time_expired,SEND_LIE [None]

TX LIE ProtocolPacket(header=PacketHeader(level=0, sender=2, major_version=19, minor_version=0), content=PacketContent)

Transition SEND_LIE [TWO WAY] > send_lie [None]

Push NEIGHBOR_OFFER

Transition LIE_RECEIVED [TWO WAY] > process_lie,VALID_REFLECTION [None]
Hackathon results and lessons learned

• Implemented framework for:
  • Generate large fat tree topology (config files, scripts, visualization, …)
  • Run topology in virtual machine (AWS instance, using network namespaces)
  • Automated “chaos monkey” perturbation testing
  • Automated testing of correct re-convergence

• Lessons learned
  • We found and fixed several implementation issues using the framework:
    • IPv6 flooding issue (IPv4 in one direction, IPv6 in the other direction)
    • Multiple scenarios where exceptions are not handled in shut-down scenarios
    • Several ideas for new show commands to help debug issues (some implemented)
    • **No new issues in protocol specification found** (only implementation issues)