ROBUSTNESS CHALLENGES IN SOFTWARE-DEFINED NETWORKS

Dr. ir. Niels van Adrichem
niels.vanadrichem@tno.nl
*Work performed while at Delft University of Technology
SOFTWARE-DEFINED NETWORKING

- Decoupling of data and control plane
  - Forwarding of data
  - Computing the Forwarding Rules
- Decreased management complexity
- Traffic Engineering
- Network optimization
- Implement QoS
- OpenFlow protocol
PROBLEM: TOPOLOGY ROBUSTNESS

- Software testbed: Open vSwitch 4759 ms
- Hardware testbed: Pica8 P3290 1697 ms

RELATED WORK

- “Carrier-grade” sub 50 ms (MPLS)

- Controller intervention takes at least 100 ms

<table>
<thead>
<tr>
<th>Authors</th>
<th>Recovery time</th>
<th>Recovery Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharma et al.</td>
<td>42 – 48 ms</td>
<td>Required BFDd + config on OS</td>
</tr>
<tr>
<td>Kempf et al.</td>
<td>28.2 ± 0.5 ms</td>
<td>Custom extension of OpenFlow</td>
</tr>
<tr>
<td>Sgambelluri et al.</td>
<td>32.74 ± 4.17 ms</td>
<td>Custom auto-reject mechanism</td>
</tr>
</tbody>
</table>

- Path computation?

- Much (and must) do better can!
  - -> Decrease failure recovery time AFAP
PROPOSAL

1. Fast detection: Couple per-link (L=1) Forwarding Detection to Fast Failover buckets

2. Fast recovery: Preprogram protection paths

3. Slow Optimality: Rely on controller to reconfigure

   » Minimal
     » detection window
     » #sessions in network $O(N \times N) \rightarrow O(L)$
     » #sessions traversing each link $O(N \times N) \rightarrow O(1)$
     » #sessions per end-point $O(N) \rightarrow O(D)$

• No false positives at an overhead of 0.067% @ 1 Gbps
Hello-protocol for path-based liveliness-monitoring
TCP-like 3-way handshake
Transmit-interval (thus detection window) relies on RTT

\[ T_{i,min} = 1.25 \cdot 2 \cdot (T_{Trans} + L \cdot T_{Prop} + T_{Proc}) \]

[Van Asten, B. J. "Increasing robustness of Software-Defined Networks." (2014).]
OPEN VSWITCH IMPLEMENTATION

› 50.9 ms delay in interface status change
› Add BFD Support to Group Table Fast Failover buckets:

Code-clean up, commited and accepted to Open vSwitch

OPTIMIZE TOPOLOGY FAILURE RECOVERY

› Feasible for real-world networks
› Independent of path length and network size
› Additionally: No packet-loss at repair-after-failure

COMPUTING FAILURE-OMITTING PATHS

✓ Failure detection: Couple per-link (L=1) BFD to Fast Failover buckets
   › Fast recovery: 13.6 ms at 5 ms interval

☒ Recovery: Preprogram protection paths

☒ Reconfiguration: Regain optimality of network usage
TRIVIAL SOLUTION

- Disjoint-path computation
- All-to-all disjoint path computation is expensive ($N^2$ repetition)
# DISJOINT PATH ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Finds</th>
<th>Optimizes</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative Dijkstra</td>
<td>1-to-1</td>
<td>Min-min</td>
<td>$O(k \cdot N \cdot \log N + k \cdot L)$</td>
</tr>
<tr>
<td>Suurballe</td>
<td>1-to-1</td>
<td>Min-sum</td>
<td>$O(k \cdot N \cdot \log N + k \cdot L)$</td>
</tr>
<tr>
<td>Suurballe – Tarjan</td>
<td>1-to-all</td>
<td>Min-sum</td>
<td>$O(N \cdot \log N + L)$</td>
</tr>
<tr>
<td>Roskind – Tarjan</td>
<td>All-to-all</td>
<td>Min-sum</td>
<td>$O(L \cdot \log L + k^2 \cdot N^2)$</td>
</tr>
<tr>
<td>Bhandari (modified Dijkstra)</td>
<td>1-to-1</td>
<td>Min-sum</td>
<td>$O(k \cdot N \cdot 2^N)$</td>
</tr>
<tr>
<td>Bhandari (Bellman-Ford)</td>
<td>1-to-1</td>
<td>Min-sum</td>
<td>$O(k \cdot N \cdot L)$</td>
</tr>
<tr>
<td>Medard-Finn-Barry-Gallager</td>
<td>1-to-all</td>
<td>Min-sum</td>
<td>NA</td>
</tr>
</tbody>
</table>
# DISJOINT PATH ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>All-to-all</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative Dijkstra</td>
<td>(O(k \cdot N^3 \cdot \log N + k \cdot N^2 \cdot L))</td>
<td>(N^2) repetition, Trap-topology unsafe</td>
</tr>
<tr>
<td>Suurballe</td>
<td>(O(k \cdot N^3 \cdot \log N + k \cdot N^2 \cdot L))</td>
<td>(N^2) repetition</td>
</tr>
<tr>
<td>Suurballe – Tarjan</td>
<td>(O(N^2 \cdot \log N + N \cdot L))</td>
<td>(k=2)</td>
</tr>
<tr>
<td>Roskind – Tarjan</td>
<td>(O(L \cdot \log L + k^2 \cdot N^2))</td>
<td>*Tree</td>
</tr>
<tr>
<td>Bhandari (modified Dijkstra)</td>
<td>(O(k \cdot N^3 \cdot 2^N))</td>
<td>(N^2) repetition</td>
</tr>
<tr>
<td>Bhandari (Bellman-Ford)</td>
<td>(O(k \cdot N^3 \cdot L))</td>
<td>(N^2) repetition</td>
</tr>
<tr>
<td>Medard-Finn-Barry-Gallager</td>
<td>(NA)</td>
<td>*Tree</td>
</tr>
</tbody>
</table>
GAP

- We are NOT looking for path-disjointness
GAP

- Although it works using crankback routing

We are actually looking for a failure-disjoint path.
EXISTING SOLUTIONS

- Routing protocols:
  - Slow detection and convergence

- IPv4 (Not-So) Loop Free Alternates (LFAs)
  - Protects 75% of traffic

- IPFRR Not-via
  - Backup path suboptimal, requires encapsulation, problems with multihoming

- RSVP- / MPLS-TE-FRR
  - Next-, or next-next-hop omission only
PER-LINK APPROACH

- Add and remove labels accordingly
  - VLAN tags
  - MPLS labels
  - MAC rewriting

[van Adrichem, Iqbal and Kuipers, Backup rules in Software-Defined Networks, IEEE SDN/NFV 2016]
PER-LINK APPROACH

**Input**: Adjacency matrix \( adj = G(N, L) \)

**Output**: Forwarding matrix \( fw \) containing primary and backup rules

1: set \( fw \) to all-to-all shortest paths matrix
2: **for** each node \( n \in N \)
3: **for** each outgoing link \( l \) of \( n \)
4: set \( tAdj \) to shadow copy of \( adj \)
5: remove link \( l \) from \( tAdj \)
6: set \( \{n'\} \) from \( N \) where \( nextLink = l \)
7: compute 1-to-\( \{n'\} \) shortest paths from \( tAdj \)
8: store all \( nextLink \) as \( fw[(curNode, l)][n'] \)
9: **return** \( fw \)

Complexity \( O((N+L) \cdot \text{ShortestPath}) \), min-min, no trap topologies

[van Adrichem, Iqbal and Kuipers, Backup rules in Software-Defined Networks, IEEE SDN/NFV 2016]
PER-NODE APPROACH

- Equal complexity
- Only per-link detection by BFD...

[van Adrichem, Iqbal and Kuipers, Backup rules in Software-Defined Networks, IEEE SDN/NFV 2016]
**PER-NODE APPROACH**

**Input:** Adjacency matrix \( \text{adj} = G(N, L) \)

**Output:** Forwarding matrix \( \text{fw} \) containing primary and backup rules

1: set \( \text{fw} \) to all-to-all shortest paths matrix
2: for each node \( n \in N \)
3: for each outgoing link \( l \) of \( n \)
4: set \( t\text{Adj} \) to shadow copy of \( \text{adj} \)
5: set \( n^R \) to node opposite of link \( l \)
6: remove node \( n^R \) and adjacent links from \( t\text{Adj} \)
7: set \( \{n'\} \) from \( N \) where \( \text{nextLink} = l \)
8: compute 1-to-\( \{n'\} \) shortest paths from \( t\text{Adj} \):
9: store all \( \text{nextLink} \) as \( \text{fw}[(\text{curNode}, n^R)][n'] \)
10: return \( \text{fw} \)

Complexity \( O( (N+L) \cdot \text{ShortestPath} ) \), min-min, no trap topologies

[van Adrichem, Iqbal and Kuipers, Backup rules in Software-Defined Networks, IEEE SDN/NFV 2016]
FIRST-LINK-THEN-NODE APPROACH

› Compute both, assume link-failure, escalate later

[van Adrichem, Iqbal and Kuipers, Backup rules in Software-Defined Networks, IEEE SDN/NFV 2016]
**ROUTING TABLE OPTIMIZATION**

USnet network

- 24 node, 42 link, weighted network

<table>
<thead>
<tr>
<th>Computation</th>
<th>#Flow Entries</th>
<th>-Shortest Path Default</th>
<th>-Node Failure Wildcard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest Path</td>
<td>552</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link-Failure Disjoint</td>
<td>+1606</td>
<td>+487</td>
<td></td>
</tr>
<tr>
<td>Node-Failure Disjoint</td>
<td>+2078</td>
<td>+576</td>
<td></td>
</tr>
<tr>
<td>Hybrid-Failure Disjoint</td>
<td>+3684 (sum)</td>
<td>+1293 (*)</td>
<td>+847</td>
</tr>
</tbody>
</table>

- L2 TCAMs are usually very large
- Multiple field matching TCAMs around 1000 to 10000 entries
EXPERIMENTAL EVALUATION

Forwarding Entries

Distinct Group Table Entries

- No increase in shortest path length, compared to 5% increase
- Significantly shorter secondary paths
- Significantly less (little to no) crankback routing

[van Adrichem and Kuipers, Computing backup forwarding rules in Software-Defined Networks, IEEE SDN/NFV 2016]
TARGET ACHIEVED

✓ Failure detection: Couple per-link (L=1) BFD to Fast Failover buckets

✓ Recovery: Preprogram protection paths
  ✓ Feasible on routing table complexity and optimal path lengths
  ✓ Open-sourced Proof-of-Concept Ryu OpenFlow controller module
  ✓ Contributed code to NetworkX

❑ Reconfiguration: Regain optimality of network usage
  ❑ Replace paths with new tags
  ❑ Destination-to-source reconfiguration
CONCLUSION AND FUTURE WORK

- Proposed and implemented all-to-all backup rule computation

- Overall better results than fully disjoint paths in terms of
  - Path lengths
  - Resource complexity
  - (Computation time)
THANK YOU FOR YOUR ATTENTION

For more inspiration: TIME.TNO.NL/EN