Demo: Fast Routing-Loops Identification in Multi-Protocol Multi-Instance IP Networks

Youcef Magnouche, Sébastien Martin, Jérémie Leguay, Mei Cong, Guofeng Qian
Huawei Technologies Ltd., Paris Research Center, France.
{firstname.lastname}@huawei.com

***Abstract*—Various routing protocols are deployed to operate networks, and border routers manage the exchange of routing information. Network engineers carefully configure routing policies to ensure reliable, efficient, and secure connectivity. However, the complexity of these configurations can lead to errors and issues like routing loops. Existing control plane verification solutions offer comprehensive analysis but struggle with scalability. This demonstration presents a scalable verification tool capable of locating routing loops in large multi-protocol and multi-instance networks, demonstrating its efficiency and performance with numerical results and live verification.**

I. Introduction

Networks are generally structured into areas and domains where various instances of routing protocols (such as OSPF, ISIS, and BGP) operate. Border routers, which connect these protocol instances, use routing policies to manage the advertisement and reception of routing information. Network administrators meticulously configure these policies to ensure optimal end-to-end connectivity, considering availability, performance, and security. However, configuring these protocols is complex and error-prone. Misconfigurations in routing policies can lead to severe issues, including routing loops.

Various solutions have been employed for control plane verification. Batfish [1] simulates the behavior of individual protocols to infer data plane information, allowing for comprehensive control plane analysis. Minesweeper [2] uses a formal method based on descriptive logic to examine control plane configurations. Although these contrasting methods can verify numerous intended properties, such as node reachability, isolation, waypointing, black holes, routing loops, bounded path length, and load-balancing, they do not scale well for practical scenarios. To enhance scalability, more targeted solutions have been developed. For example, ARC [3] and Tiramisu [4] abstract the control plane with graph transformations to verify a limited set of routing properties, including security policies, reachability after failures, disjointness, and waypointing. Despite these improvements, scalability remains a challenge.

In this demonstration, we introduce a scalable verification tool specifically designed to detect routing loops in multi-instance and multi-protocol networks. This centralized management tool can identify loops in networks with up to 10,000 nodes and 1 million prefixes in an hour. According to RFC 9067 [5], a routing policy defines how routes are imported, exported, modified, and advertised between protocol instances or within a single protocol instance. The remainder of this paper begins with a use case that illustrates how routing policy misconfigurations can cause loops. We then provide a high-level overview of our algorithmic solution to verify routing policies and present the demonstrator along with results.

979-8-3503-5171-2/24/31.002024$IEEE$

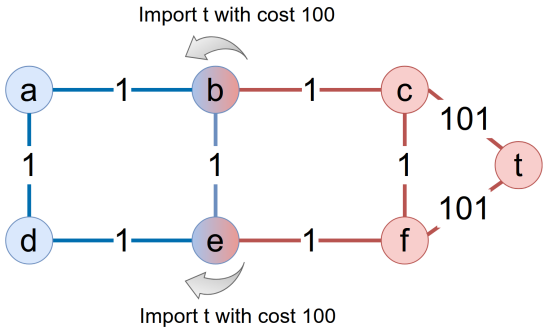

Fig. 1. Miss-configuration of two IGP instances (blue, red) inducing a routing loop for prefix t. Integers on links represent IGP costs.

A. Motivation use case

Fig. 1 shows a network consisting of two instances of IGP protocols (OSPF or ISIS). The first instance is represented in blue, and the second instance is in red. Link weights indicate IGP costs. Nodes b and e are border routers and belong to both instances. Other nodes are only visible within their respective instances. To allow an IP prefix to be reached from another instance, routing policies can be configured with an "import" policy at border routers. In the use-case, two imports allow reaching prefix t via border routers b and e, each with a cost of 100. We notice, for example, the path cost from node a to t is 101 through node b, and 102 through node e.

In this example, import costs are not properly defined and causing a loop. From the perspective of node b, there are two routes: 1) in the Red instance: $b \rightarrow c \rightarrow t$ with a cost of 102, and 2) in the Blue instance: $b \rightarrow e \rightarrow t$ with a cost of 101. Similarly, from node e's perspective, the routes are: 1) in the Red instance: $e \rightarrow f \rightarrow t$ with a cost of 102, and 2) in the Blue instance: $e \rightarrow b \rightarrow t$ with a cost of 101. Consequently, any packet to t arriving at e will be forwarded to b, and any packet to t arriving at b will be forwarded to e, causing a loop.

II. Fast Routing Loops Identification

Algorithmic framework. In the following, we propose a loops identification algorithm for a given prefix. Let us

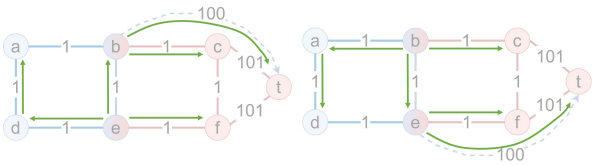

Fig. 2. Two Dijkstra trees rooted at b and e

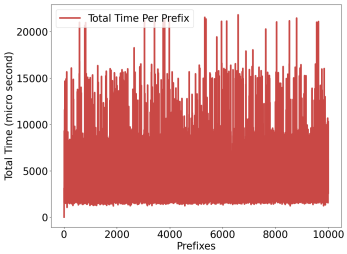

Fig. 3. Computation time per prefix

consider the example in Fig. 1. When an import of a prefix t is configured in either b or e, an LSA/LSP is propagated within the red instance to advertise the route to t. Naturally, b and e will disregard their own LSA/LSP. Consequently, the border router b believes that a route with a cost of 100 is available between e and t, while e believes that there is a route with a cost of 100 from b to t. This situation can be captured by constructing a transformed graph for each border router and each prefix. The transformed graph represents the perspective of the border router concerning a specific prefix. Fig. 2 shows the transformed networks from the perspective of b (Left) and e (Right). This allows us to design a loop identification algorithm using Dijkstra algorithm (Alg. 1).

Steps 1-9 construct a transformed graph for every border router. It represents the network from the perspective of the border router. For a border router br, on line 8 a Dijkstra tree G^{br} rooted at br is computed. On line 10, we iterate over every prefix and every border router and verify the existence of the loop using a linear-time algorithm. For a given br, we concatenate the path between border router br to br', where br' is the next border router in the path between br and t in $T^{br'}$, and then set $br = br'$. We continue until a border router is crossed twice. In this case, a loop is detected. Fig. 2 shows the two Dijkstra trees rooted in b and e. Clearly, if we merge the two trees, the cycle $b \rightarrow$e$\rightarrow b$ will be generated.

Demonstration. Fig. 4 presents the Routing Loops Identification Tool we will demonstrate. It shows a network with 5 instances and some border router configurations. 106 loops have been detected and loop n°4 to prefix 229 is displayed. We can see that it is due to a misconfigured import between OSPF-1 and ISIS-0 (protocol with instance identifier). A video presenting the tool is available here: https://tinyurl.com/Routing-Loops.

Numerical results. The tool has been tested on a large-scale instance, randomly generated with 5 instances (OSPF and ISIS), 9 980 nodes, 311 000 edges, 30 border routers, and 24 011 import policies for 10 000 prefixes. Fig. 3 shows the computation times per prefix, allowing us to identify 29 670 loops. On average 3.7 milliseconds per prefix.

Algorithm 1 Loop identification

Input:
- Graph $G = (V, E)$ and set of prefixes $R \subset V$
- Ω: Set of configuration pairs (Border router, Import cost, prefix)

Output: Loop

```
1: for each border router br ∈ V do
2:     G^br ← G
3:     for each (br', c, t) ∈ Ω do
4:         if br ≠ br' then
5:             add virtual link (br', t) to G^br of cost c
6:         end if
7:     end for
8:     T^br ← compute Dijkstra tree rooted at br in G^br
9: end for
10: for each prefix t ∈ R do
11:     for each border router br ∈ V do
12:         br' ← br
13:         while br' ≠ null do
14:             br' ← next border router in T^br' to reach t
15:             if br' was already visited then
16:                 loop = concatenation of visited sub-paths
17:                 return loop
18:             end if
19:         end while
20:     end for
21: end for
```

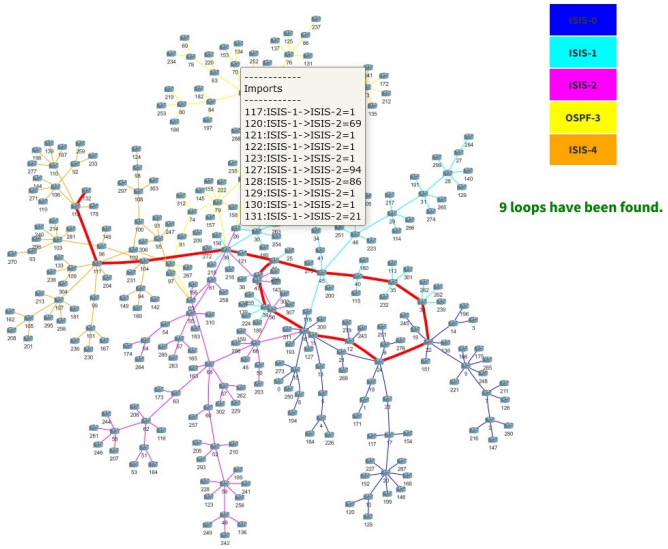

Fig. 4. Routing Loops Identification Tool

III. Conclusion

We presented a scalable verification tool to locate routing loops in large multi-protocol multi-instance networks. In contrast to existing solutions, this tool allows a fast and light loop identification without any heavy simulation of the network. The proposed solution can be extended to handle other policy parameters and other protocols such as BGP.

References

[1] Fogel A, Fung S, Pedrosa L, Walraed-Sullivan M, Govindan R, Mahajan R, Millstein T. *A general approach to network configuration analysis*. USENIX NSDI Conference. 2015.

[2] Beckett R, Gupta A, Mahajan R, Walker D. *A General Approach to Network Configuration Verification.* ACM SIGCOMM Conference. 2017.

[3] Gember-Jacobson A, Viswanathan R, Akella A, Mahajan R. *Fast control plane analysis using an abstract representation.* ACM SIGCOMM Conference. 2016.

[4] Abhashkumar A, Gember-Jacobson A, Akella A. *Tiramisu: Fast and general network verification.* USENIX NSDI conference. 2020.

[5] QU, Y., et al. RFC 9067 A YANG Data Model for Routing Policy. 2021.