Evaluating OSPF Convergence with ns-3 DCE

Nicolas Rybowski, Olivier Bonaventure
nicolas.rybowski@uclouvain.be

ICTEAM
UCLouvain

Workshop on ns-3 (WNS3) 2022
Agenda

- Background and Motivation
- BIRD Integration with DCE
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
Agenda

- Background and Motivation
- BIRD Integration with DCE
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
Classical Link-state Routing

Background

- Link-state Routing Protocols (LSR)
  - LSR PDU (LSA/LSP) describing node’s neighborhood
    - distributed by flooding
  - Link State Database (LSDB)
  - Shortest Path algorithm (local)

Equal-Cost MultiPath (ECMP)
Load-balancing packets on paths with the same cost

Bidirectional Forwarding Detection (BFD)
Quick link failure detection
Classical Link-state Routing

Background

- Link-state Routing Protocols (LSR)
  - LSR PDU (LSA/LSP) describing node’s neighborhood
    - distributed by flooding
  - Link State Database (LSDB)
  - Shortest Path algorithm (local)

- Equal-Cost MultiPath (ECMP)
  - Load-balancing packets on paths with the same cost
Classical Link-state Routing

Background

- **Link-state Routing Protocols (LSR)**
  - LSR PDU (LSA/LSP) describing node’s neighborhood
    - distributed by flooding
  - Link State Database (LSDB)
  - Shortest Path algorithm (local)
- **Equal-Cost MultiPath (ECMP)**
  - Load-balancing packets on paths with the same cost
- **Bidirectional Forwarding Detection (BFD)**
  - Quick link failure detection
How to measure IGP convergence duration after a failure?

Control plane is known to have slow reactions to failures
Routing Protocols Evaluation
Motivation

- Physical testbeds
  - ✔ Real world measurements
  - ✗ Timing measurement not reproducible
  - ✗ Costly for large topologies

Emulation
- Cheap to setup on commodity hardware
- Each node competes for resources
- Timing measurement not reproducible

Simulation
- Cheap to set up on commodity hardware
- No competition for resources since no real-time
- Reproducible timing measurement
Routing Protocols Evaluation

Motivation

- **Physical testbeds**
  - ✔ Real world measurements
  - ✗ Timing measurement not reproducible
  - ✗ Costly for large topologies

- **Emulation**
  - ✔ Cheap to setup on commodity hardware
  - ✗ Each node competes for resources
  - ✗ Timing measurement not reproducible
Routing Protocols Evaluation

Motivation

- Physical testbeds
  - ✓ Real world measurements
  - ✗ Timing measurement not reproducible
  - ✗ Costly for large topologies

- Emulation
  - ✓ Cheap to setup on commodity hardware
  - ✗ Each node competes for resources
  - ✗ Timing measurement not reproducible

- Simulation
  - ✓ Cheap to set up on commodity hardware
  - ✓ No competition for resources since no real-time
  - ✓ Reproducible timing measurement
Agenda

- Background and Motivation
- **BIRD Integration with DCE**
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
Syscalls Interception in DCE
BIRD Integration with DCE

Native
- `ffs`
- `localtime_r`
- `longjmp`

Custom Handling in DCE
- `dce_mmap64`
  - Support for private anonymous memory mapping
Toolchain I
BIRD Integration with DCE

Topology Specification: Network Topology Format (NTF)
(head, end, metric, delay)

um ul 1 5  br bl 1 5
um ur 10 5  br ur 1 5
ul um 1 5  ur br 1 5
ul ur 1 5  ur ul 1 5
ul bl 1 5  ur um 10 5
ul bm 10 5  ur bm 1 5
bl ul 1 5  bm ul 10 5
bl br 1 5  bm ur 1 5
Toolchain I
BIRD Integration with DCE

Topology Specification: Network Topology Format (NTF)
(head, end, metric, delay)

```
um ul 1 5
um ur 10 5
ul um 1 5
ul ur 1 5
ul bl 1 5
ul bm 10 5
bl ul 1 5
bl br 1 5
```

BIRD Module in ns-3: dce-bird

```
NTF Parser → DCE setup → Topology Generation → BIRD Configuration → ns-3 Simulation Start
```
Toolchain II
BIRD Integration with DCE

Full Toolchain

Wrapper
Toolchain II
BIRD Integration with DCE

Full Toolchain

Wrapper → Container

./setup.sh → ./dce-bird
Toolchain II
BIRD Integration with DCE

Full Toolchain

NPF → Wrapper → Container

./setup.sh → ./dce-bird → Logs Parsing

Workload parallelization: each container simulates a specific scenario
Toolchain II
BIRD Integration with DCE

Full Toolchain

Workload parallelization: each container simulates a specific scenario
Agenda

- Background and Motivation
- BIRD Integration with DCE
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
**Convergence components**

D + O + F + SPT + FIB + DD

- **D**: Failure Detection
- **O**: LSP Origination
- **F**: LSP Flooding
- **SPT**: SPT Computation
- **FIB**: FIB Update
- **DD**: Linecard Update

Internal Node's Delays Parameters

<table>
<thead>
<tr>
<th>Delay source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSP processing</td>
<td>[2, 4] ms</td>
</tr>
<tr>
<td>Maintenance timer</td>
<td>{10, 25, 50, 100} ms</td>
</tr>
<tr>
<td>SPT computation</td>
<td>[2, 4] ms</td>
</tr>
<tr>
<td>FIB prefix update</td>
<td>[100, 110] µs/prefix</td>
</tr>
</tbody>
</table>
Router Model
Network Model

**Convergence components**

D + O + F + SPT + FIB + DD
- **D**: Failure Detection
- **O**: LSP Origination
- **F**: LSP Flooding
- **SPT**: SPT Computation
- **FIB**: FIB Update
- **DD**: Linecard Update

**Internal Node’s Delays Parameters**

<table>
<thead>
<tr>
<th>Delay source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSP processing</td>
<td>[2,4]ms</td>
</tr>
<tr>
<td>Maintenance timer</td>
<td>{10,25,50,100}ms</td>
</tr>
<tr>
<td>SPT computation</td>
<td>[2,4]ms</td>
</tr>
<tr>
<td>FIB prefix update</td>
<td>[100, 110]µs/prefix</td>
</tr>
</tbody>
</table>
Router Model

Network Model

Convergence components

D + O + F + SPT + FIB + DD

D  Failure Detection
O  LSP Origination
F  LSP Flooding
SPT SPT Computation
FIB FIB Update
DD Linecard Update

Internal Node’s Delays Parameters

<table>
<thead>
<tr>
<th>Delay source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSP processing</td>
<td>[2,4] ms</td>
</tr>
<tr>
<td>Maintenance timer</td>
<td>{10,25,50,100} ms</td>
</tr>
<tr>
<td>SPT computation</td>
<td>[2,4] ms</td>
</tr>
<tr>
<td>FIB prefix update</td>
<td>[100, 110] µs/prefix</td>
</tr>
</tbody>
</table>
Network Failures Model

Network Model

Single Link Failure

\[ D \in \{15, 18\} \, ms \]

- Extended NTF: \((\text{head}, \text{end}, \text{metric}, \text{delay}, T_F)\)
  - \(T_F\): the delay since the start of the simulation \((T_0)\) in seconds
Network Failures Model

Network Model

Single Link Failure

\[ D \in \{15, 18\} \text{ ms} \]

Extended NTF: (head, end, metric, delay, \( T_F \))
- \( T_F \): the delay since the start of the simulation (\( T_0 \)) in seconds

Parallel links not supported yet

Cannot differentiate links with the same characteristics
Network Failures Model

Network Model

**Single Link Failure**

\[ D \in \{15, 18\} \text{ ms} \]

- Extended NTF: (head, end, metric, delay, \(T_F\))
  - \(T_F\): the delay since the start of the simulation (\(T_0\)) in seconds

**Parallel links not supported yet**

Cannot differentiate links with the same characteristics

**Node Failure**

The Single Link Failure model is applied on each node's link
Agenda

- Background and Motivation
- BIRD Integration with DCE
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
UDP Flows
Micro-loops Detection Methods

Constant bit-rate UDP flows

- UDP payload = packet generation timestamp
- Full mesh of UDP flows
- $T_{xi} = T_{xi-1} + P$

$P = 5\text{ms}$
UDP Flows
Micro-loops Detection Methods

Constant bit-rate UDP flows

- UDP payload = packet generation timestamp
- Full mesh of UDP flows
- $T_{xi} = T_{xi-1} + P$

Reordering

$\{T_{x1}; T_{x2}; T_{x5}; T_{x3}; T_{x4}; T_{x6}; T_{x7}; T_{x8}; T_{x9}; \ldots\}$

Estimated loop duration: 15ms
UDP Flows
Micro-loops Detection Methods

**Constant bit-rate UDP flows**

- $P = 5ms$
- $T_{x_1} \rightarrow T_{x_2} \rightarrow T_{x_3} \rightarrow T_{x_4} \rightarrow T_{x_5} \rightarrow T_{x_6} \rightarrow T_{x_7} \rightarrow T_{x_8} \rightarrow T_{x_9}$
- UDP payload = packet generation timestamp
- Full mesh of UDP flows
- $T_{x_i} = T_{x_{i-1}} + P$

**Reordering**

- $\{ T_{x_1}; T_{x_2}; T_{x_5}; T_{x_3}; T_{x_4}; T_{x_6}; T_{x_7}; T_{x_8}; T_{x_9}; \ldots \}$
- Estimated loop duration: 15ms

**Black-hole**

- $\{ T_{x_1}; T_{x_2}; T_{x_6}; T_{x_7}; T_{x_8}; T_{x_9}; \ldots \}$
- Estimated loop duration: 15ms
- No route to host
- TTL reached 0
UDP Flows Limitations I
Micro-loops Detection Methods

**Timing Overestimation**

- Node converged at $T_C$
- $T_{x1}$ is lost
- Next timestamp sent at $T_{x2}$

-Convergence overestimated by $\Delta t$
**Timing Overestimation**

- Node converged at $T_C$
- $T_{x1}$ is lost
- Next timestamp sent at $T_{x2}$

Convergence overestimated by $\Delta t$

**Solution?**

Increasing the granularity (flow rate) by lowering $P$?

- No: highly increases simulation duration
- Trade-off: Timing accuracy vs simulation overhead
ECMP Reordering

- IGP metrics are dimensionless
  - They may reflect the link latency
  - If not, ECMP load-balancing could provoke packet reordering
- False positive for micro-loop detection
FIB Traversal
Micro-loops Detection Methods

FIB Snapshots
- Capture FIB of each node upon FIB update of a single node
- For each snapshot, rebuild routes for each \((source, destination)\)
FIB Traversal
Micro-loops Detection Methods

FIB Snapshots
- Capture FIB of each node upon FIB update of a single node
- For each snapshot, rebuild routes for each \((source, destination)\)

Micro-loop

\[
egin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
0 \text{ via 1} \\
1 \text{ via 1} \\
3 \text{ via 1,5} \\
4 \text{ via 4} \\
5 \text{ via 5} \\
0 \text{ via 1} \\
1 \text{ via 2,3} \\
2 \text{ via 2} \\
3 \text{ via 3} \\
4 \text{ via 2}
\end{array}
\]
FIB Traversal
Micro-loops Detection Methods

FIB Snapshots
- Capture FIB of each node upon FIB update of a single node
- For each snapshot, rebuild routes for each (source, destination)

Micro-loop

```
0 via 1
1 via 1
3 via 1,5
4 via 4
5 via 5

0 via 5
1 via 5
3 via 1
4 via 4
5 via 5
```

```
0 via 1
1 via 2,3
2 via 2
3 via 3
4 via 2

0 via 1
1 via 2,3
2 via 2
3 via 3
4 via 2
```

$T_x$ $T_{x1}$
FIB Traversal
Micro-loops Detection Methods

FIB Snapshots
- Capture FIB of each node upon FIB update of a single node
- For each snapshot, rebuild routes for each \((\text{source}, \text{destination})\)

Micro-loop
FIB Traversal
Micro-loops Detection Methods

FIB Snapshots
- Capture FIB of each node upon FIB update of a single node
- For each snapshot, rebuild routes for each \((\text{source}, \text{destination})\)

Micro-loop
FIB Traversal
Micro-loops Detection Methods

FIB Snapshots
- Capture FIB of each node upon FIB update of a single node
- For each snapshot, rebuild routes for each \((\text{source}, \text{destination})\)

Micro-loop

Black-hole
FIB Traversal
Micro-loops Detection Methods

FIB Snapshots
- Capture FIB of each node upon FIB update of a single node
- For each snapshot, rebuild routes for each (source, destination)

Micro-loop

Black-hole
Agenda

- Background and Motivation
- BIRD Integration with DCE
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
House Topology: Toy Example I
Sample Simulations

**Figure:** Link failures - ECMP off

**Figure:** Link failures - ECMP on
House Topology: Toy Example II

Sample Simulations

Figure: Node failures - ECMP on
GEANT Topology: Real World Network

Sample Simulations

**Figure:** Link failures - ECMP on

**Figure:** Node failures - ECMP on
Agenda

- Background and Motivation
- BIRD Integration with DCE
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
Scalability I
Framework Evaluation

Does BIRD over DCE scale?
- Tested on small topologies
- Search lower and upper bounds for
  - Memory consumption
  - CPU time consumption

Which topology?
- Simplest topology for LSR: Ring
- Worst topology for LSR: Full Mesh

Which size?
Number of nodes (n) linearly increased by 10 up to 100
Scalability I
Framework Evaluation

Does BIRD over DCE scale?
- Tested on small topologies
- Search lower and upper bounds for
  - Memory consumption
  - CPU time consumption

How?
- What?
  - Initial convergence (5 simulated minutes)
- Which topology?
  - Simplest topology for LSR: Ring
  - Worst topology for LSR: Full Mesh
- Which size?
  - Number of nodes \((n)\) linearly increased by 10 up to 100
Scalability II
Framework Evaluation

Figure: Peak memory usage

Figure: Total runtime
Code is available

https://github.com/nrybowski/ns3-sim/tree/wns3-22

- C++ ns-3 modules
  - NTF Toplogy generator
  - BIRD daemon configurator
- Dockerfiles with patched ns-3
- RUST wrapper to launch the simulations
- NPF scripts to reproduce the figures
Questions?

- Background and Motivation
- BIRD Integration with DCE
- Network Model
- Micro-loops Detection Methods
- Sample Simulations
- Framework Evaluation
Why BIRD?

Backup Slide

Requirements

- Implement ISIS and/or OSPF
- Open-Source
  - Must recompile with DCE flags
  - Must be able to modify the code
    - Internal delays model
    - Implement new LSR extensions (Future Work)
- C/C++ code to run in DCE
- Actively maintained
  - Implement recent extensions
  - Community Support
  - Used in real-world deployments
Why BIRD?

Backup Slide
Why BIRD?
Backup Slide

FRRouting

- Implement ISIS and OSPF
- Open-Source
- C code
- Actively maintained: last update from June 2022
- Many DCE modifications
## Why BIRD?

Backup Slide

### FRRouting

- ✓ Implement ISIS and OSPF
- ✓ Open-Source
- ✓ C code
- ✓ Actively maintained: last update from June 2022
- ✗ Many DCE modifications

### Quagga

- ✓ Implement ISIS and OSPF
- ✓ Open-Source
- ✓ C code
- ✓ Actively maintained: last update from 2018
Why BIRD?
Backup Slide

FRRouting
✓ Implement ISIS **and** OSPF
✓ Open-Source
✓ C code
✓ Actively maintained: last update from June 2022
✗ Many DCE modifications

XORP
✓ Implement ISIS or OSPF
✓ Open-Source
✓ C++ code
✗ Actively maintained: last update from 2012

Quagga
✓ Implement ISIS **and** OSPF
✓ Open-Source
✓ C code
✗ Actively maintained: last update from 2018
Why BIRD?
Backup Slide

FRRouting
- Implement ISIS and OSPF
- Open-Source
- C code
- Actively maintained: last update from June 2022
- Many DCE modifications

XORP
- Implement ISIS or OSPF
- Open-Source
- C++ code
- Actively maintained: last update from 2012

Quagga
- Implement ISIS and OSPF
- Open-Source
- C code
- Actively maintained: last update from 2018

FreeRtr
- Implement ISIS and OSPF
- Open-Source
- Java code
- Actively maintained: last update from June 2022