RIFT open source implementation

- On GitHub: [https://github.com/brunorijsman/rift-python](https://github.com/brunorijsman/rift-python)
- Grew out of IETF 102 hackathon
  - Original modest goal was to test the LIE FSM
  - Work is continuing to become complete RIFT implementation
- Goals:
  - Help get the RIFT specification to the point that it is clear and complete
  - To be a reference RIFT implementation
- Current emphasis on debuggability, not performance
- Implemented in Python
- Extensive documentation: [README.md](https://github.com/brunorijsman/rift-python/README.md)
- Not associated with any vendor
Getting started with RIFT-Python

https://github.com/brunorijsman/rift-python/blob/master/README.md

Routing In Fat Trees (RIFT)

This repository contains a Python implementation of the Routing In Fat Trees (RIFT) protocol specified in Internet Draft (ID) draft-draft-rift-03.

The code is currently still a work in progress (see Feature List below for the status).

Documentation

- Feature List
- Installation Instructions
- Startup Instructions
- Command Line Options
- Command Line Interface (CLI)
- Logging
- Log Visualization

• Installation Instructions
• Startup Instructions

24-Oct-2018 v1  RIFT Open Source Implementation Update (Bruno Rijisman)  3
## Current status summary

<table>
<thead>
<tr>
<th>Feature group</th>
<th>Completeness estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjacencies</td>
<td>75%</td>
</tr>
<tr>
<td>Zero touch provisioning (ZTP)</td>
<td>100%</td>
</tr>
<tr>
<td>Flooding</td>
<td>50%</td>
</tr>
<tr>
<td>Route calculation</td>
<td>0%</td>
</tr>
<tr>
<td>Management interface</td>
<td>50%</td>
</tr>
<tr>
<td>Development toolchain</td>
<td>75%</td>
</tr>
</tbody>
</table>

Note: all estimates are a finger in the wind estimates
## Current status: adjacencies

<table>
<thead>
<tr>
<th>Complete</th>
<th>Not Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exchange LIE packets</td>
<td>IPv6 adjacencies</td>
</tr>
<tr>
<td>LIE finite state machine</td>
<td>New multi-neighbor state</td>
</tr>
<tr>
<td>IPv4 adjacencies</td>
<td>Interactions with BFD</td>
</tr>
<tr>
<td><strong>Interoperability with vendor RIFT</strong></td>
<td>Security procedures (nonce)</td>
</tr>
</tbody>
</table>
## Current status: Zero Touch Provisioning (ZTP)

<table>
<thead>
<tr>
<th>Complete</th>
<th>Not Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTP finite state machine</td>
<td>-</td>
</tr>
<tr>
<td>Automatic level determination</td>
<td></td>
</tr>
<tr>
<td>Interoperability with vendor RIFT</td>
<td></td>
</tr>
</tbody>
</table>
## Current status: flooding

<table>
<thead>
<tr>
<th>Complete</th>
<th>Not Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exchange TIE / TIDE / TIRE packets</td>
<td>Efficient TIE propagation (w/o decode)</td>
</tr>
<tr>
<td>Node TIEs</td>
<td>Positive disaggregation TIEs</td>
</tr>
<tr>
<td>Prefix TIEs</td>
<td>Negative disaggregation TIEs</td>
</tr>
<tr>
<td>TIE database</td>
<td>Key-value TIEs</td>
</tr>
<tr>
<td>TX / RTX / REQ / ACK queues</td>
<td>External TIEs</td>
</tr>
<tr>
<td>Flooding procedures</td>
<td>Policy-guided prefixes</td>
</tr>
<tr>
<td>Flooding scope rules (N, S, EW)</td>
<td>Setting sent overload bit</td>
</tr>
<tr>
<td>South-bound default route origination</td>
<td>Clock comparison</td>
</tr>
<tr>
<td>Honoring received overload bit</td>
<td></td>
</tr>
</tbody>
</table>
# Current status: route calculation

<table>
<thead>
<tr>
<th>Complete</th>
<th>Not Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Routing Information Base (RIB)</td>
</tr>
<tr>
<td></td>
<td>Forwarding Information Base (FIB)</td>
</tr>
<tr>
<td></td>
<td>North-bound SPF</td>
</tr>
<tr>
<td></td>
<td>South-bound SPF</td>
</tr>
<tr>
<td></td>
<td>East-west forwarding</td>
</tr>
<tr>
<td></td>
<td>Positive disaggregation procedures</td>
</tr>
<tr>
<td></td>
<td>Negative disaggregation procedures</td>
</tr>
<tr>
<td></td>
<td>Optimized route calculation on leafs</td>
</tr>
<tr>
<td></td>
<td>Fabric bandwidth balancing</td>
</tr>
<tr>
<td></td>
<td>Label binding / segment routing</td>
</tr>
</tbody>
</table>
**Current status: management**

<table>
<thead>
<tr>
<th>Complete</th>
<th>Partial</th>
<th>Not Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration file</td>
<td>Configuration commands</td>
<td>SSH CLI client</td>
</tr>
<tr>
<td>Telnet CLI client</td>
<td>Command history</td>
<td>Command completion</td>
</tr>
<tr>
<td>Operational commands</td>
<td>Command help</td>
<td>YANG data models</td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-node topologies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logging</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Current status: development toolchain

<table>
<thead>
<tr>
<th>Complete</th>
<th>Not Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated unit tests</td>
<td>100% code coverage</td>
</tr>
<tr>
<td>Automated system tests</td>
<td>Wireshark dissector</td>
</tr>
<tr>
<td>Automated interop tests</td>
<td></td>
</tr>
<tr>
<td>Travis continuous integration (CI)</td>
<td></td>
</tr>
<tr>
<td>Codecov code coverage (~ 80%)</td>
<td></td>
</tr>
<tr>
<td>Strict pylint</td>
<td></td>
</tr>
<tr>
<td>Finite state machine (FSM) framework</td>
<td></td>
</tr>
<tr>
<td>Visualization tool</td>
<td></td>
</tr>
</tbody>
</table>
Protocol issues discovered (and fixed)

- **Multi-neighbor oscillation**
  - Connecting 3 RIFT nodes to a LAN causes traffic spike (LIEs)
  - Two flavors: amplified and non-amplified
  - Caused by “triggered loops” in the finite state machine
  - Solution: new multi-neighbor state

- **Flooding oscillations**
  - In stable topology, you should only see TIDEs, not TIREs or TIEs
  - We observed persistent “oscillations” of TIRE and TIE messages
  - Various variations of the problem observed
  - Solution for now: tweak the flooding scope rules
  - Considered for future: explicit flooding scope in TIE header

- **Other minor issues (not discussed here)**
Multi-neighbor scenario

Node X  Node Y  Node Z

Multi-point LAN is not supported by RIFT
But could happen by accident.
How does the protocol behave?
Multi-neighbor traffic explosion

Connect 3 nodes to LAN:
Traffic spikes to line rate
All LIE messages
Multi-neighbor amplified oscillation
Cause of multi-neighbor oscillation

**X receives LIE from Y**
- Event: New Neighbor
- Action: Multicast LIE to Y and Z

**X receives LIE from Z**
- Event: Multi-Neighbor
- Action: Multicast LIE to Y and Z

**Each Cycle:**
- X receives 1 LIE from Y
- X receives 1 LIE from Z
- X multicasts 2 LIEs
- Each is received by both Y and Z
- Y sends 1 LIE, receives 2 LIEs from X (and also 2 LIEs from Y)
- Z sends 1 LIE, receives 2 LIEs from X (and also 2 LIEs from Y)
- All actions triggers by packets
- No timers involved
Cause of multi-neighbor oscillation

Exponential growth of number of LIE messages

Each Cycle:
- X receives 1 LIE from Y
- X receives 1 LIE from Z
- X multicasts 2 LIEs
- Each is received by both Y and Z
- Y sends 1 LIE, receives 2 LIEs from X (and also 2 LIEs from Y)
- Z sends 1 LIE, receives 2 LIEs from X (and also 2 LIEs from Y)
- All actions triggers by packets
- No timers involved

FSM oscillates as fast as it can, not constrained by timer ticks
Solution: new multi-neighbor state

State: ONE WAY
X receives LIE from Y
Event New Neighbor

State: TWO WAY
X receives LIE from Z
Event Multi-Neighbor

State: MULTI-NEIGHBOR

"Cool-down" timer expires
Flooding oscillation #1

Node 1
Level 1
North

TIDE  TIRE  TIDE  TIRE  TIDE  TIRE  TIDE  TIRE  TIDE  TIRE

TIE    TIE    TIE    TIE    TIE    TIE    TIE    TIE

Node 2
Level 0
South
Flooding oscillation #1

**Step 1: Node 2 send TIE**
- Dir = North
- Originator = 2
- Type = Node
- TIE Nr = xxx
- Seq Nr = yyy

**Step 2: Node 1 sends TIRE**
- ACKs received TIE
- North:2:Node:xxx:yyy

**Step 3: Node 1 sends TIDE**
- Is missing TIE header:
  - North:2:Node:xxx:yyy

**Step 4: Node 1 sends TIDE**
- Node 1 retransmits TIE
- Back to step 1
Flooding oscillation #2

Node 1
Level 1
North

TIDE
TIRE
TIRE
TIRE
TIRE
TIRE
TIRE
TIDE

Node 2
Level 0
South
Flooding oscillation #2

**Step 1: Node 1 sends TIDE**
Announces a TIE header:
North:1:Node:xxx:yyy

**Step 2: Node 2 sends TIRE**
Node 2 does not have TIE
Node 2 requests TIE

**Step 3: Node 1 does NOT send TIE**
The flooding scope rules don’t allow node 1 to send the requested TIE

**Step 4: Node 3 resends TIRE**
Node 1 retransmits TIRE
Back to step 2
Solution for flooding oscillations

• The flooding scope rules are “sensitive”
  • A tiny change in the rules can have unanticipated consequences (e.g. oscillations)
  • The rules for TIE flooding, TIDE contents, and TIRE contents must be consistent (which much more non-trivial than one would guess)

• Solution for now: tweak the flooding scope rules
• Considered for future: explicit flooding scope in TIE header
• For more details see http://bit.ly/rift-flooding-oscillations
Interoperability testing

- Run RIFT-Vendor in one process (publicly available)
- Run RIFT-Python in another process
- Both use common “topology file”
  - Specifies the topology of the complete “network under test”
  - Specifies which nodes are run by RIFT-Vendor and which by RIFT-Python
- Interoperability testing is fully automated
  - Run full suite of system tests
  - For each system test, try all permutations of Vendor / Python nodes
- So far, successfully completed interop testing for:
  - Adjacency establishment and automatic level determination
  - Flooding (not automated yet)
Conclusions

• Open source RIFT-Python implementation has helped the draft progress
  • Editorial improvements
  • Protocol improvements
• Interoperability testing at a very early stage has flushed out issues
• Visualization tool is essential to understand the protocol behavior
• Weekly RIFT calls are essential (the deep discussions happen here)
• Additional contributors (pull requests) for RIFT-Python are welcome