Localizing link failures in legacy and SDN networks

Basuki, Akbari Indra; Kuipers, Fernando

DOI
10.1109/RNDM.2018.8489829

Publication date
2018

Document Version
Accepted author manuscript

Published in
Proceedings of 2018 10th International Workshop on Resilient Networks Design and Modeling, RNDM 2018

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.
Localizing link failures in legacy and SDN networks

Akbari Indra Basuki and Fernando Kuipers
Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
I.B.Akbari@tudelft.nl, F.A.Kuipers@tudelft.nl

Abstract—Localizing network link failures is crucial to guaran-
tee sufficient network capacity and to efficiently manage network
resources. However, since most of today’s networks use link
aggregation to increase bandwidth, localizing a single physical
link failure within such aggregated links is challenging. In
this paper, we propose and evaluate methods, for both legacy
networks as well as SDN networks, to localize link failures in
the presence of aggregated links.

For legacy networks, we propose STreLo for localizing single
link failures. We compare STreLo to a state-of-the-art solution,
called SCMon, and show that, at the expense of using more
probe packets, it is faster and uses less MPLS labels. Since probe
packets are small, we deem the trade-off beneficial.

For SDN networks, we propose StaF, which works in a
decentralized way, requires no controller interaction, and can
adapt to topological changes. Moreover, StaF can localize multi-
link failures. Both approaches have been tested via Mininet
implementations and experiments.

I. INTRODUCTION

Today’s networks consist of high-speed lines having tens
to hundreds of Gb/s capacity per link. One single link failure
could therefore significantly reduce network capacity. In order
to timely react to link failures or reduced performance that
is indicative of (imminent) link failure, it is important to be
able to quickly and accurately pinpoint the location of a link
failure.

At the RIPE 65 and NANOG 57 meetings, Nicolas Guil-
bau and Ross Cartlidge from Google presented a problem
entitled “localizing packet loss.” They expressed that too
ten the mechanisms within devices to check their own vital
parameters, like CPU load or interface status, were inaccurate.
They therefore argued to complement those device monitoring
capabilities with so-called black-box network monitoring tools
that, based on probe packets from preselected vantage points
in the network, can infer the location of network failures. Also
Aubry et al. [2] argue that the complexity of current networks
warrants black-box network monitoring.

To illustrate the complexity in localizing link failures, we
start by highlighting the diversity in types of failures, as shown
in Fig. 1, such as (1) cable cuts, (2) partial link-bundle failures,
(3) data-plane failures, and (4) control-plane failures:

1) Cable cuts
A link can fail when an Ethernet or fiber-optic cable
is cut or damaged. Such failures could be detected via
standard protocols, such as the Link Layer Discovery
Protocol (LLDP).

2) Partial link-bundle failure
Often, several physical links are combined into a sin-
gle data-link to improve network resiliency and link
capacity, e.g. see the Link Aggregation Control Protocol
(LACP) IEEE 802.1AX. The failure of one physical link
within a link bundle cannot be detected by standard data-
link-layer protocols.

3) Data-plane failure
In some cases, an interface at a switch or router may
malfunction, e.g. by reporting that it is up, while in
reality it cannot send and/or receive any packets.

4) Control-plane failure
Network protocols are processed by a switch’s or
router’s control plane. If we use a specific protocol like
LLDP to test for link failure, but the LLDP module is
broken, a link failure may be reported, while no packet
loss has manifested.

In addition to the different types of failures, the past few
years have seen a paradigm shift in network management due
to the rise of software-defined networking (SDN). SDN net-
works differ from legacy, e.g. all-optical or Ethernet, networks
in that they logically centralize the control-plane functionality
and allow for enhanced network programmability. As such,
these different types of networks may have different monitoring
capabilities.

The main problem considered in this paper is that of
localizing link failures via one or more vantage points, which
we refer to as the Localizing Link Failure (LLF) problem. In
general, this problem is challenging not only for the reasons
Consisting of three segment IDs: Node SIDs of the link’s SIDs. Each monitoring path has a unique MPLS label stack. To identify the paths, we use node, via the shortest path, to the looping link; traversing that path will correspond to a monitoring path (from the monitoring remaining links are called looping links and each looping link network is a tree, these paths will not cover all links. The paths tree algorithm, we compute the shortest paths from each link inside the link bundles on that path, a segment ID must be inserted, which can quickly increase the number of required segment IDs. Therefore, we propose Shortest-paths Tree Loops (STreLo), which reduces the number of segment labels, when compared to SCMon, at the expense of an increased number of probe packets. Since those packets are small in size, we deem the trade-off beneficial. We will present the algorithm for one monitoring node, but the approach can be extended to multiple monitoring nodes, by assigning each monitoring node to a part of the network.

**A. STreLo**

STreLo comprises three phases: (1) path-computing phase, (2) probing phase, and (3) correlation phase, as explained below.

1) Path-computing phase: Through Dijkstra’s shortest-paths tree algorithm, we compute the shortest paths from the monitoring node (Mn) to all other nodes. Unless the network is a tree, these paths will not cover all links. The remaining links are called looping links and each looping link will correspond to a monitoring path (from the monitoring node, via the shortest path, to the looping link; traversing that link; and back via the shortest path to the monitoring node).

Considering that STreLo uses cyclic monitoring paths, the network should be three-edge-connected, else it cannot correctly localize a single link failure [17]. Algorithm 1 contains the monitoring paths computation and its respective label stacks computation. To identify the paths, we use node Segment IDs (Node SIDs) and adjacency Segment IDs (Adj SIDs). Each monitoring path has a unique MPLS label stack consisting of three segment IDs: Node SIDs of the link’s adjacent nodes, Adj SIDs of the looping link, and Node SID of the monitoring node.

**Algorithm 1 Monitoring path computation**

1: **procedure** Monitoring path computation(G, Mn)
2: \[ \text{All links} (AE) \leftarrow G.\text{links}() \]
3: \[ \text{Shortest path} (SP) \leftarrow \text{DijkstraShortestPath}(G, Mn) \]
4: \[ \text{Probe packet} (PP) \leftarrow \text{null} \]
5: for each link in AE do
6: \[ \text{if link} \in SP \text{ then} \]
7: \[ \text{link.weight} = 1 \]
8: \[ \text{else} \]
9: \[ \text{link.weight} = 65535 \]
10: \[ \text{stack} \leftarrow \text{null} \]
11: \[ \text{stack} \leftarrow \text{Nodes.SID(adjacent node)} \]
12: \[ \text{stack} \leftarrow \text{Adj.SID(link)} \]
13: \[ \text{stack} \leftarrow \text{Nodes.SID(Mn)} \]
14: \[ PP \leftarrow \text{stack} \]
15: return PP

By using high weights for looping links, we ensure that the probe packets (in the probing phase) will indeed follow the shortest paths.

2) Probing phase: The monitoring node forwards probe packets at fixed intervals, which we set to 50 ms. Every probe packet will be sent via segment routing by reading the top-most label from the stack of segment labels. If the top-most label is a node segment ID, segment routing will forward it to the corresponding destination node without popping the top-most label. The adjacent node right before the destination nodes will pop the top-most label before forwarding it to the destination node. If the label is an Adjacency segment ID, segment routing will forward the probe packet to its adjacent node by always popping the top-most label. The bottom label of all probe packets is the segment ID of the monitoring node, the last destination.

3) Correlation phase: The correlation phase, as represented by Algorithm 2, uses “alarm codes” to determine the location of a link failure. The alarm code of link A consists of the probe packets that traverse link A. If none of them returns to the monitoring node, then link A failed. For this to work, every link must have its unique alarm code. uPP in the algorithm reflects the set of unreturned probe packets.

**B. Experiments**

In this section, we compare STreLo to SCMon, on topologies from the Rocketfuel project [15]. We use Mininet to build a testbed of OpenVswitch [13] switches. Since STreLo uses a shortest path algorithm instead of a cycle cover algorithm [2] or P-Cycle algorithm [1], it can generate monitoring paths and their label stacks approximately 20 times faster than SCMon, see Table I.

STreLo sacrifices the number of probe packets in favor of a stack of a fixed number (three) of MPLS labels, irrespective of the topology. SCMon uses fewer probe packets, but needs
Table I: Time to generate monitoring paths.

<table>
<thead>
<tr>
<th>Topology</th>
<th>STreLo</th>
<th>SCMon</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVH Europe</td>
<td>0.085 ms</td>
<td>0.678 ms</td>
</tr>
<tr>
<td>RF1239</td>
<td>29.521 ms</td>
<td>593.632 ms</td>
</tr>
<tr>
<td>RF1755</td>
<td>1.593 ms</td>
<td>32.098 ms</td>
</tr>
<tr>
<td>RF3257</td>
<td>4.415 ms</td>
<td>121.027 ms</td>
</tr>
<tr>
<td>RF3967</td>
<td>1.358 ms</td>
<td>24.340 ms</td>
</tr>
</tbody>
</table>

Table II: Number of probe packets required.

<table>
<thead>
<tr>
<th>Topology</th>
<th>STreLo</th>
<th>SCMon</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVH Europe</td>
<td>160</td>
<td>-</td>
</tr>
<tr>
<td>RF1239</td>
<td>858</td>
<td>295</td>
</tr>
<tr>
<td>RF1755</td>
<td>182</td>
<td>53</td>
</tr>
<tr>
<td>RF3257</td>
<td>382</td>
<td>110</td>
</tr>
<tr>
<td>RF3967</td>
<td>152</td>
<td>65</td>
</tr>
</tbody>
</table>

Table III: Maximum link-failure localization time.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Label stacks</th>
<th>Cycles</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVH Europe</td>
<td>5</td>
<td>195</td>
<td>360</td>
</tr>
<tr>
<td>RF1239</td>
<td>5</td>
<td>34</td>
<td>130</td>
</tr>
<tr>
<td>RF1755</td>
<td>5</td>
<td>76</td>
<td>127</td>
</tr>
<tr>
<td>RF3257</td>
<td>5</td>
<td>24</td>
<td>206</td>
</tr>
</tbody>
</table>

Algorithm 2 Correlation Algorithm

1: procedure ALARM CODE - AC - GENERATION(G, PP)  
2: for each link in G.links() do  
3: AC[link] ← null  
4: for each probe in PP do  
5: for each label in probe.header do  
6: if label is a node SID then  
7: if label = bottom of stack then  
8: Node ← Adj_node(Node[SID], adjlink)  
9: for each link in [Node .. Mn] do  
10: AC[link] ← probe  
11: else  
12: for each link in [Mn .. Node[SID]] do  
13: AC[link] ← probe  
14: else  
15: adjlink ← link[SID]  
16: AC[adjlink] ← probe  
17: procedure ALARM CODE LOCALIZATION(AC, uPP)  
18: Link_fail(LF) ← null  
19: for each alarm_code in AC do  
20: if AC[link] ⊆ uPP then  
21: LF ← link  
22: Break  
23: return LF

more labels stacked on top of each other, see Table II. In networks with many link bundles, such as OVH Europe, SCMon requires many labels to operate.

Table II: Number of probe packets required.

<table>
<thead>
<tr>
<th>Topology</th>
<th>STreLo</th>
<th>SCMon</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVH Europe</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>RF1239</td>
<td>858</td>
<td>295</td>
</tr>
<tr>
<td>RF1755</td>
<td>182</td>
<td>53</td>
</tr>
<tr>
<td>RF3257</td>
<td>382</td>
<td>110</td>
</tr>
<tr>
<td>RF3967</td>
<td>152</td>
<td>65</td>
</tr>
</tbody>
</table>

Except for RF3257, STreLo also required less failure localization time, see Table III, which was derived from the longest probe travel time.

III. SOFTWARE Defined NETWORKING (SDN)

In out-of-band SDN networks, such as in data centers, the controller can test every link directly via the direct connection between a switch and the controller. We therefore only consider in-band SDN networks in this section and address the problem of localizing multiple link failures that occur at the same time.

The popular SDN southbound protocol OpenFlow [11] enables sending a unique probe packet to each link. However, constructing monitoring paths by using static flow rules is inefficient and inflexible [8], [9]. Even in the best case, each switch must install \( \approx 2E \) static flow rules, where \( E \) is the number of links. Segment routing allows to reduce the number of flow rules to \( N + I \), where \( N \) is the number of nodes, and \( I \) is the number of physical interfaces for each node, which may be much smaller than \( E \) in large networks with many link bundles. Monitoring via Per-Link Segment Routing (PLSR) can be initiated by any node without controller involvement, if it knows the network topology and corresponding segment routing configuration (node segment IDs and Adjacency segment IDs). The flow rule updates for the segment routing implementation are handled by the controller [14]. The drawback of this method is that it relies on external processes to route the probe packets. If these routes are affected by a link failure, it could take some time to converge to a new routing state, which could delay the process of localizing the failure of multiple links. Contrary to PLSR, in the following section we will propose a stateful SDN solution to investigate whether localization accuracy can be improved.

A. Stateful SDN

A stateful SDN switch is able to memorize a certain condition or network state. In SDN, statefulness can be implemented in various ways. The most common approach is to use a switch’s register to store state values, e.g. as is done in P4 switches [5], [16], and OpenState SDN [4]. Another approach is to use a learning action to install a new OpenFlow rule based on the input packet, as is done in OpenVswitch [7].

We propose a stateful flooding method to localize multi-link failures. The idea is that every data-plane tests its adjacent links and reports the result to the leader node, which is the monitoring node. Stateful flooding works by flooding the network using a single probe packet to install a new state into the data-plane of every node. The state is used to construct reporting paths to monitoring nodes without intervention of the controller. For every unique probe packet that is received by the data-plane, there are two states that must be installed: a visited state in the ingress table and a shortest-path state in
the egress table. The visited state prevents the probe packet from looping by dropping the probe packet at the second encounter. The shortest-path state keeps the interface number that received the probe packet for the first time. This number will later be used as a destination port to forward the probe packet back to the monitoring node. Our method consists of two parts, (1) statefulness and (2) flooding, and is called StaF. Statefulness is used to store information regarding adjacent link status and to prevent looping caused by the flooding process. Flooding is used to construct the monitoring paths. The entire localization process is run by the switches in a distributed fashion without involving the controller.

The localization process consists of four phases, as shown in Fig. 2:

1) **Initialization phase**
   In this phase, every switch has a pre-installed set of static flow rules to implement the stateful flooding. Every switch uses two kinds of flow rules to memorize two different states: a loop breaker state and a shortest path state. The loop breaker flow rule must reside in the ingress flow table (IT) to break loops. In the egress flow table (ET), shortest path flow rules are used to report back the result to the monitoring node.

2) **Signaling phase**
   In this phase, the monitoring node sends a single packet as a synchronization signal using stateful flooding. The signal packet is a UDP packet containing the Monitor ID (MN-ID). Every switch will run three actions after receiving the synchronization signal: 1) install the states, 2) re-flood the probe packet to any other switch, and 3) start or report link testing. In this phase, every switch must store two states, one state indicating the visiting status and another state recording the incoming port number.

3) **Testing phase**
   Link testing can be implemented in two different ways: proactive testing or reactive testing. In proactive testing, the BFD protocol [3] is used to test the network link at certain heartbeat intervals. The test result is stored within the stateful switch. The signal packet from the monitoring node will trigger the switch to immediately send the result to the monitoring node.

In reactive testing, the signal from the monitoring node must be duplicated to test all of the adjacent links. In this case, the switches will not store the test results, but only forward the adjacent link test results to the monitoring node. We have implemented this as follows: The sender node will insert its information (switch ID and outgoing port), change the packet mode into test packet, and send the packet to the respective port. Subsequently, the adjacent node will insert its information (switch ID and incoming port), change test packet mode into report packet mode, and send the packet to the sender via the input port.

4) **Reporting phase**
   Every switch that receives report packets will forward the packet via the egress flow table where there exist flow rules to report the packet to the monitoring node.

The advantage of stateful flooding is that it quickly reacts to link failures by avoiding them during state installation. The obvious disadvantage is that it needs to maintain state, the amount of which is limited by the available memory. To solve this problem, one could limit the number of requesters to a specified number $m$. This means that at any time there are at most $m$ monitoring nodes.

**B. Experiments**

We have implemented and evaluated Stateful Flooding via Mininet [10]. For our experiments, we have used a 3x3 grid topology, see Fig. 3, and two scenarios: no errors and many errors. The latter condition is where several link failures occur without partitioning the network, namely links s1-s4, s2-s5, s5-s8, and s6-s9 fail. We measure Time-of-Arrival (ToA) of the probe packets for 1 ms (Fig. 4) and 10 ms (Fig. 5) link delays.

![Figure 2: Localization phases in Stateful Flooding: 1) Initialization, 2) Signaling, 3) Testing, 4) Reporting.](image-url)
PLSR is faster than StaF, because PLSR only uses OpenFlow match and action commands. StaF, on the other hand, must install states into the data-plane, which accounts for the additional latency of $\leq 15$ ms. The states are installed through the user-space domain in OpenVSwitch, by installing a new flow rule into the flow table [7]. We argue that if the time to install states can be shortened, such as for hardware switches or SmartNICs that use registers [12], the additional latency encountered in StaF can be minimized, approaching the base forwarding speed of a typical OpenFlow match-action. Also, for when link delays are high (10 versus 1 ms), the relative difference in Time of Arrival (ToA) between PLSR and StaF diminishes.

PLSR, after one link failure, needs the controller to recompute the monitoring paths, after which other failures can be localized. For example, Fig. 6 shows that with PLSR, two failing links (marked by the vertical dashed lines around 50 and 60 ms) led it to temporarily believe that 18 links had failed. Only after a convergence time $\geq 10$ seconds (at the time indicated by the solid vertical line) did it correctly detect the failures.

In stateful flooding, the selection of forwarding and reporting paths does not depend on the controller, since they are constructed automatically by keeping state. Even though there are multiple link failures, each time the monitoring node sends a probe packet, a new path will be constructed by avoiding the link failure. Hence, it can determine the exact number of link failures immediately in the next probing session (Fig. 7).

Any limitation of StaF predominantly lies in its technical implementation of how to keep state. In OpenVSwitch, there is no mechanism to remove the state, except by letting an installed flow rule to time-out. In proactive link testing, considering that the results of link testing are stored as states, if a particular link fails, the information can only be known after $x$ seconds, where $x$ is the probing interval. $x$ cannot be smaller than 1 second, because the minimum time-out value for state keeping is 1 second. In case we want to localize link failures faster, we will have to do reactive link testing.

Reactive link testing does not require to save the link testing results. Link testing is run by duplicating the signal...
packet, sending them to the adjacent nodes, and receiving and reporting them back to the monitoring node. Fig. 8 shows that StaF can indeed be sped up by using a reactive method, for example with a 100 ms probing interval. In that case, two links that fail can be detected immediately within 100 ms.

For SDN networks, we have proposed StaF, which can localize multiple concurrent link failures and can be started from any node in the network by sending a single signal packet. Stateful flooding can operate at a speed close to that of the forwarding plane.

ACKNOWLEDGEMENT

This work is supported in part by the Indonesian Ministry of Research, Technology and Higher Education and by SURFnet.

REFERENCES


Figure 7: StaF localizing multiple link failures within the next probing interval (1 second).

Figure 8: Fast localization speed (100 ms) in StaF with reactive link testing. At two points in time, two links (i.e. 4 in total) fail.

IV. Conclusion

In this paper, we have proposed two methods for localizing link failures, namely STreLo and StaF. STreLo uses segment routing and is applicable to legacy networks. It has millisecond localization speed and is scalable. The advantage of STreLo is that it uses a fixed number of MPLS labels, so it can be applied widely in almost any existing Ethernet router that supports segment routing and MPLS.