RIFT Open Source Implementation

Status Update, Lessons Learned, and Interop Testing

Bruno Rijsman, 23-Oct-2018, v1

RIFT open source implementation

- On GitHub: 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
- Not associated with any vendor

Getting started with RIFT-Python

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

build passing codecov 81%

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

Current status summary

Feature group	Completeness estimate
Adjacencies	75%
Zero touch provisioning (ZTP)	100%
Flooding	50%
Route calculation	0%
Management interface	50%
Development toolchain	75%

Note: all estimates are a finger in the wind estimates

Current status: adjacencies

Complete	Not Complete
Exchange LIE packets LIE finite state machine IPv4 adjacencies **Interoperability with vendor RIFT**	IPv6 adjacencies New multi-neighbor state Interactions with BFD Security procedures (nonce)

Current status: Zero Touch Provisioning (ZTP)

Complete	Not Complete
ZTP finite state machine Automatic level determination **Interoperability with vendor RIFT**	-

Current status: flooding

Complete	Not Complete
Exchange TIE / TIDE / TIRE packets	Efficient TIE propagation (w/o decode)
Node TIEs	Positive disaggregation TIEs
Prefix TIEs	Negative disaggregation TIEs
TIE database	Key-value TIEs
TX / RTX / REQ / ACK queues	External TIEs
Flooding procedures	Policy-guided prefixes
Flooding scope rules (N, S, EW)	Setting sent overload bit
South-bound default route origination	Clock comparison
Honoring received overload bit	
Interoperability with vendor RIFT	

Current status: route calculation

Complete	Not Complete
-	Routing Information Base (RIB) Forwarding Information Base (FIB) North-bound SPF South-bound SPF East-west forwarding Positive disaggregation procedures Negative disaggregation procedures Optimized route calculation on leafs Fabric bandwidth balancing Label binding / segment routing

Current status: management

Complete	Partial	Not Complete
Configuration file Telnet CLI client Operational commands Documentation Multi-node topologies Logging	Configuration commands Command history Command help	SSH CLI client Command completion YANG data models

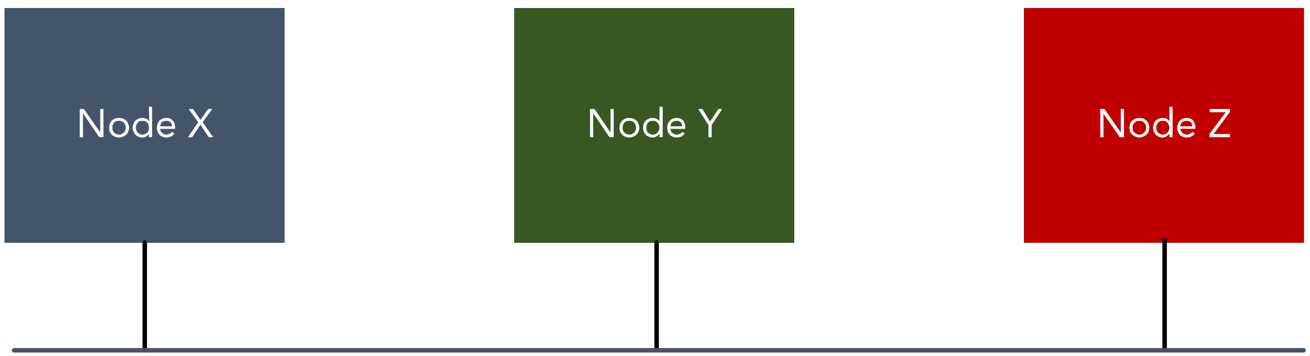
Current status: development toolchain

Complete	Not Complete
Automated unit tests Automated system tests Automated interop tests Travis continuous integration (CI) Codecov code coverage (~ 80%) Strict pylint Finite state machine (FSM) framework Visualization tool	100% code coverage Wireshark dissector

Protocol issues discovered (and fixed)

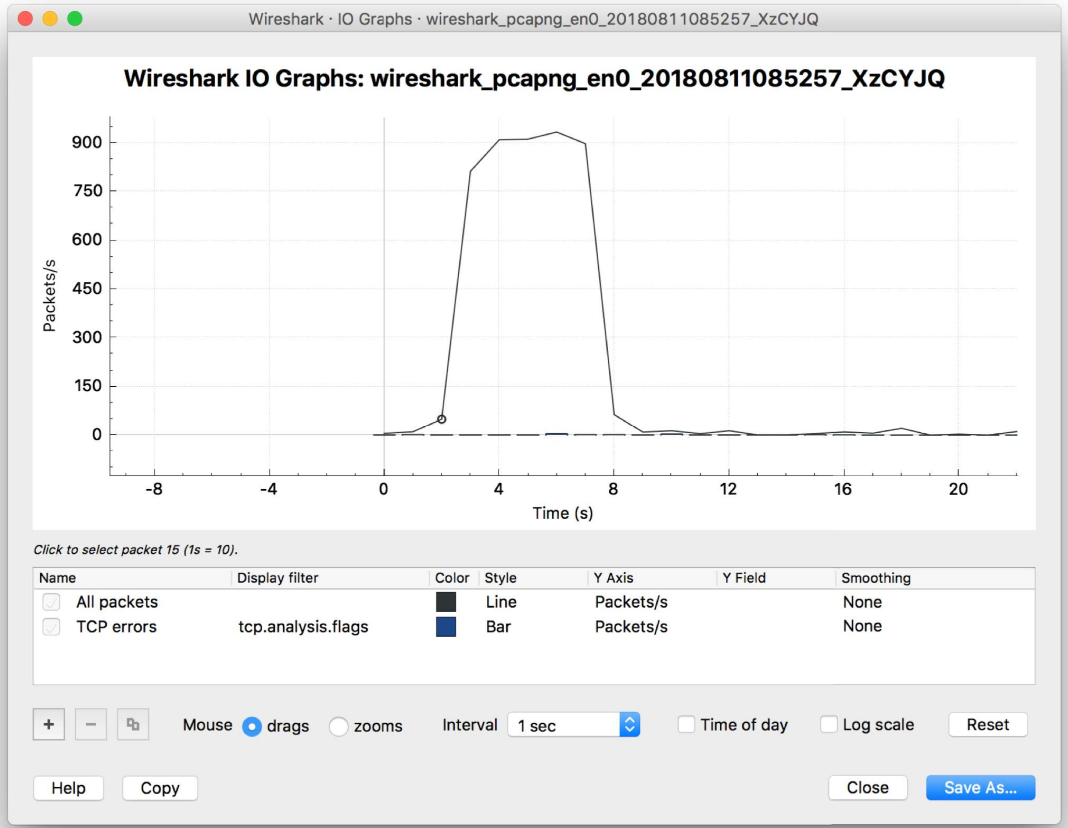
- **Multi-neighbor oscillation**
  - Connecting 3 RIFT nodes to a LAN causes traffic spike (LIEs)
  - Two flavors: amplified and non-amplified
  - Caused by "triggered loops" in the finite state machine
  - Solution: new multi-neighbor state
- **Flooding oscillations**
  - In stable topology, you should 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



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

Multi-neighbor traffic explosion

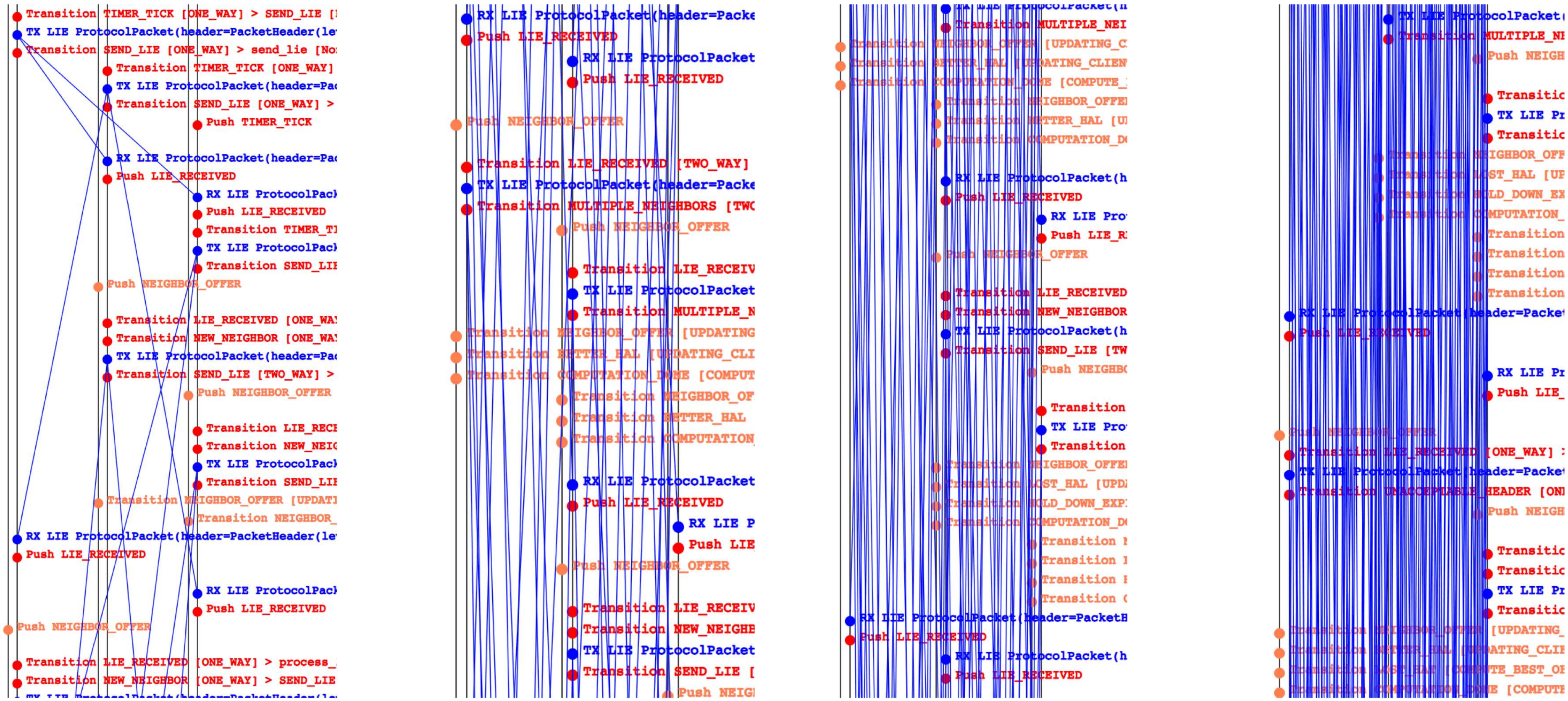


<u>Connect 3 nodes to LAN:</u>

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

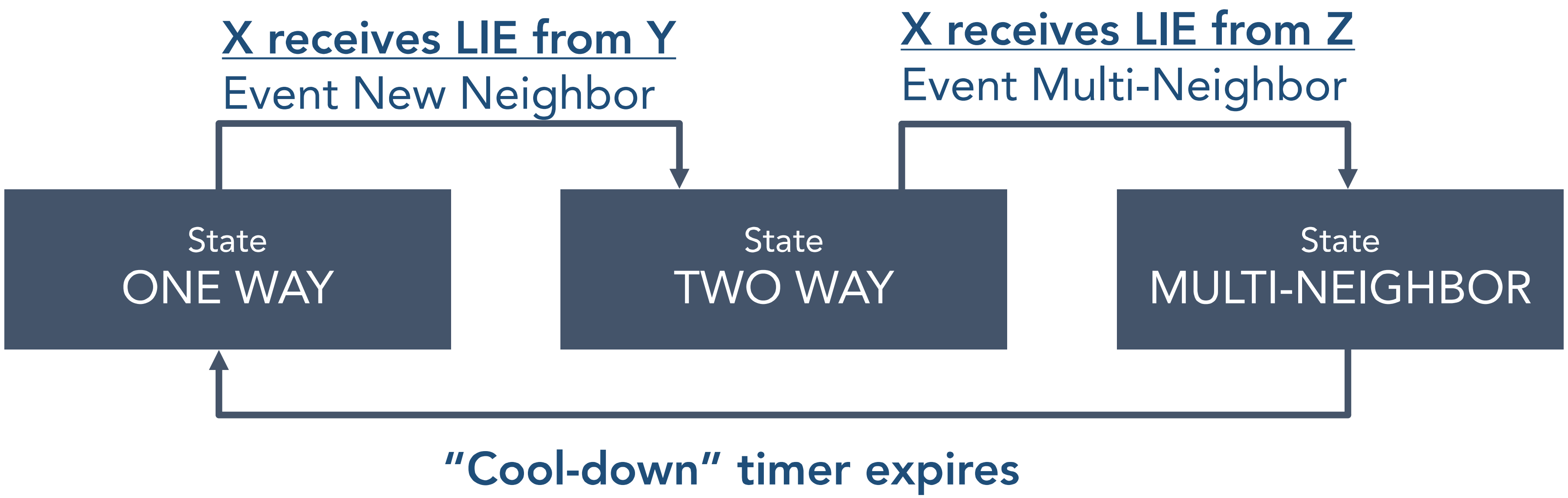
Exponential growth of number of LIE messages

FSM oscillates as fast as it can, not constrained by timer ticks

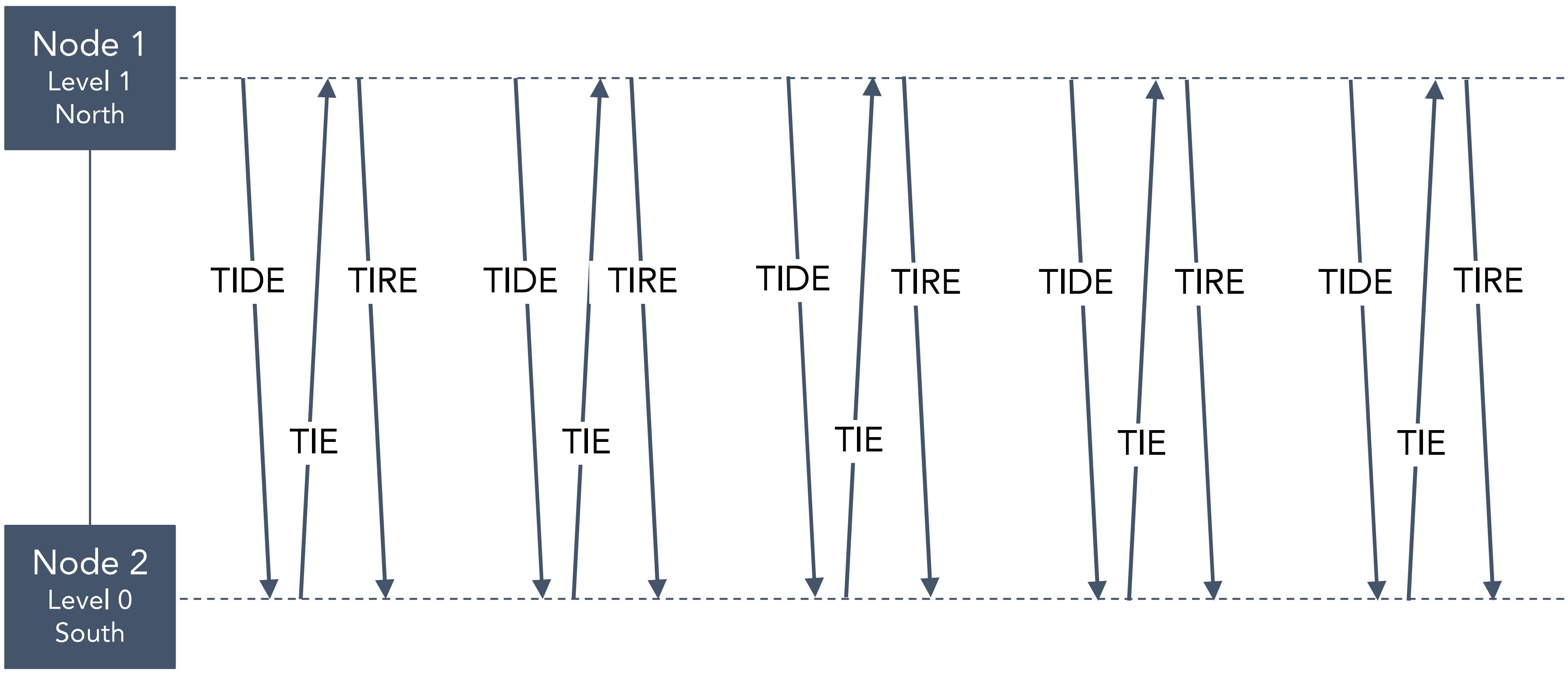
Each Cycle:

- 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

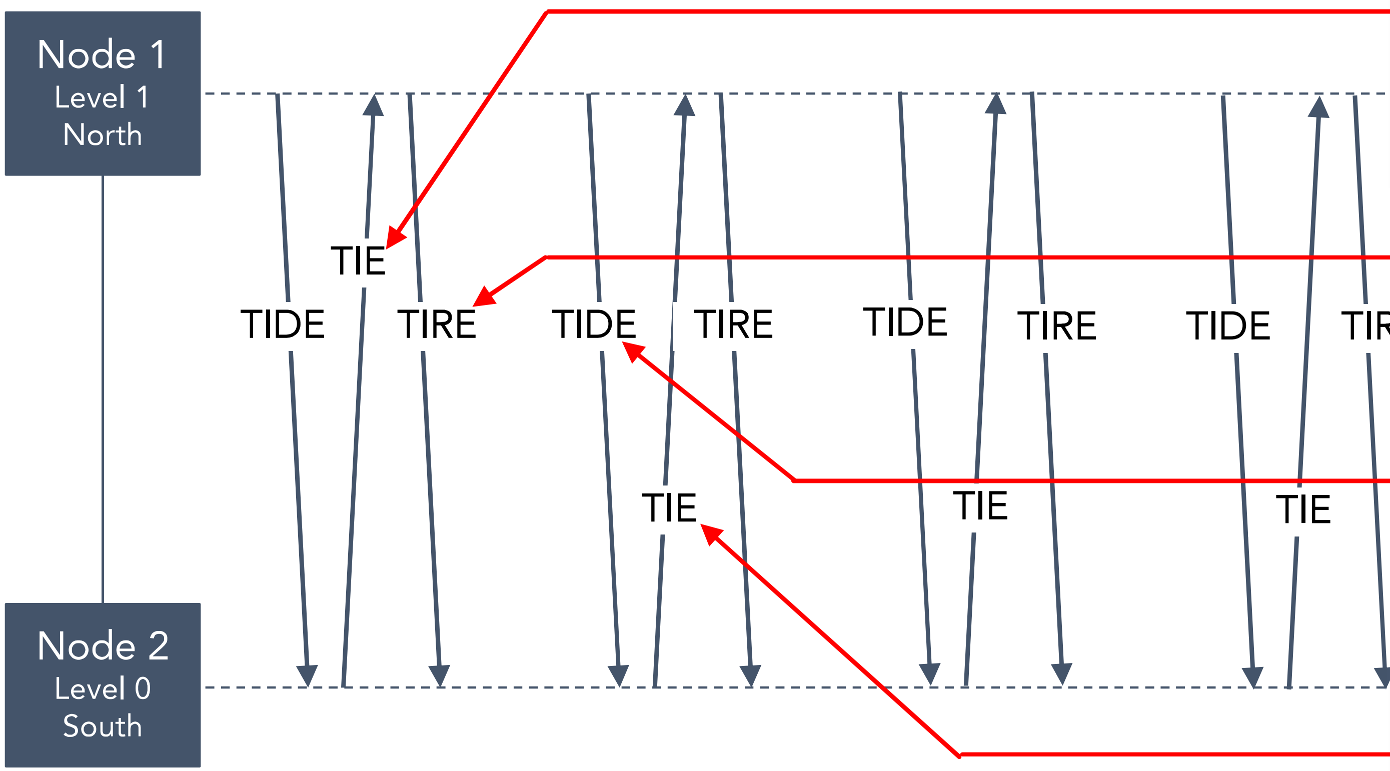
Solution: new multi-neighbor state



Flooding oscillation #1



Flooding oscillation #1



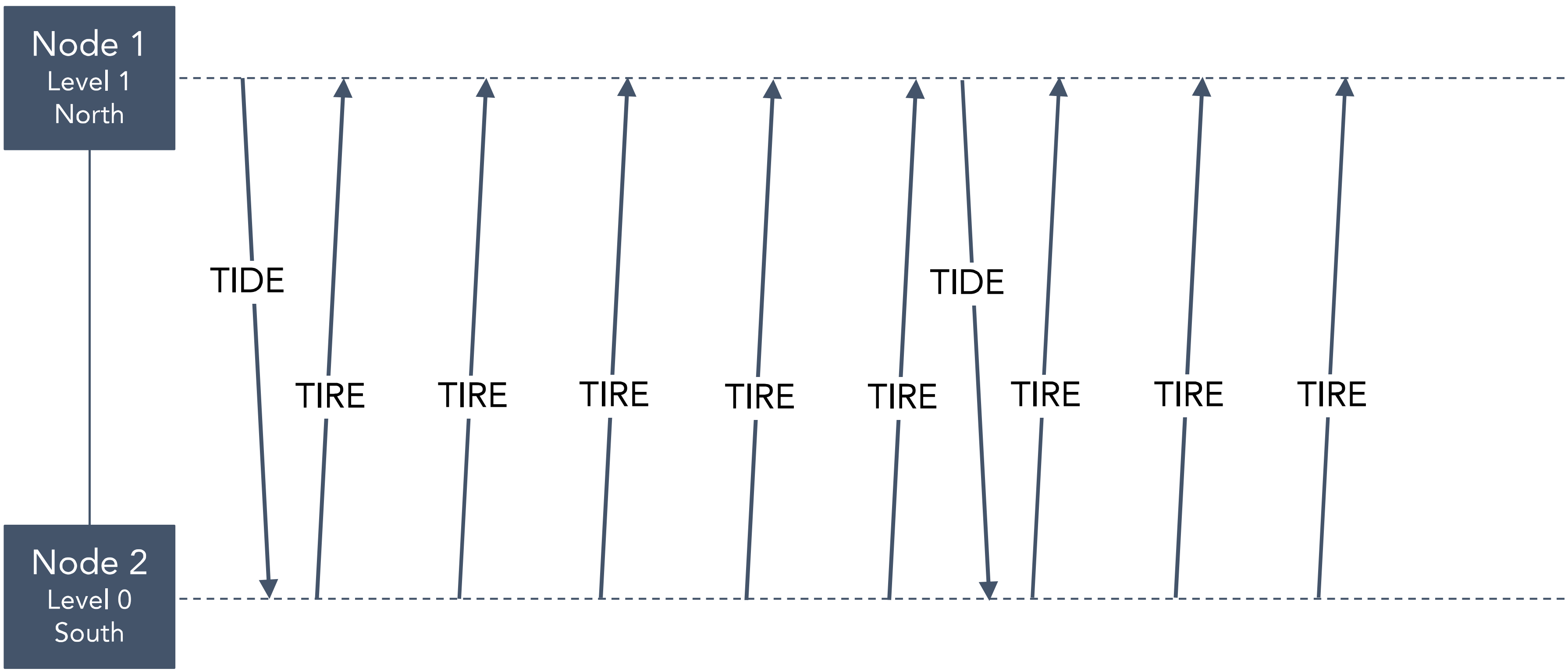
Step 1: Node 2 send TIE
Dir = North
Originator = 2
Type = Node
TIE Nr = xxx
Seq Nr = yyy

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

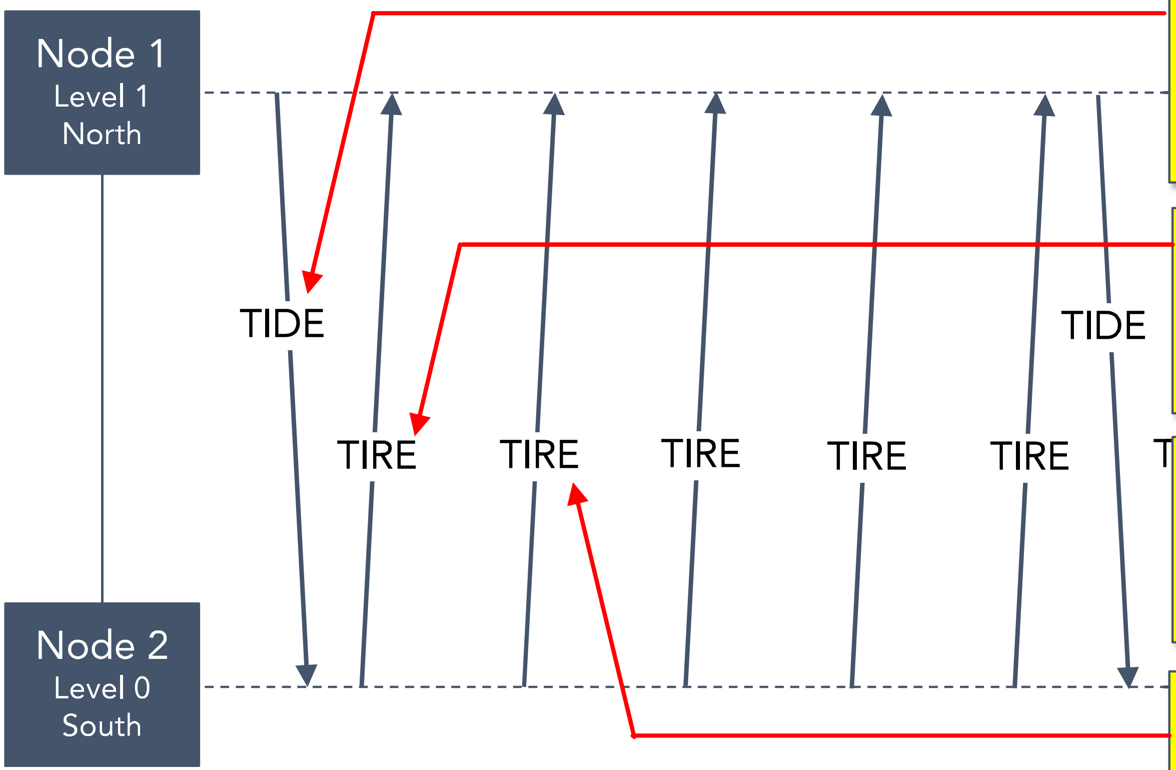
Step 3: Node 1 sends TIDE
Is missing TIE header:
~~North:2:Node:xxx:yyy~~

Step 4: Node 1 sends TIDE
Node 1 retransmits TIE
Back to step 1

Flooding oscillation #2



Flooding oscillation #2



Step 1: Node 1 send 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