BUPT T. Pan
Internet-Draft M. Gao
Intended status: Informational E. Song
Expires: April 28, 2021 Z. Bian
X. Lin
Beijing University of Posts and Telecommunications
October 25, 2020

In-band Network-Wide Telemetry
draft-tian-bupt-inwt-mechanism-policy-01

Abstract

This document describes INT-path, a cost-effective network-wide telemetry framework based on INT (In-band Network Telemetry), by decoupling the network monitoring system into a routing mechanism and a routing path generation policy. INT-path embeds SR (Source Routing) into INT probes to allow specifying the route that the probe packet takes through the network. Above this probing path control mechanism, an Euler trail-based path planning policy is developed to generate non-overlapped INT paths that cover the entire network with a minimum path number, reducing the overall telemetry overhead. INT-path is very suitable for deployment in data center networks thanks to their symmetric topologies.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

Status of This Memo

This Internet-Draft will expire on April 28, 2021.

Copyright Notice

Table of Contents

1. Introduction

At present, conducting fine-grained, network-wide traffic monitoring plays an increasingly significant role in maintaining large-scale computer networks. It enables fine-grained network-wide visibility to ease the fast detection and localization of network gray failures[Jia]. It also improves the network traffic load balancing with the prior knowledge of link congestion. The network-wide traffic monitoring can be well applied to all types of networks, especially data center networks, where traffic is highly dynamic and network failures occur silently, while user perception of network latency is expected to be bounded.

In traditional network monitoring, management protocols, such as SNMP (Simple Network Management Protocol)[RFC1157]are coarse-grained and involve a large device query latency due to the constant interaction between the control plane and the data plane. To ameliorate the performance issue, INT is proposed to achieve fine-grained network monitoring. INT allows packets to query device-internal states such as queue depth, queuing latency when they pass through the data plane pipeline, without requiring additional intervention from the control plane CPU. At the last hop of the path, the packet containing the end-to-end monitoring data can be sent to a remote central controller for data analysis. INT can print device-internal states to either specified flows of traffic or additionally introduced probe packets. In this document, our proposal relies on INT’s probe packet mode.

INT is essentially an underlying primitive that need the support of special hardware for device-internal state exposure. To achieve network-wide traffic monitoring, INT further requires a high-level orchestration built upon it. In our proposal, multiple controllable probing paths are generated to monitor the entire network. To reduce the telemetry overhead of additional probe packets for the original network, such orchestration should be better follow the following design principles:

o It should use non-overlapped probing paths to completely cover all network edges to reduce unnecessary bandwidth occupation.

o It should keep the path number as small as possible to lessen the processing overhead of the telemetry workload sent to the controller.

This document addresses the problem of "In-band Network-wide Telemetry", and proposes INT-path, a telemetry framework to achieve lightweight network-wide traffic monitoring. Specifically, we embed SR into INT probes to allow specifying the route the probe packet takes through the network. Based on the routing mechanism, we design two path planning policies to generate multiple non-overlapped INT paths that cover the entire network. The first is based on DFS (Depth-First Search) which is straightforward but computationally-efficient. The second is an Euler trail-based algorithm that can optimally generate non-overlapped INT paths with a minimum path number.

2. Problem Statement

This document proposes INT-path to solve the following technical challenges of building a network-wide telemetry system based on INT:

- Uncontrollable probing path. As an underlying primitive, INT only defines how to extract device-internal states using probe packets. However, the probe packet itself cannot proactively decide which path to monitor since it does not have any path-related prior knowledge. If the INT header is embedded in an IP packet, the probing path will be passively decided by its destination IP address together with the routing table in each network device, leaving probing path totally uncontrollable by the INT packet sender. Given the uncontrollable probing path, it is not easy to work out purposive strategies to optimally generate multiple probing paths for achieving cost-effective network-wide telemetry.

- Telemetry overhead. During traffic monitoring, we need periodically perform the INT operation at all devices and notify the controller about the underlying traffic status. However, straightforwardly conducting INT at each device or device chain incurs significant performance overhead: (1) INT will inject probes into the network, which will occupy a fraction of link bandwidth (the finer INT sampling granularity, the more bandwidth will be consumed). (2) INT agents must be deployed for probe generation and collection as the extra cost (the higher number of separated INT paths, the more INT agents need to be deployed). (3) Besides, as the INT agent number grows, the controller will suffer from a performance penalty for handling increased telemetry workload sent from those INT agents.

To tackle these problems, this document proposes INT-path, a framework for network-wide telemetry, by decoupling the system into a routing mechanism and a route generation policy. The underlying mechanism allows network operators or INT agents to specify a particular path for monitoring, addressing the uncontrollable path issue (see section3 for details). The policy built upon the mechanism generates multiple INT paths to cover the entire network and a good policy is expected to minimize the telemetry overhead with the least path overlapping as well as the minimized total path number (see section4 and section5 for details).

3. Source Routing-based Path Monitoring (Mechanism)

This document addresses the uncontrollable path issue via the technique of SR. Figure 1 shows the SR-based telemetry architecture as well as the probe packet format. Although SR is not a new technique, we innovate to couple it with the INT probe using the P4 language [P4] to implement user-specified or on demand path monitoring.

```
                             |<--   Variable length   -->|
                             | 22B  |
                             +------+------+------+------+
                             | INTn | .... | INT2 | INT1 |
                             +------+------+------+------+
                              \                         /
                                 \                    /
                                   \               /
                                      \         /
                 +-------+------+------+-------+-------+
                 |  ETH  |  IP  |  SR  |  INT  |  UDP  |
                 +-------+------+------+-------+-------+
                               /        \
                            /              \
                         /                    \
                      /                          \
                    +-------+------+-------+-------+
                    | Portn | .... | Port2 | Port1 |
                    +-------+------+-------+-------+
                    |  4b   |
                    |<----------  512b   --------->|
```

Figure 1. Probe packet format

3.1. Packet Header Format

Theoretically, the SR label stack and the INT label stack can be placed above either the IP header or the UDP header. If placed above the UDP header, they need to occupy an extra port number, which is likely to conflict with the port number chosen by the end hosts for a certain application. While, if placed above the IP header, they only occupy an IP protocol number, and then we can choose an unused protocol number according to existing RFC specifications. Therefore, the program chosen by this document is to place the SR label stack and the INT label stack above the IP header as shown in Figure1. One thing to declare is that although we design a customized header format as follows for the probe packets, the network devices can still correctly forward these packets provided that protocol-independent forwarding is supported.

- DP: DP means Destination Port which is set to "SR_INT_PORT" to inform the packet parser that it is an SR-INT probe (i.e., INT probe packet with an SR label stack).

- DIP: DIP means destination IP address of the probe packet which is set using controller’s IP to guarantee that the probe packet will finally be forwarded to the controller for further analysis.

- SR: A 512-bit space is reserved for the SR label stack above the IP header. A 4-bit space is allocated for each SR label to denote the router output port ID thus can maximally support 16 output ports for each router.

- INT: Above the fixed-length SR label stack, a variable-length INT label stack is allocated. Each INT label occupies 22B containing the information such as device ID, ingress/egress port, egress queue depth. Since P4 currently does not well support parsing double variable length stacks in the packet header, the SR label stack with a fixed length is statically allocated and the right shift operation is used to perform the "stack pop" behavior.

3.2. Forwarding Behaviors

In the SR-based telemetry architecture, three types of logic routers are proposed with different functionalities: the INT generator, the INT forwarder and the INT collector (as shown in Figure 2).

```
               +--------+                         +--------+
               |End Host|                         |End Host|
               +--------+                         +--------+
                    |                                 |
                    |                                 |
               +---------+                       +---------+
               |   INT   |                       |   INT   |
               |generator|                       |collector|
               +---------+                       +---------+
                |   |           +---------+           |   |
                |   |           |   INT   |           |   |
                |   |           |forwarder|           |   |
                |   |           +---------+           |   |
                |   |          /           \          |   |
                |   |         /             \         |   |
                |  +---------+               +---------+  |
                |  |   INT   |               |   INT   |  |
                |  |forwarder|               |forwarder|  |
                |  +---------+               +---------+  |
                |             \             /             |
                |              \           /              |
                |               +---------+               |
                |               |   INT   |               |
                |               |forwarder|               |
                |               +---------+               |
                |                                         |
               +-------------------------------------------+
               |                 Controller                |
               +-------------------------------------------+
```

Figure 2. Source routing-based path monitoring

o INT generator: The INT generator is responsible for spawning the SR-INT probe packets at the first hop of the monitoring path. Since packet generation directly from the data plane is currently undefined in P4, we consider a workaround to periodically generate "empty" probes from the outside by either the router/switch CPU or a host attached to the network device. When the probe arrives at the data plane, the INT generator will rewrite its packet header to allocate the SR label stack and add its local INT information using header.setValid() in P4 before forwarding the packet. Specifically, the INT generator will push the output port IDs into the SR label stack in the packet header. The sequence of the output port IDs (i.e., how to forward the packet across the network) is predetermined at the controller via centralized route calculation.

- INT forwarder: The INT forwarder performs packet forwarding of either the SR-INT probes or the background traffic, according to the DP number of the incoming traffic. If the DP is "SR_INT_PORT", the INT forwarder will perform label switching and forward the packet only according to the output port ID popped from the SR label stack. The SR label is popped once at a router by right shifting the SR header by 4 bits at each hop. Besides, the INT forwarder will also push its local INT information into the INT label stack before forwarding the probe.

- INT collector: At the last hop of the monitoring path, since the DIP is filled with controller’s IP address, the INT collector will finally forward the probe packet to the controller for further analysis.

4. DFS-based path planning algorithm

4.1. Algorithm Outline

In this section, we propose a simple algorithm based on DFS.

When traversing a tree or a graph, DFS starts at the root and explores as far as possible along each branch before backtracking. The basic idea of the DFS-based path planning algorithm is to consecutively add the visited vertices into the current path before backtracking; if we have nowhere to go and have to backtrack, we just create a new path and add the fork vertex (the first vertex along the backtracking path that has unvisited edges) as the first node of the new path. After all the edges are visited in the DFS order, we can extract multiple non-overlapped paths covering the entire graph.

A recursive version of the DFS-based path planning algorithm is as follows:

- Step1: Choose v0 as the first vertex of the depth-first traversal and start the algorithm.

- Step2: Select one of v0’s adjacency vertices v1 and mark the edge between v0 and v1 as visited, add v0 and v1 into the current INT path; continue to search for v1’s adjacency vertex v2 and mark the edge between v1 and v2 as visited, add v2 into the current INT path; continue to search for v2’s adjacency vertex v3 and mark the edge between v2 and v3 as visited, add v3 into the current INT path; and continue until a vertex vi has no adjacency vertices that has unvisited edges, then mark the vertex vi as visited, and store the current INT path into set INT path.

- o Step3: Backtrack at the visited vertex vi until a previous vertex that has unvisited edges is found, and find an adjacency vertex of the previous vertex and mark the edge between them as visited, then add the previous vertex and the adjacency vertex into a new current INT path, that is, continue as step2. If all the adjacency vertices of the previous vertex have been accessed, then mark the previous vertex as visited. Step3 continues until all of the vertices that are reachable by backtracking from vi are visited.

- o Step4: If there are any unvisited vertices, this algorithm selects one of them as a new source and repeats step2 and step3 from that vertex. Step4 continues until every edge and every vertex has been visited. Then the set INT path is the multiple non-overlapped paths covering the entire graph.

4.2. Example

We show an algorithm example on a network graph of five devices as shown in Figure 3.

```
                       +----+     +----+     +----+
                       | v0 |-----| v1 |-----| v2 |
                       +----+     +-+--+     +--+-+
                                    |       /   |
                                    |      /    |
                                    |     /     |
                                    |    /      |
                                    |   /       |
                                  +-+--+     +--+-+
                                  | v3 |-----| v4 |
                                  +----+     +----+
```

Figure 3. Depth-first graph traversal

At the beginning, v0 is pushed into the call stack, path1 = {v0, v1} and the edge between v0 and v1 is marked as visited. The path1 expands as more and more vertices are visited in the DFS order. When path1 expands to {v0, v1, v2, v3, v1}, we have nowhere to go and have to find the fork vertex. At this time, we pop v1 from the stack and return to v3, which has an unvisited edge to v4 (notice that the call stack push/pop operations are implicitly performed during recursion). Then, we identify v3 as the fork vertex because it is the first vertex along the backtracking path that has unvisited edges. Based on v3, we create a new path as path2 = {v3, v4}. When path2 expands to {v3, v4, v2}, we have again nowhere to go and have to backtrack. But at this time, although we check all the vertices popped from the call stack, we still cannot find any fork vertex. The recursion halts when the call stack finally becomes empty. At last, we extract two non-overlapped INT probing paths (i.e., path1 and path2).

5. Euler trail-based path planning algorithm

5.1. Algorithm Outline

Although the DFS-based path planning algorithm is computationally-efficient, it has no guarantee to minimize the number of generated paths, which will potentially increase the telemetry overhead, especially the telemetry workload at the centralized. Here, we propose an optimal path planning algorithm taking advantage of the mathematical properties of the Euler trail/circuit.

In fact, the mathematical properties of the Euler trail/circuit already indicated the theoretical value of the minimum non-overlapped path number for covering a given graph. To achieve the theoretical minimum, each extracted path from a graph should start from one odd vertex and end at another odd vertex. In other words, removing one such path from a graph will eliminate a pair of odd vertices from that graph. According to the above observation, we devise an Euler trail-based algorithm to iteratively extract a path between a pair of odd vertices until all the vertices/edges are extracted from the original graph.

Although the algorithm sounds rather straightforward as an iterative path extraction process, the devil lies in the detail of dealing with several boundary cases. To be more specific, the devil lies in the possibility that an extracted path can split one connected graph into multiple subgraphs, which definitely complicates the iterative path extraction process.

Next we explain the optimal algorithm in detail. We use G to represent the network graph, which is initialized to be one connected graph and may also become multiple disconnected subgraphs caused by path extraction during algorithm iteration. We use Q to represent the path set which is initialized as an empty set and will finally contain the generated non-overlapped INT paths. We use G-p to represent extracting a path p from the graph G which will possibly further split the graph(s) G into more subgraphs.

Actually, without considering the complexity of graph split, for a given connected graph, there are mainly three different cases for path extraction. We propose solutions in each of these three cases as follows:

- The first case is: The graph does not contain any odd vertex. We can extract an Euler circuit from the graph, which will traverse every vertex of the graph. The Euler circuit can be found with the Hierholzer’s algorithm [Hierholzer] and inserted into Q. The Hierholzer’s algorithm is an efficient algorithm for finding Euler trails and Euler circuits.

o The second case is: The graph contains two odd vertices. We just find an Euler trail between the two odd vertices as the path to be extracted. Under this circumstance, an Euler trail can also be found with the Hierholzer’s algorithm and inserted into Q.

o The last case is: The graph contains more than two odd vertices. We should extract an Euler trail between any pair of odd vertices. In this case, the specific algorithm is as follows:

Step1: If G is not empty, perform the following steps. Otherwise the algorithm terminates.

Step2: Choose two odd vertices randomly and find a path p to connect the pair of odd vertices with the Dijkstra’s algorithm or any other algorithms. Then delete the edges along path p from G, and add path p into Q. After this, if the graph(s) in G have been broken into multiple disconnected subgraphs, then go to step3. If the graph in G has not been broken into multiple disconnected subgraphs, then go to step1.

Step3: Use S to store the disconnected subgraphs split from G. Then, select graphs with no odd vertex from S and store them into T. If neither set T nor set Q is an empty set, then go to step4. If set T is not empty and set Q is empty, each subgraph in the set T is processed in the same way as in the first case, that is, using Hierholzer’s algorithm to extract an Euler circuit from the subgraph, then delete the edges along the Euler circuit from G and add the Euler circuit into Q. For each subgraph in set S, if it has two odd vertices, the subgraph is processed in the same way as in the second case, that is, using Hierholzer’s algorithm to find an Euler trail between the two odd vertices as the path to be extracted, then delete the edges along the Euler trail from G and add the Euler trail into Q. For each subgraph in set S, if it has more than two odd vertices, then go to step2.

Step4: For each graph in set T, generate an Euler circuit T_circuit for its full edge coverage. Then search the current Q to find a path T_path having at least a same vertex with T_circuit. Then, connect T_circuit with T_path to create a longer new path, and replace the original T_path with the new path in Q, and delete the edges along path T_circuit from G.

5.2. Example

Figure 4 shows a path extraction process of the Euler trail-based algorithm. At the start, there is only one connected graph G1 {1,2,3,4,5,6,7} with 4 odd vertices. Since the number of the odd vertices is larger than 2, we randomly choose two odd vertices (1 and 3), extract a path 1-4-3 from G1 and insert the path into Q. The above path extraction behavior will split the original G1 into two subgraphs G1 {1,2,3} and G2 {4,5,6,7}. Since G1 has no odd vertex and set Q is not empty, we paste the path 1-4-3 in Q with the Euler circuit 1-2-3-1 generated from G1 to create a new path 1-2-3-1-4-3. The new path will replace the original path 1-4-3 in Q. After the path paste, there is only one graph G1 {4,5,6,7} with 2 odd vertices. We use the Hierholzer’s algorithm to find its Euler trail 5-4-6-5-7-6 as the second INT path in Q. The algorithm halts after all the paths are extracted.

```
                      +----+3+----+   +----+6+----+
                     /      +      \ /      +      \
                    +       |       +       |       +
                    2       |       4       |       7
                    +       |       +       |       +
                     \      +      / \      +      /
                      +----+1+----+   +----+5+----+
                                   (a)
                           G1={1,2,3,4,5,6,7};
                            S={G1}, T=Empty;
                              odd_num=4>2;
                               Q={1-4-3};

                      +----+3         +----+6+----+
                     /      +        /      +      \
                    +       |       +       |       +
                    2       |       4       |       7
                    +       |       +       |       +
                     \      +        \      +      /
                      +----+1         +----+5+----+
                                   (b)
                         G1={1,2,3},G2={4,5,6,7};
                            S={G1,G2}, T={G1};
                       T_path=1-4-3,T_circuit=1-2-3-1;
                            path=1-2-3-1-4-3;
                             Q={1-2-3-1-4-3};

                              +----+6+----+
                             /      +      \
                            +       |       +
                            4       |       7
                            +       |       +
                             \      +      /
                              +----+5+----+
                                   (c)
                              G1={4,5,6,7};
                             S={G2}, T=Empty;
                               odd_num=2;
                        Q={1-2-3-1-4-3,5-4-6-5-7-6};
```

Figure 4. Path extraction process of the Euler trail-based algorithm

6. IANA Considerations

This document introduces no new security issues.

7. Security Considerations

This document makes no request of IANA.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

8.2. Informative References

[Hierholzer] Hierholzer, H. and W. Wiener, "Ueber die Moeglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren", March 1873, <https://doi.org/10.1007/BF01442866>.

[Jia] Jia, CH., Pan, T., Bian, ZZ., Lin, XC., Song, EG., Xu, C., Huang, T., and YJ. Liu, "Rapid Detection and Localization of Gray Failures in Data Centers via In-band Network Telemetry", April 2020, <https://doi.org/10.1109/NOMS47738.2020.9110326>.

[P4] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., and D. Talayco, "P4: programming protocol-independent packet processors", July 2014, <https://doi.org/10.1145/2656877.2656890>.

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, DOI 10.17487/RFC1157, May 1990, <https://www.rfc-editor.org/info/rfc1157>.

Authors’ Addresses

Tian Pan
Beijing University of Posts and Telecommunications
Beijing
China

Email: pan@bupt.edu.cn

Minglan Gao
Beijing University of Posts and Telecommunications
China

Email: gml@bupt.edu.cn

Enge Song
Beijing University of Posts and Telecommunications
China

Email: songenge@bupt.edu.cn

Zizheng Bian
Beijing University of Posts and Telecommunications
China

Email: zizheng_bian@bupt.edu.cn

Xingchen Lin
Beijing University of Posts and Telecommunications
China

Email: linxchen3907@163.com