Mathematical Biosciences and Engineering

MBE, 20(11): 19468–19484.
DOI: 10.3934/mbe.2023862
Received: 06 July 2023
Revised: 15 September 2023
Accepted: 06 October 2023
Published: 20 October 2023

http://www.aimspress.com/journal/MBE

Research article

A lightweight path consistency verification based on INT in SDN

Ping Wu[1,*], Yuwei Shang[2], Shuaitao Bai[2], Lingjian Cheng[2] and Huilin Tang[1]

[1] Beijing Yungu Kechuang Information Technology Co.,Ltd. Beijing 100036, China
[2] China Electric Power Research Institute Co.,Ltd. Beijing 100192, China

* **Correspondence:** Email: wpieucs@sina.com.

Abstract: The existing path consistency verification solutions in software-defined networking (SDN) were implemented by proactive injecting large number of probing packets or by embedding linear-scale tags as the path lengthens, which incurred significant bandwidth and communication overhead. A lightweight path consistency validation mechanism based on in-band network telemetry (INT) in SDN is proposed. Based on INT, in the scheme, the ingress switch inserts a telemetry instruction header with probability, each subsequent switch updates the telemetry data using a uniform sampling algorithm and only carries partial path information in INT packet to keep the head space size constant, the egress switch reports the final sampled telemetry data to the controller to verify the path compliance according to aggregated telemetry data. A heuristic flow selection algorithm is proposed to implement network-level path consistency validation. The proposed scheme was implemented and evaluated. The analyses and experiments demonstrate the proposed mechanism effectively limits the packet head overhead and introduces less than 7% of additional forwarding delays and 6% of throughput degradation at most.

Keywords: software-defined networking; path consistency verification; in-band network telemetry; uniform sampling algorithm

1. Introduction

Software-defined networking [1] (SDN) is one of the most promising architectures for next-generation computer networks. SDN decouples the rigid network infrastructure into a control plane and data plane to dynamically control network traffic via a programmable control plane. In SDN, the centralized control plane acts as the “brain” of the network to manage and control the network traffic,

while the data plane acts as the "limb" of the network to process each flow according to the decision of the control plane, the control plane and the data plane interacts through the south interface protocol.

Open network programming interface in SDN promotes the innovation of network technology, and also greatly accelerates the development of the next generation of network technology. Owing to the decoupling of the control plane and the data plane, SDN always assumes that the rules installed by the control plane can be correct and enforced by the data plane. However, this assumption is broken by possible hardware or software drawbacks of switches and operating systems, or errors caused by network configuration. The serious consequence is that packets of a flow may deviate from their authorized path to violate critical network security policies such as access control [2]. The current network architecture of SDN does not guarantee that packets of a flow follow the path authorized by the controller. The controller is ignorant about the true forwarding path of the packets, the reason is that SDN networks lack tools to ensure that the data plane follows policies or proactively verify network behavior.

In SDN, most of the existing path consistency verification solutions are implemented by injecting a large number of probing packets into the network, or embedding linear-scale fields into the packet header space as the forwarding path lengthens. However, there exist many differences between active probing packets injected into the network with the real network traffic. The observed value cannot represent the real situation of the network operation, nor can it reflect the real-time status of the network. A large number of probing packets injected is not the actual traffic, so additional network communication overhead is introduced [3]. Embedding linear-scale network status tags in the header space as the path lengthens also significantly increases the computing and bandwidth overhead of the network [4].

To address the issue of higher network communication overhead produced via injecting a large number of probing packets and the computation and bandwidth overhead introduced by embedding linear-scale fields into the header space as the path lengthens, making use of the typical traits of programmability and centralized control of SDN, LPV, a lightweight path consistency verification mechanism based on in-band telemetry (INT) [5] in SDN is proposed. To summarize, the contributions of the paper include the following threefold.

1) We proposed LPV, a lightweight path consistency verification mechanism in SDN. LPV is based on in-band network telemetry technology, which combines packet forwarding with network measurement, limiting the length of tags inserted in the packet header space efficiently, and implementing path consistency validation. LPV overhead is constant and is independent of the path length, which can efficiently forward the packets and verify path consistency in approximately real time.

2) We propose a heuristic-based flow selection algorithm to realize the network-level path consistency verification.

3) By extending the programmable protocol-independent packet processor model P4 [6], we implement LPV, evaluate its performance in a virtual network environment and demonstrate the effectiveness of the proposed mechanism.

The rest of the article is organized as follows. Section 2 introduces the related works. In Section 3, we present the lightweight in-band telemetry-based path consistency verification mechanism LPV in SDN. In Section 4, we implement and perform experiments to evaluate the performance of LPV via the extended P4 software switch, and Section 5 concludes the paper.

2. Related works

The traits of centralized control and the decoupling of the control plane and the data plane in SDN make the network programmable, but also accelerate the issue of path consistency between the control plane with the data plane, i.e., the actual forwarding path of the data plane is not consistent with the path intended or authorized by the control plane. To address the problem, many solutions have been proposed in the past years.

Monocle [7] transforms the switch forwarding rules to a boolean satisfiability problem, which injects probing packets into the network for specific flow rules and can implement consistency verification of installed flow rules in the steady status of the network. VeriDP [8] injects probing packets into the network and collects the path information of the probing data, comparing it with the expected path in the control plane, which can effectively verify the transmission path of the real packets transferred. Based on P4, P4Consist [9] proposed a consistency verification method of the control and data plane for SDN, the method is similar to [7,8], and a large number of probing packets need to be injected into the network to validate all rules installed and transmission paths. ATPG [10] checks data reachability from source to destination by injecting probing data into the network and cannot be used to validate packet transmission path consistency, while RuleChecker [11] checks consistency between the real data plane switch rules with the control plane rules expected. The literature [7–11] all verify the flow transmission path or validate the consistency of rules installed by injecting probing packets into the network, which degrade the communication efficiency of the network, and increase data plane overhead. While the status of the network that packets injected and the status of network of actual traffic are not the same, the detection result is not real-time.

The P4-based in-band telemetry technology (INT) in SDN has attracted much attention in academia and industry in recent years [12]. INT collects the status of switches in the data plane and embeds the status information into packets forwarded, which has the advantages of real-time measurement and fine granularity.

By partitioning network status into two categories: count-based network status and threshold-based network status, Wang [13] proposed a mechanism for tracing the packet rule matched. The database in the scheme records all the rules issued to the switches, and the rule manager assigns a unique ID to each rule issued in the network, when an INT package matches a rule on the switch, the switch copies the unique rule ID to the corresponding field of the INT header and controller collect all IDs to validate consistency of rules. To reduce the network communication overhead, sINT [14] maintains a ratio table at the INT source node to adjust the packet sampling probability and insert telemetry instructions according to the variation of the network status.

FS-INT [15] classifies network measurement methods into two types of rate-based and event-based. The strategy of rate-based is similar to sINT, and the event-based is similar to [13], which inserts telemetry information into the packet header space according to the policies to detect the data transmission path.

Based on probing and in-band telemetry technology, Fast-INT [16] realizes an efficient network monitoring framework combined with reinforcement learning, whose goal is to design a lightweight INT network collection framework by implementing a dynamic collection of network status information.

The path consistency verification mechanisms above are based on in-band telemetry which embeds the telemetry data into the packet header space and acquires the network status of each switch

on the path, all can accurately collect and verify the real path of packet transmission. However, these INT-based methods are implemented by increasing the size of the packet header that is linear to the path length. First, it reduces network communication goodput for a lesser fraction of the bytes in a packet now becoming usable for transferring real traffic. Second, the header space size increasing beyond MTU (Maximum Transport Unit) fragments the packet, which introduces significant overhead of packet reassembly and potentially forwarding latency, resulting in degradation of network performance [3].

In the paper, we aim to verify path compliance while reducing the network overhead of the data plane as much as possible. We propose a lightweight path consistency validation mechanism LPV in SDN. Based on in-band telemetry, LPV effectively limits the header space overhead via the uniform telemetry data sampling algorithm proposed, and we design a heuristic flow selection algorithm, implementing network-level path consistency verification.

3. Design of LPV

The probe-based mechanisms increase the network communication overhead of the data plane and cannot reflect the network real-time status, while the linear-scale tags inserted in INT header space based on in-band telemetry mechanism will inevitably increase the flow forwarding delay and degrade network service performance [4]. How to address the above problems, by introducing limited network overhead and realizing approximate real-time verification of flow path consistency is the main goal of this paper, this section will elaborate on how to solve these design challenges.

Section 3.1 summarizes the proposed mechanism of LPV, and Sections 3.2 and 3.3 respectively elaborate on the path consistency verification based on a uniform sampling algorithm and network-level path verification based on a heuristic flow selection algorithm.

3.1. Sketch of the LPV

As mentioned above, accurate and detailed path information can be obtained by INT of per-flow per-packet. INT enables network devices to embed telemetry information directly into delivered packets and can obtain the near-real-time microscopic visibility of the network. Of course, it also incurs significant overhead. However, most applications should not need accurate results, but only the sketchy results of path sequence operation.

Based on INT, we aim at verifying path consistency as well as reducing the network overhead of the data plane as much as possible, especially the packet bandwidth cost of the network. Under the circumstances of any packet transmission path length L, the size of its INT packet header space in LPV is fixed and remains constant, and will not increase the telemetry data linearly due to path length increasing.

In short, LPV fragments the complete path information into multiple INT packets by uniformly telemetry data sampling, each INT packet carries a fragment of the path information to compress packet head overhead, and the controller verifies path consistency through these aggregated partial path fragments. Table 1 below shows some of the relevant notations used in the paper.

The overall architecture of LPV is shown in Figure 1.

Table 1. Some notations in LPV.

Notation	Description
sw_i	Switch identification in data plane
inport	Ingress port of packet passing through a switch
egport	Egress port of packet passing through a switch
hops	Total hops of a packet forwarding so far
nhop	Hops of when a packet passing through a switch *sw*
L	Path length a packet transferring
random( )	Random function
P	A packet
flow	Flow identification
Ψ	Set of active flows in the network
F	Set of selected active flows in the network

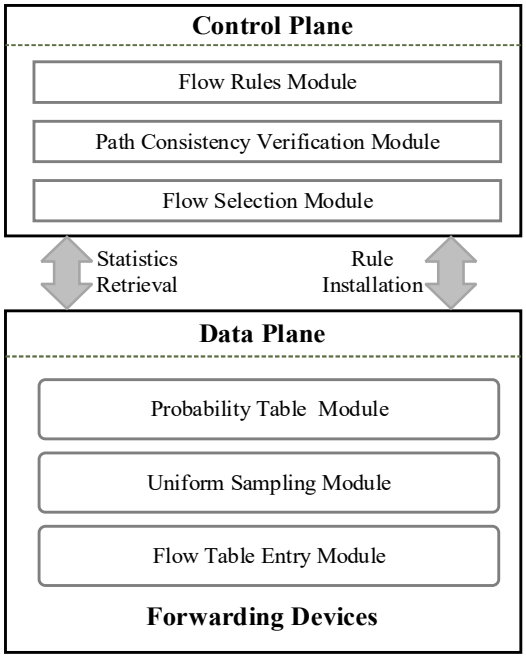

Figure 1. The architecture of LPV.

In the data plane, there are three core modules.

Probability Table Module: The INT source switch maintains a probability table, and according to the table, the INT source switch inserts telemetry instruction with probability.

Uniform Sampling Module: If a packet is an INT packet, according to the uniform sampling algorithm, the switch updates the telemetry data or keeps the telemetry data unchanged.

Flow Table Entry Module: The module stores flow rules installed by the control plane, and the switch forwards packets based on the rules installed.

In the control plane, there are three modules also.

Flow Rules Module: For a new flow, the control layer computes the path of flow transmission and installs flow forwarding rules for the switches in the data plane.

Path Consistency Verification Module: The controller retrieves aggregated INT packets of a

flow from the INT sink, to verify the real forwarding path.

Flow Selection Module: By a heuristic flow selection algorithm, the control plane implements network-level path consistency validation.

3.2. Flow-level path validation

As shown in Figure 2, a packet of a flow transfers from source S to destination D, and the controller calculates the forwarding path for the flow and installs the flow rules.

A packet enters the INT source node, the first network device on the packet forwarding path, which starts INT and embeds the telemetry command (INT header) (Based on predefined probability) and data (telemetry info or telemetry data) in the packet header space.

INT transit node is a network switch located on the packet forwarding path. By sampling, based on probability, the transit node updates the telemetry information in the received INT packet header space or keeps the telemetry information of the packet unchanged. INT sink Node (INT receiver), as the last INT node on the packet forwarding path, the INT receiver executes the same operation as the INT transit node, and removes all the telemetry information from the packet, constructing an INT report and sending the INT report to the controller. The INT sink node can be configured to send all or selective partial reports to the controller according to a predefined policy. The controller verifies the packet forwarding path consistency after aggregating a certain amount of partial path information of the flow.

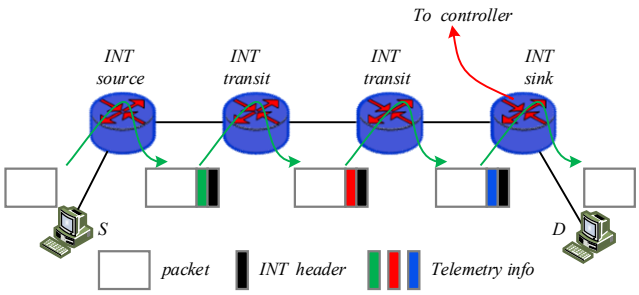

Figure 2. Packet forwarding of LPV.

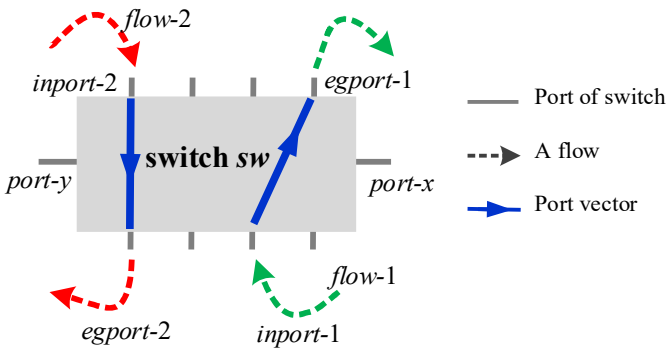

Figure 3. A packet passing port vector of switch *sw*.

Definition 1: The transferring path of a packet consists of a series of nodes from the source to the destination $\{(R_1, nhop), (R_2, nhop), \cdots\cdots, (R_L, nhop)\}$, and here $R_i = (sw_i, <inport, egport>)$, sw_i is switch identification, $inport$ and $egport$ are ingress port and egress port of a switch when a packet passing through, and $nhop$ is hops when the packet has reached sw_i.

As shown in Figure 3 above, the switch sw contains several network interfaces, there exist two flows of *flow-1* and *flow-2* passing through the switch sw. The two flows' ingress and egress port vector are $<inport-1, egport-1>$ and $<inport-2, egport-2>$, respectively.

During in transmission, only when a packet passes through the network in order as $\{(R_1, nhop), (R_2, nhop), \cdots\cdots, (R_L, nhop)\}$, the real forwarding path of the packet is consistent with the expected path of the controller authorizing.

As mentioned above, the mechanism of LPV proposes a uniform telemetry information updating sampling algorithm to fragment the complete path information into multiple INT packets of a flow, each INT packet carries only a fragment of complete path information. Based on the shared key *K* between the controller with the switch, the switch calculates the MAC (Message Authentication Code) of the path information fragment and the packet. The controller aggregates the path information fragment and verifies its validity. When the aggregated path fragment information meets with a predefined condition, the controller verifies the transmission path consistency. Table 2 shows the telemetry service primitive used in LPV.

Table 2. Service primitive used in LPV.

Telemetry data	**Description**
Switch ID	Identifier associated with a device
Ingress Port ID	Identifier of the packet's ingress port
Egress Port ID	Identifier of the packet's egress port
Hops	Hops of a packet forwarding
MAC	Message Authentication Code

INT source node maintains a flow telemetry instruction insertion probability table, as shown in Table 3 below, when a flow packet *P* enters the INT source node, INT source node inserts the telemetry instruction according to probability α.

Table 3. Probability table of telemetry instructions inserted.

A flow	**Probability table**
flow1	α_1
flow2	α_2
.....	.....
flown	α_n

When a packet embedded telemetry instruction in the header space transfers from the source to the destination switch node R_i （$1 \le i \le L$）, R_i will update the telemetry data in the packet *P* header space with probability $1/i$. As shown in Algorithm 1.

Algorithm 1. The uniform sampling algorithm of LPV.

Input：Packet *P*

Output: Switch telemetry data

INT source

1) ***if*** ($random(0,1) < \alpha$) ***then***
2) $p.sw = sw_1$
3) $p.inport = inport$
4) $p.egport = egport$
5) $p.hops = 1$ // *hops of Packet P has transmitted*
6) $p.nhop = 1$ // *hops when Packet P reached*
// *switch* sw.
7) ***endif***

INT transit and sink

1) $++ p.hops$
2) $\lambda = random([1, p.hops])$
3) ***if*** $\lambda == 1$ ***then***
4) $update(P)$
5) $p.nhop = p.hops$
6) ***endif***

Theorem 1: In Algorithm 1, on the path of $\{R_1, R_2, \cdots\cdots, R_L\}$, the node $R_i\ (1 \le i \le L)$ updates the telemetry information in the header space of packet *P* with probability $1/i$. On any forwarding path of length L, the controller only needs expectation value of L INT packets to verify the complete and real path information of a flow.

Proof: For the INT source node, when the embedded fragment of path information is not updated by all subsequent nodes $R_k\ (1 < k \le L)$ on the path, the probability is:

$$\Pr = 1*(1-\frac{1}{2})*(1-\frac{1}{3})\cdots\cdots(1-\frac{1}{L}) = \frac{1}{L} \tag{1}$$

For the INT transit node, when the node R_i has updated the telemetry information with probability $1/i$, and the information is not updated by subsequent nodes $R_k\ (i+1 \le k \le L)$, the probability is:

$$\Pr = \frac{1}{i}*(1-\frac{1}{i+1})*(1-\frac{1}{i+2})\cdots\cdots(1-\frac{1}{L}) = \frac{1}{L} \tag{2}$$

And for the INT sink node, it also update the fragment telemetry path information with probability $\Pr = 1/L$.

Therefore, LPV does not need to insert the complete path information in each packet as path lengthen, by uniform telemetry data sampling algorithm 1, an INT packet carries only fragment of the complete path information. On a transporting path of length L, the controller can aggregate fragments of path information with L INT packets, and verify path consistency of a flow.

The following algorithm 2 is the controller path verification algorithm in LPV. Comparing the fragment path of L telemetry items with the complete path information authorized, the controller verifies the path consistency of a flow.

Algorithm 2. Path validation in LPV.

Input: L *telemetry data* **and** $\{R_1, R_2, \cdots\cdots, R_L\}$

Output: True of False

```
1) for (k=1; k<=L; k++)
2)     if  ∃R_i, data[k] == R_i
3)         mark  R_i  // Next round of search will exclude  R_i
4)         continue;
5)     else
6)         return  FALSE;
7)     endif
8) endfor
9) return TRUE;
```

When the controller aggregates L fragments path information verified correctly of a flow, then the INT source node decreases the probability of inserting telemetry instruction.

$$\alpha = \omega \cdot \alpha,\ \omega \in [\frac{1}{2}, 1) \tag{3}$$

Otherwise, the INT source node enlarges insertion probability as following Eq (4).

$$\alpha = \frac{\alpha + 1}{2} \tag{4}$$

3.3. Network-level path validation

If flow-level path validation described in Section 3.2 is applied to all active flows in the network, then in the network the INT source or the transit switch continuously inserts or updates telemetry data and the controller continuously collects telemetry information and verifies the path consistency of flows. Of course, it will inevitably bring a large network communication and bandwidth overhead of data plane and increase load of the controller.

A packet of an active flow passes through multiple network nodes $\{R_1, R_2, \cdots\cdots, R_L\}$, and maybe there exists several different active flows pass through the same network node $R_i = (sw_i, < inport, egport >)$. The goal of this section is to select a sub-set of active flows in the network to achieve network-level path consistency validation.

The scheme of ATPG [10] discusses a similar issue, which searches for the minimum number of probing packets to validate the rules of all flows in the network. The issue is eventually deduced to a minimum set coverage problem, it is an NP problem, and formally, to find the minimum set coverage of flows is to solve the problem described in formula (5) [17].

$$\begin{aligned} \min \quad & \sum\nolimits_{flow \in \Psi} x_{flow} \\ s.t. \quad & \sum\nolimits_{R \in S_{flow}} x_{flow} \geq 1;\ \ \forall R \in \bigcup_{flow \in \Psi} S_{flow} \\ & x_{flow} = 0,1;\ \forall flow \in \Psi \end{aligned} \tag{5}$$

The notation $R=(sw,<inport,egport>)$ is the aforementioned network node，and S_{flow} is the set of switch nodes on the path authorized of a flow. To achieve network-level path consistency verification, we propose a heuristic flow select algorithm 3 shown as below which select the sub-set flows of active flows in the network.

Algorithm 3. Sub-set flows selection algorithm in LPV.

Input: Network flow set Ψ

Output: The selected sub-set flows F

1) $F \leftarrow \{\}$;

2) $\bigcup_{flow} \leftarrow \{\}$;

3) ***foreach*** *flow*, $flow \in \Psi$ ***do***

4) $\bigcup_{flow} = \bigcup_{flow} \cup S_{flow}$

5) ***while*** $\exists flow \in \Psi, s.t. S_{flow} \subseteq \bigcup_{flow}$, ***do***

6) ***select*** $flow \in \Psi, s.t.\ \max |S_{flow}|$

7) $F \leftarrow F \cup \{flow\}$

8) $\Psi \leftarrow \Psi \setminus \{flow\}$

9) $\bigcup_{flow} \leftarrow \bigcup_{flow} \setminus S_{flow}$

10) ***endwhile***

In algorithm 3, $\bigcup_{flow}$ is the set of switch nodes for all active flows in the network, $R_i = R_j$, if and only if $sw_i = sw_j$, $inport_i = inport_j$ and $egport_i = egport_j$. Lines 1) to 4) initialize the subset F selected flows and the set of switch nodes $\bigcup_{flow}$ respectively. Each iteration of the algorithm, it checks the set Ψ of current network flows，and determine whether there is a flow that the set of switch nodes S_{flow} exist in the set of $\bigcup_{flow}$ (Line 5)). If validation fails, the algorithm exits and outputs the final result of selected subset F. Otherwise, it selects a flow with the longest path length (that is $|S_{flow}|$ is the maximum). Synchronously, it updates the set F and Ψ. Line 9) updates the node set $\bigcup_{flow}$. When there are multiple flows passing through the same node $R=(sw,<inport,egport>)$, Only one of a flow which path length is maximum is selected and other flows are excluded.

4. Experiments and evaluation

This section constructs a simple simulation network environment to evaluate the effectiveness of LPV proposed by extending the switch behavior model (behavioral-model version 2, BMV2) and the programmable P4 switch.

4.1. Experiment setup

With 64-bit Ubuntu16.04 operating system, the experiment emulation platform is configured with Intel (R) Core (TM) i7-8550 CPU, 1.8 GHz, 8 GB of memory. Our experiments are performed on the Mininet, programmable P4 software switch, and controller components based on P4Runtime interface. In this paper, we extend the switch behavior model BMV2 using C++ to implement operation of message authentication code and uniform telemetry data sampling algorithm.

4.2. *Implementation*

According to the INT standard specifications, the packet bandwidth overhead introduced by INT includes telemetry instructions (also known as telemetry metadata header) and telemetry data (telemetry metadata) inserted by INT source node. The telemetry instruction length is 12 bytes. The instruction is actually a bitmap, where each bit represents a different type of network status or information about switch, such as switch identification, ingress and egress ports of packet of a flow, hops, etc. If a bit is set to 1, then a switch copies relative status of the node to the telemetry metadata data in the packet header space, and each field is 4 bytes. Figure 4 below shows the packet header format in LPV.

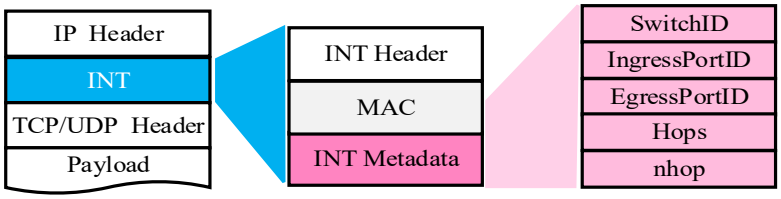

Figure 4. LPV Packet header format.

In LPV, the INT source node inserts the telemetry instructions and telemetry data with probability α based on the Table 3, INT transit nodes execute the telemetry instructions, and updates telemetry data based on the uniform telemetry data sampling Algorithm 1.

The INT metadata data in Figure 4 includes the switch ID, the value *Hops* (*Hops* is updated at each switch) of the packet transporting so far, the ingress and egress port of the switch, and the value *nhop* (only when telemetry data is updated according to sampling probability, *nhop* is updated) of an INT packet reaching a switch. According to Algorithm 1, if a switch *sw* determines to update the telemetry metadata (path fragment), the value *nhop* will be updated to *Hops*, otherwise, the item *nhop* will be kept unchanged. Based on uniform sampling, INT Metadata and MAC will be updated or be kept unchanged by the subsequent switch on the path and eventually be sent to the controller by the INT sink switch.

The switch *sw* calculates message authentication code MAC of the telemetry packet based on Eq (6), which prevent malicious message tampering, and MAC is 8 bytes. The controller verifies validity of each MAC, if all the aggregated MACs are valid, then the controller judges that the path is consistent.

$$MAC = Hash_{Ki}(IP_{INVAR} \parallel payload \parallel Header \parallel Metadata) \tag{6}$$

Where IP_{INVAR} is the invariable field in the IP header space, *payload* represents packet load, *Header* is the telemetry header, and Metadata is the telemetry data of fragment of complete path. The process diagram about forwarding packets for P4 switch in LPV is shown in Figure 5, which includes Input, Parse, Ingress, Egress, Output, etc. And more specific details about P4 switch, refer to the literature [6] please.

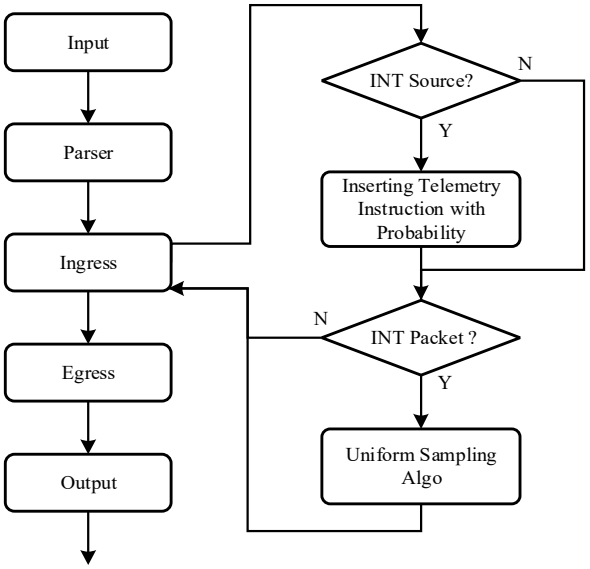

Figure 5. The process of packet forwarding in LPV.

4.3. *Performance evaluation*

This section evaluates the performance of the proposed mechanism, including packet header space overhead, network packet forwarding delay, network throughput, etc.

***Experiment* 1. Number of aggregated packets of path verification.** We performed experiments to test the expected INT packet count N required for path consistency verification for algorithm 1. The experimental result is depicted below in Figure 6.

Figure 6 shows average result of embedding telemetry instructions with the probability of $\alpha = 0.2$ and $\alpha = 0.5$, respectively. The result is in line with Theorem 1. When a forwarding path length is L, based on the uniform telemetry data sampling algorithm, with aggregated L INT packets, the controller can validate whether if a flow does follow forwarding path authorized really.

***Experiment* 2. Packet forwarding delay.** In LPV, the core network performance includes packet transmission latency and network throughput. The INT source switch inserts telemetry instructions with probability, the subsequent INT transit switch update the telemetry data based on uniform update sampling algorithm 1, all degrade the network performance.

In [13–15], the telemetry data inserted increase linearly as path lengthen, while LPV, basing on the proposed uniform sampling, all switches update the telemetry data with uniform probability, the INT packet header space size always keep constant. Table 4 lists the telemetry data bandwidth cost of a packet which embedded telemetry instruction in header space. Here, supposing that [13–15] need 16

bytes of size to collect a switch status at least, we can conclude that the header space overhead of LPV is less than [13–15] even if path length is a small value.

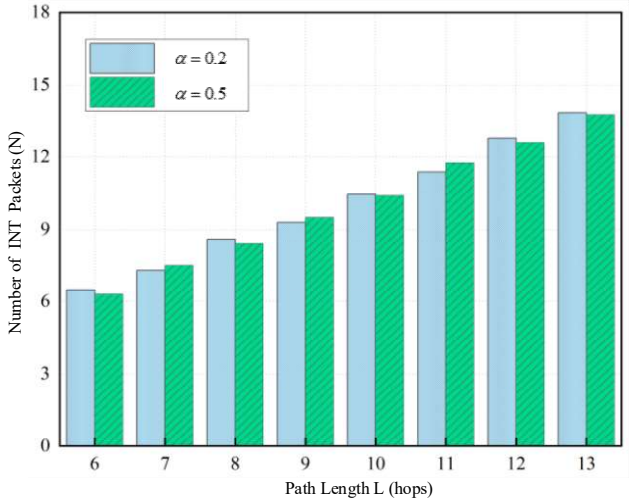

Figure 6. The expected INT packet count of path validation.

Furthermore, we define the header space communication overhead ratio γ between telemetry data inserted and the packet payload size, e.g., for the payload size of 900 bytes, $\gamma_{sINT} = (12+16L)/900$, $\gamma_{LPV} = 40/900$. As following Table 5, it shows that when the path length is 10 hops, the payload size varying from 300 to 1200 bytes, the header space communication overhead ratio γ of [13–15] varies from 57.3 to 14.3%, while only 13.3 and 3.33% communication overhead ratio γ of LPV.

Table 4. Typical scheme packet header space overhead (L: Path length).

Scheme	INT packet header space overhead (Bytes)
Wang [13]	12 + 16L
Sint [14]	12 + 16L
FS-INT [15]	12 + 16L
LPV	40

Table 5. The header space overhead ratio.

	The overhead ratio of different payload size (Path length L = 10)			
	300 B	600 B	900 B	1200 B
[13–15]	57.3%	28.6%	19.1 %	14.3 %
LPV	13.3%	6.66%	4.44%	3.33%

Figure 7 shows the results of round-trip delay (RTT) experiments tested at different path lengths L, here the *Baseline* is the original network without running the LPV protocol. We also evaluate the round-trip delay of LPV protocol inserting telemetry instructions with probability $\alpha = 0.2$ and $\alpha = 0.5$ respectively. From the Figure 7, we learn that when L = 10, the Baseline delay is about 26 ms. When inserting telemetry instructions with probability $\alpha = 0.2$, the average RTT is about 27.5 ms, while $\alpha = 0.5$, the average RTT is about 27.6 ms. So, LPV introduces about 7% additional forwarding delay.

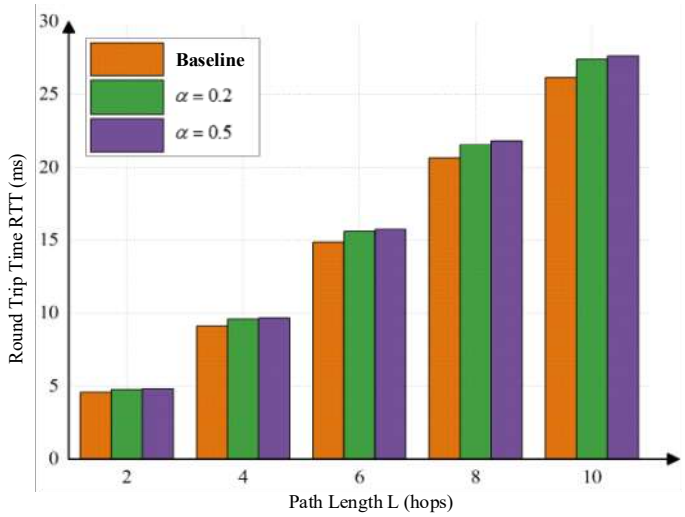

Figure 7. Round trip time.

***Experiment* 3. Network throughput.** We performed experiments for testing network throughput on the path of 10 hops length. Packet payload size varies from 300 to 1200 bytes, the INT source inserts INT instructions with probability $\alpha = 0.2$ and $\alpha = 0.5$ respectively. From Figure 8, we learn that when the payload is 900 bytes, the degradation of throughput is trivial of 6%, which is about 94–95% of Baseline.

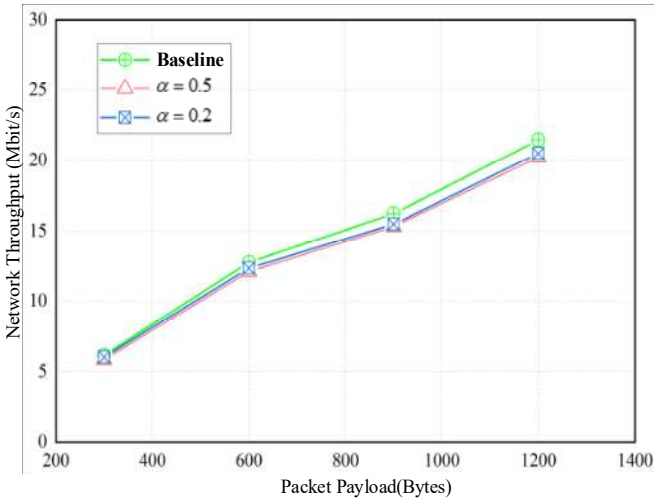

Figure 8. Throughput under different payload.

***Experiment* 4. The controller load.** We performed experiment testing controller load based on the heuristic flow selection algorithm when embedding telemetry instructions with different probabilities. Of the 50 different flows run simultaneously in the network, the experimental result is shown below in Figure 9. We can learn that embedding telemetry instructions with probability $\alpha = 0.2$ and $\alpha = 0.5$ respectively, when without executing the flow selection algorithm, the average controller CPU load is about 15–37%, and when performing the flow selection algorithm, the average controller CPU load is 10–24%.

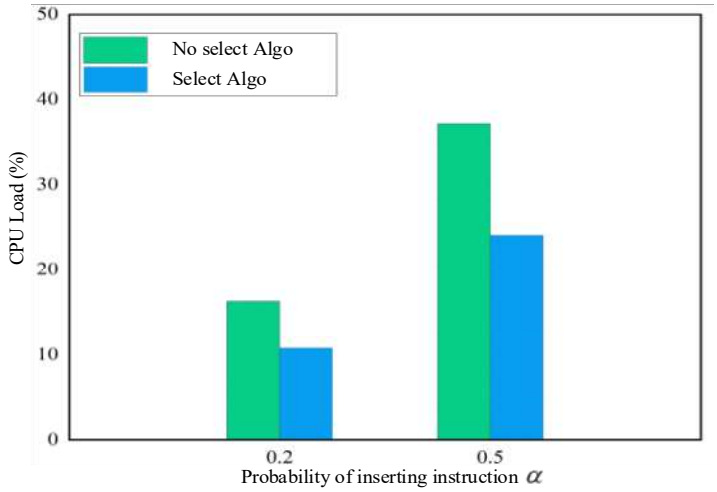

Figure 9. Average CPU load.

5. Conclusions

Existing path consistency verification mechanisms introduce significant communication overhead by active sending a large number of probing packets or embedding linear-scale tags in the header space of packets as path lengthen in software-defined networking. We propose LPV, a path consistency verification mechanism based on in-band telemetry. The INT source inserts telemetry instructions with probability, the transit switch updates telemetry data basing on the uniform sampling algorithm proposed. LPV fragments the complete path information into multiple INT packets which keep the packet header space size constant, and the controller aggregates path fragment to verify path consistency. We propose a heuristic flow selection algorithm to implement network-level path consistency validation. The analysis shows LPV effectively compress the INT packet header space size, the simulation experiment results demonstrate the effectiveness of the proposed mechanism and only introduce less than 7% of additional forwarding delays and 6% of throughput degradation at most.

Use of AI tools declaration

The authors declare that they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by Science and Technology Fund of the State Grid of China，Research on Key Technologies of Operation and Maintenance Security of Low-voltage Power Distribution Internet of Things (Grant No.5400-202255159A-1-1-ZN)

Conflict of interest

The authors declare there is no conflict of interest.

References

1. A. T. Mckeown, H. Balakrishna, OpenFlow: enabling innovation in campus networks, *ACM Comput. Commun. Rev.*, **38** (2008), 69–74. https://doi.org/10.1145/1355734.1355746
2. D. Singh, A. Shiv, S. K. Chamoli, Software Defined Networking (SDN) Challenges, issues and solution, *Int. Comput. Sci. Eng.*, **7** (2019), 884–889. https://doi.org/10.26438/ijcse/v7i1.884889
3. L. Tan, W. Su, Z. Zhang, J. Miao, N. Li, In-band network telemetry: A survey, *Comput. Networks*, **186** (2020). https://doi.org/10.1016/j.comnet.2020.107763
4. S. R. Chowdhury, R. Boutaba, J. Franois, LINT: Accuracy-adaptive and lightweight in-band network telemetry, in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, (2021), 349–357. https://ieeexplore.ieee.org/document/9464012
5. G. Simsek, D. Ergenç, E. Onur, Efficient network monitoring via in-band telemetry, in *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*, (2021), 1–6. https://doi.org/10.1109/DRCN51631.2021.9477344
6. P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, et al., P4: programming protocol-independent packet processors, *ACM Comput. Commun. Rev.*, **44** (2014), 87–95. https://doi.org/10.1145/2656877.2656890
7. P. Pereíni, M. Kuniar, D. Kosti, Monocle: dynamic, fine-grained data plane monitoring, in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, (2015), 1–13. https://doi.org/10.1145/2716281.2836117
8. Z. Peng, L. Hao, C. Hu, Mind the gap: Monitoring the control-data plane consistency in software defined networks, in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, (2016), 19–33. https://doi.org/10.1145/2999572.2999605
9. A. Shukla, S. Fathalli, T. Zinner, A. Hecker, S. Schmid, P4Consist: toward consistent P4 SDNs, *IEEE J. Sel. Areas Commun.*, **38** (2020), 1293–1307. https://doi.org/10.1109/JSAC.2020.2999653
10. H. Zeng, P. Kazemina, G. Varghese, Automatic test packet generation, *IEEE Trans. Networking*, **22** (2014), 554–566. https://doi.org/10.1109/TNET.2013.2253121
11. C. Hu, Z. Peng, C. Zhang, Fast testing network data plane with RuleChecker, in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, 2017. https://doi.org/10.1109/ICNP.2017.8117541
12. P. Manzanares-Lopez, J. P. Munoz-Gea, J. Malgosa-Sanahuja, Passive in-band network telemetry systems: the potential of programmable data plane on network-wide telemetry, *IEEE Access*, **9** (2021), 20391–20409. https://doi.org/10.1109/ACCESS.2021.3055462
13. S. Y. Wang, Y. R. Chen, J. Y. Li, A bandwidth-efficient int system for tracking the rules matched by the packets of a flow, in *2019 IEEE Global Communications Conference (GLOBECOM)*, (2019), 1–6. https://doi.org/10.1109/GLOBECOM38437.2019.9013581
14. Y. Kim, D. Suh, S. Pack, Selective in-band network telemetry for overhead reduction, in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, 2018. https://doi.org/10.1109/CloudNet.2018.8549351
15. D. Suh, S. Jang, S. Han, S. Pack, X. Wang, Flexible sampling-based in-band network telemetry in programmable data plane, *ICT Express*, **6** (2020), 62–65. https://doi.org/10.1016/j.icte.2019.08.005

16. F. Yang, W. Quan, N. Cheng, Z Xu, X. Zhang, D. Gao, Fast-INT: Light-weight and efficient in-band network telemetry in programmable data plane, in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), IEEE*, 2020. https://doi.org/10.1109/VTC2020-Fall49728.2020.9348823
17. P. Zhang, H. Wu, D. Zhang, Verifying rule enforcement in software defined networks with REV, *IEEE Trans. Networking*, **28** (2020), 917–929. https://doi.org/10.1109/TNET.2020.2977006

AIMS Press