INT-label: Lightweight In-band Network-Wide Telemetry via Distributed Labeling

Enge Song, Tian Pan, Haoyu Song, Qiang Fu, Yingjiang Liu, Chenhao Jia, Chuanying Yuan, Minglan Gao, Jiao Zhang, Tao Huang, Yunjie Liu

Abstract—In-band Network Telemetry (INT) enables hop-by-hop device-internal state exposure for maintaining and troubleshooting data center networks. To achieve *network-wide* telemetry coverage, orchestration on top of the INT primitive is required. A straightforward solution would flood the network with INT probe packets for maximum measurement coverage, which leads to a huge bandwidth overhead. A refined solution leverages the SDN controller to collect the network topology information and carry out centralized probing path planning, which, however, is inefficient in reacting to topology changes. To tackle the above problems, we propose *INT-label*, a lightweight In-band Network-Wide Telemetry architecture via the distributed labeling approach. INT-label periodically labels the sampled packets with device-internal states. It is cost-effective with a minor bandwidth overhead and able to seamlessly adapt to topology changes. In order to reduce the number of labeled packets, we introduce a times-based probabilistic labeling algorithm, which allows fewer packets to carry more INT information than the interval-based algorithm. In addition, to counteract the degradation of telemetry resolution due to loss of labeled packets, we design a feedback mechanism which can adaptively change the instant labeling frequency. We provide theoretical proof that INT-label can achieve network-wide telemetry. We analyze the impact of transmission delay on coverage rate and labeling times distribution under the INT-label architecture. Evaluation on software P4 switches suggests that INT-label can achieve 99.72% measurement coverage under the labeling frequency of 20 times per second. With the adaptive labeling enabled, even if 60% of the packets are lost, the coverage can still reach 92%.

Index Terms—In-band Network Telemetry, Distributed Labeling, P4, Adaptive Labeling, Network-Wide Coverage.

1 Introduction

Fine-grained, network-wide visibility is vital to maintaining and troubleshooting high-density, mega-scale data center networks which accommodate heterogeneous mission-critical applications [1, 2]. However, traditional network management protocols, such as SNMP [3], fall short of high-resolution monitoring of highly dynamic data center network traffic, due to the inefficient controller-driven, per-device polling mechanism. With full-mesh pings launched by end hosts, Pingmesh [4] is capable of providing the maximum end-to-end latency measurement coverage. However, it cannot extract hop-by-hop latency or look into the queue depth of switches for in-depth analysis. For network applications such as load balancing [5, 6], failure localization [7, 8], and management automation [9], such detailed information becomes increasingly crucial. In-band Network Telemetry (INT) [10], one of the killer applications of P4 [11], allows probe or user packets to query device-internal states, such as queue depth and queuing latency, when they pass through the devices. Considered promising for high-precision monitoring, it has been embedded into the latest merchant silicons for switches [12–14]. However, as a chip-level primitive, INT only defines the contract between incoming packets and device-internal states. For network-wide telemetry coverage, further orchestration on top of INT is needed.

There are generally two design patterns to realize network-wide telemetry coverage as shown in Fig. 1: sending probe packets [4, 5, 15, 16] and in-situ measurement [17–21]. HULA [5] follows the probe packet paradigm and employs the ToR switches to flood the probes into data center network's multi-rooted topology for high measurement coverage. Since the probe senders have no global view of the network for coordination, one link may be repetitively monitored by many probes with a large bandwidth overhead. For high-resolution monitoring, the bandwidth waste is even worse. To overcome this limitation, centralized probing relies on an SDN controller [22] to make optimized probing path planning. For example, INT-path [16] collects the network topology and generates non-overlapped probing paths to cover the entire network with the minimum path number using an Euler trail-based algorithm. INT-path is theoretically optimal but has deployment issues. First, the path planning result is tightly coupled with the topology. Any topology change in data center networks will require topology recollection and path recalculation in the controller, and probe generators/collectors (*i.e.*, path endpoints) reconfiguration at the data plane, interrupting the telemetry service. Besides, the probe packets require to use source routing [23] for path specification, bloating the probe packet header for long probing paths and needing

This work is supported by the National Natural Science Foundation of China (62372053). A preliminary version of this work was presented at the 40th International Conference on Computer Communications (INFOCOM'21), May, 2021.

E. Song, T. Pan, Y. Liu, C. Jia, C. Yuan, M. Gao, J. Zhang, T. Huang and Y. Liu are with Beijing University of Posts and Telecommunications, China (email: {songenge, pan, yingjiang, 564822241, ycy224, gml, jiaozhang, htao, liuyj}@bupt.edu.cn).

H. Song is with Futurewei Technologies, USA (email: hsong@futurewei.com).

Q. Fu is with Royal Melbourne Institute of Technology, Australia (email: qiang.fu@rmit.edu.au).

Corresponding author: Tian Pan (pan@bupt.edu.cn).

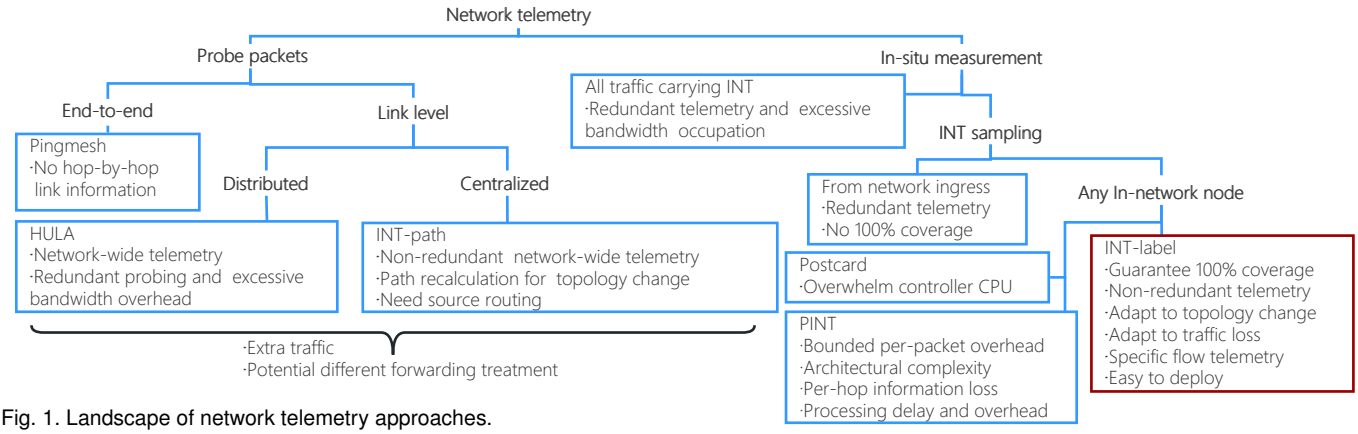


Fig. 1. Landscape of network telemetry approaches.

additional switch support of source routing-based forwarding. Such probe packets may be treated differently from the normal user traffic, affecting the measurement accuracy.

The in-situ measurement approaches adopt a "probeless" architecture [17, 18, 21]. They insert the INT header and data into user packets rather than probe packets. The INT sampling of [17, 18] is conducted at the network ingress. Specifically, the INT header is inserted into the sampled packets of specific flows at some ingress nodes, and each forwarding node writes INT data into the packets labeled with the INT header. The INT sample rate can be adjusted according to the traffic fluctuation of the specific flows. Labeling user packets at the flow level will result in redundant probing since different flows may share the same path and different paths may converge at some nodes. Besides, they cannot guarantee the network-wide telemetry coverage. Postcard [19, 20] and PINT [21] achieve INT sampling at any in-network node. In postcard-based approaches [19, 20], each device reports its internal states to the controller for every triggering packet it sees. The huge amount of postcards generated tend to overwhelm the southbound interface and the controller. PINT [21] aims to bound the per-packet overhead according to limits set by the user, for example, without exceeding the MTU (Maximum Transmission Unit). To do this, PINT has to approximate and spread the INT information over multiple packets, implicitly having the effect of sampling. Different types of INT items (queries) are probabilistically encoded onto different packets. This means the packets carrying the different pieces of the same INT information are scattered apart to be reassembled, leading to significant reporting delay. PINT uses a fixed-sized digest instead of the standard INT telemetry header to bound the per-packet overhead. To implement the fixed-sized digest, a combination of techniques have to be in place, including Global Hashes, Distributed Coding and Value Approximation. All these result in architectural complexity, per-hop information loss and processing delay and overhead. This is not suitable for network-wide telemetry that requires per-hop complete INT information and timely reporting without the reassembling delay imposed by PINT.

Another method to reduce the network-wide telemetry redundancy is to label packet only when the device-internal states change [17, 24]. A device can choose to label packets for drastic state changes to reduce the telemetry redundancy [24]. However, these methods may consume significant storage on data-plane devices. In addition, the labeling decisions on different devices are independent, making them impossible to monitor specific flows which is a necessary task in network maintenance. For example, the public cloud providers may provide services for flow-level information query [25]. Flow-level information can also help to deduce the flows or users affected by single-point equipment failure, allowing the cloud providers to infer the failure impact.

To tackle the above problems, we propose *INT-label*, a lightweight In-band Network-Wide Telemetry architecture. INT-label follows the "probeless" architecture. The INT-label-capable devices periodically label user packets with device-internal states rather than producing probe packets. Specifically, on each egress device port, the packets are sampled according to a predefined labeling interval T and labeled with the instant device-internal states. As a result, INT-label can achieve network-wide coverage with fine-grained telemetry resolution while introducing a minor bandwidth overhead. Along a forwarding path consisting of multiple devices, the same packet can be labeled independently according to the local sampling decision. This means that INT-label has no path-related dependency. Therefore, there is *no* need for centralized path planning on the SDN controller. In other words, INT-label is decoupled from the network topology and adaptive to arbitrary link failures. INT-label also relies on the data plane programmability, which is well supported by a growing number of merchant silicons such as Barefoot's Tofino [26] and Broadcom's Trident 3 [14]. In INT-label, the INT data carried by sampled packets are sent to the controller at the last-hop device for analysis, so the in-network labeling is *transparent* to end hosts. To avoid excessive INT packets, we design a *probabilistic labeling* algorithm which makes label decision for an incoming packet based on the number of already collected INT data in it, in order to aggregate INT data into fewer packets and reduce the number of INT export packets. Based on the labeling times, we further extend the algorithm to support specific flow telemetry, a challenging task under the distributed telemetry architecture. While both INT-label and PINT assume that per-packet INT data collection is not necessary, INT-label does not have per-hop INT information loss that PINT imposes. To counteract the telemetry resolution degradation due to loss of labeled packets on some congested links, we design a *feedback* mechanism to adapt the labeling frequency to the packet loss once the controller detects it by analyzing the telemetry data. Similarly, if poor

telemetry resolution is observed due to the large labeling interval, the labeling frequency can be adjusted accordingly. The controller monitors the telemetry resolution and adjusts the labeling frequency in the switch, to ensure that the switch only needs to maintain the least amount of state information. This is particularly useful, when INT-label operates in a highly dynamic network where the labeling interval may need to adjust from time to time.

INT-label is directly based on the standard INT and therefore only works when INT is supported. This makes it particularly suitable for data center networks, where programmable dataplane and SDN are expected to be deployed. Our main contributions are summarized as follows:

- We propose INT-label, an INT-based lightweight "probeless" telemetry architecture. We elaborate the packet encapsulation format and the corresponding packet processing procedure (§2).
- We design an interval-based distributed labeling strategy, allowing redundancy-free labeling to maximize the network coverage. We also propose a probabilistic labeling algorithm to suppress excessive INT packets and support specific flow telemetry. Furthermore, a feedback mechanism for labeling frequency adaptation is adopted to avoid telemetry resolution degradation due to loss of labeled packets (§3).
- We demonstrate that INT-label can theoretically achieve network-wide telemetry coverage. We analyze the impact of transmission delay on coverage rate, which can provide guidance for system parameter setting. Besides, we establish a mathematical model to analyze the distribution of the number of collected INT data in a packet under different scales of the FatTree topology (§4).
- We design an INT-label system with P4. We describe each module of the system and, in order to verify the impact of probability function on the probabilistic labeling algorithm (Base B), we add an interface to deploy various complex probability functions (§5).
- We implement an INT-label prototype using software P4 switches [27] and open source the project [28]. Extensive evaluation suggests that INT-label can achieve 99.72% measurement coverage under the labeling frequency of 20 times per second. The probabilistic labeling algorithm reduces the number of INT export packets by 35.2%. With the adaptive labeling enabled, even if 60% of the packets are lost, the measurement coverage can still reach 92%. Compared with HULA [5], INT-label consumes only 3% extra bandwidth (§6).

2 Telemetry System Architecture

Design overview. The basic idea of INT-label is that every network device conducts distributed *packet sampling* at each port, and writes the real-time device-internal states onto the sampled packets in the same way as the standard INT [29]. The packet labeling operation is triggered periodically at each port and the labeling decision is made locally without global synchronization. The distributed labeling guarantees network-wide telemetry coverage except for the idle device

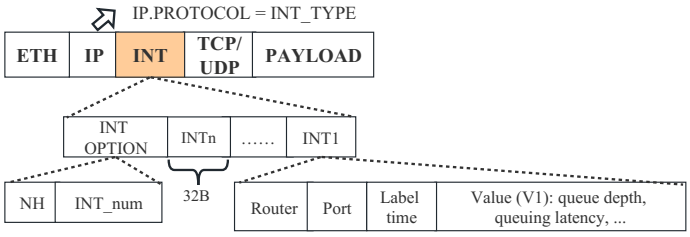


Fig. 2. INT-label packet encapsulation.

ports with no traffic. However, these idle ports are insignificant for network management and easy to speculate. For example, the latency is very low and there is no queue buildup. Generally, the network-wide telemetry resolution is expected to be as high as possible (*i.e.*, the INT data of each port should be collected as frequently as possible by the controller). However, this also leads to a tremendous overhead [30]. If each device writes INT data to each packet on each port, the controller will receive excessive redundant information, which will consume too much bandwidth and overwhelm the controller. For real deployment, a proper labeling frequency should be decided according to the given telemetry resolution requirement under the constraint of affordable system cost.

Packet encapsulation. Since the INT data are tagged to user traffic, the in-network labeling operation should (1) not interfere with the normal packet forwarding, and (2) be transparent to end hosts. As shown in Fig. 2, we place the INT header after the IP protocol header, because the packet labeling operation is conducted at each port and the IP header needs to be parsed and processed first to get a packet's egress port. This procedure follows the current INT specification [29]. Another option is to add the INT headers right after the TCP/UDP header. We fill the *PROTOCOL* field in the IP header with *INT_TYPE* when writing INT data to the packet for the first time. The first field of the INT header is *INT OPTION*. It contains the original value of the *PROTOCOL* field in the IP header, which we refer to as *Next Header* (*NH*), and the number of INT data items (*INT_num*). *NH* is used to restore the original packet at the last-hop device. A variable-length INT data stack is allocated after the *INT OPTION* field. In Fig. 2, each INT data item (INTn) occupies 32B, containing the information such as device ID (Router in INT1), ingress/egress port ID (Port in INT1), the timestamp of labeling (Label time in INT1), and latency and queue depth (Value/V1 in INT1). Both encapsulation and decapsulation of the INT header are conducted in the network and thus transparent to the end hosts. In the FatTree topology, a packet can reach its destination by at most 5 hops, so the variable part of the INT header is at most 160B (32B*5), a typical case in data centers. With longer paths (in terms of hops), the labeled packet may exceed the MTU and thus has to break into multiple packets through IP fragmentation. This does not affect how INT-label works.

Forwarding behaviors. In Fig. 3, the INT packet traverses the path with the red color from R1 to R6. The INT labeling operation is done at the egress port. Each port of the switch/router uses a special register (*Reg1*) to record the last INT labeling time LT. When a packet is ready to be forwarded, the network device determines whether the

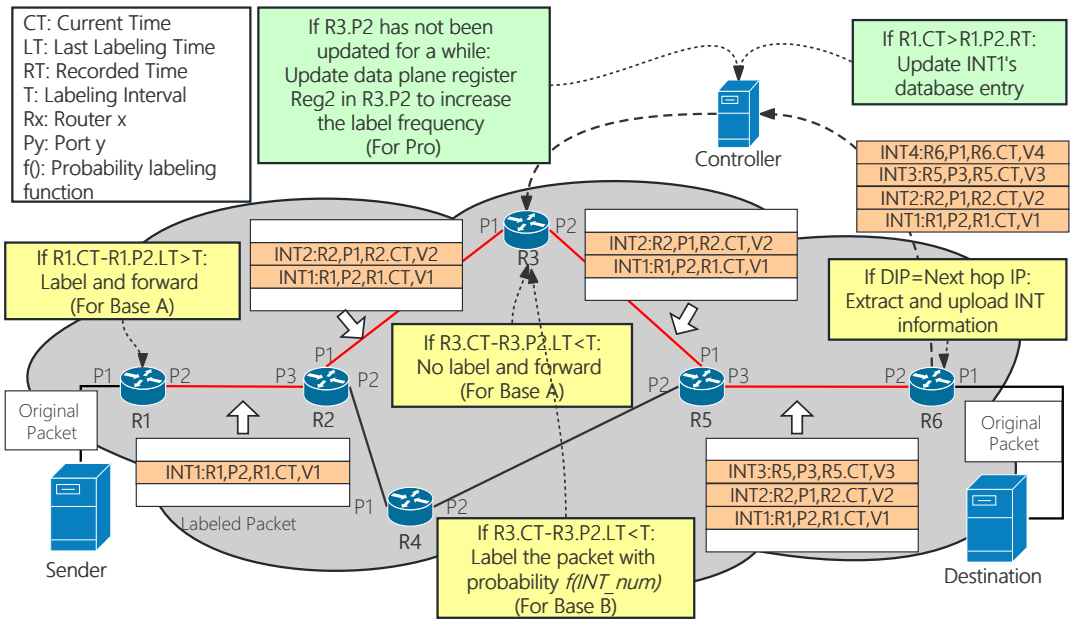


Fig. 3. INT-label architecture.

port has written INT data to any packet within a labeling interval T. If so, no action is taken to the current packet (*e.g.*, R3 in Fig. 3). Otherwise, the port labels the packet and updates *Reg1* with the current time *CT* (*e.g.*, R1, R2, R5 and R6 in Fig. 3). If the IP header's *PROTOCOL* field is not *INT_TYPE*, it also needs to be changed to *INT_TYPE* for INT header encapsulation. The original *PROTOCOL* field is copied to *INT OPTION* for later restoration. At the last hop, the INT header is extracted and the INT data are exported to the SDN controller. The original IP packet is restored with the *NH* field in *INT OPTION* and forwarded to its destination. For example, R6 in Fig. 3 restores and forwards the original packet to the destination, and uploads the collected INT information to the controller. Different from INT-path [16], INT-label does not have any path-related dependency. Therefore, a global view of the network topology and centralized path planning are unnecessary. INT-label will produce up-to-date information as long as each port on a device is probed within a certain period of time. If there is no traffic going through the port, the port is therefore not probed as we do not generate dedicated probe packets. We argue that if the port is not used, there is no urgency to probe the port.

Telemetry data collection. The controller uses a database to record each port's information, including the port states such as latency, queue depth, and the *timestamp of labeling*. Due to the potential end-to-end latency variation of different paths, even for the same port, packets labeled by a switch may not arrive in-order at the controller. Therefore, the controller needs to determine whether the database needs to be updated with the newly arrived INT data. If the timestamp of newly arrived data is greater than that already stored in the database, the database should be updated; otherwise, the newly arrived data are discarded. In this way, the most up-to-date network view is maintained in the database.

Reduction of labeled packets. The INT data collected from the data plane need to be exported to the controller for analysis. However, excessive INT export packets may overload the southbound interface and the controller. Our goal is to reduce the number of the labeled packets by aggregating labels into fewer packets using a probabilistic approach. We change the original interval-based labeling to *probabilistic labeling* according to labeling times received by packets. In our design, a packet which has already been labeled for several times will have a higher probability to be labeled again on its forwarding path. As a result, the INT data are more concentrated. By aggregating the INT data, the number of labeled packets is reduced without compromising the telemetry granularity. The number of INT export packets is reduced as well.

Telemetry resolution tuning. Packets may be lost on unreliable links. If a labeled packet is lost, the port labeling the lost packet does not know this and only believes that the INT data have been provided during the current period. Hence the controller database fails to be updated in a timely manner, causing telemetry resolution degradation. To resolve this issue, we adopt *adaptive labelling* instead. If the controller finds that a port has not been updated for a long time, it infers that there may be packet loss on the link from that port, so we should increase the frequency of packet labeling at this port to increase the number of labeled packets and offset the impact of packet loss. Similarly, if the labeling interval is too large, resulting in poor telemetry resolution, the labeling frequency can be adjusted accordingly. Each port of the switch/router has a special register (*Reg2*) to record the INT labeling frequency. In Fig. 3, the controller may update *Reg2* to modify the instant INT labeling frequency at a particular port. Note that increasing the labeling frequency does not increase the number of packets in the network.

3 Labeling Algorithms

We first propose a simple interval-based labeling algorithm (*Base A*) based on the interval between the current time and the last labeling time on the egress port, followed by a more sophisticated algorithm with probabilistic labeling based on labeling times (*Base B*). At last, we introduce the adaptive labeling algorithm (Pro).

Algorithm 1: Interval-based labeling (Base A)

```
   Input: Labeling interval T, each arriving packet
   Output: (Labeled or unlabeled) outgoing packet
1  Function LabelPacket(packet):
2      if IP.PROTOCOL ≠ INT_TYPE then
3          Write original IP.PROTOCOL to NH and modify the
             IP.PROTOCOL to INT_TYPE
4          Set INT_num in the INT OPTION to 0
5      Write INT data and increase INT_num by 1
6      Write R_x.CT to the register Reg1 of the port R_x.P_y
7  Function BaseA(T, packet):
8      Look up the routing table to get the forwarding port
         R_x.P_y
9      Get last labeling time R_x.P_y.LT from register Reg1 and
         current time R_x.CT of router R_x
10     if R_x.CT − R_x.P_y.LT > T then
11         LabelPacket(packet)
12     else
13         NoAction()
14     if DIP = next hop IP and IP.PROTOCOL = INT_TYPE then
15         Extract and upload INT data to the controller
16         Modify the IP.PROTOCOL with the NH
17         Recalculate the checksum
18     Forward packet
```

3.1 Interval-based and probabilistic labeling

Interval-based sampling is a common practice to reduce overhead. For example, it has been used by NetFlow [31]. In this section, we design labeling algorithms by leveraging sampling in different ways.

Interval-based labeling (Base A). Algo. 1 details the packet labeling and decision making algorithm, which is invoked for each arriving packet. For packet labeling, if the packet has never been labeled, we allocate the INT header and change the *IP.PROTOCOL*. The original *PROTOCOL* field is recorded (line 2-4). The INT data is then written to the packet header with *INT_num* incremented by 1. In addition, the last labeling time (*LT*) in *Reg1* is updated with the current time *CT* (line 5-6). The *Base A* algorithm leverages the difference of *CT* and *LT* to determine if labeling is required. The labeling operation is done only when (1) there is an arriving packet, and (2) the labeling interval *T* has elapsed (line 10-13). Note that line 12-13, while taking no action in *Base A*, can be replaced with different code to realize the *Base B* algorithm. Next, Destination IP (DIP) is compared with the next hop IP to determine whether the device is at the last hop (line 14). If it is true for a labeled packet, the INT data are extracted and sent to the controller while the original packet is forwarded to the destination (line 15-18).

Algorithm 2: Labeling times-based probabilistic labeling (Base B)

```
   Input: Labeling interval T, each arriving packet
   Output: (Labeled or unlabeled) outgoing packet
1  Function BaseB(T, packet):
2      .....
3      else
4          Read INT_num from the INT OPTION
5          p = f(INT_num)
6          if random(0, 1) < p then
7              LabelPacket(packet)
8      .....
```

Probabilistic labeling (Base B). Algo. 2 describes the probabilistic labeling algorithm based on labeling times. The basic idea of *Base B* is to label each packet with a probability p, which is determined by the number of already collected INT data items in the packet (*INT_num*). The larger the *INT_num*, the higher the probability p. *Base B* differs from *Base A* in that even when $R_x.CT - R_x.P_y.LT \leq T$, the device may also label the packet. Due to space limitation, Algo. 2 only shows the difference where line 13 of Algo. 1 is replaced with line 4-7 of Algo. 2. Specifically, if $CT - LT > T$, the packet is labeled (line 10-11 in Algo. 1); otherwise, the device reads *INT_num* from the *INT OPTION* (line 4 in Algo. 2), calculates the labeling probability p according to *INT_num* (*i.e.*, $p = f(INT\_num)$), and randomly labels the packet with the probability p (line 5-7).

$f(\cdot)$ is an increasing function satisfying some specific design requirement. A simple function is $f(INT\_num) = \frac{INT\_num}{MAX\_hop-1}$. When $INT\_num = 0$, $p = 0$; when $INT\_num = MAX\_hop - 1$, $p = 100\%$. In this way, the packets carrying fewer INT data items are still mainly labeled according to the interval, but the packets carrying more INT data items have a higher probability to be labeled. Compared with *Base A*, *Base B* increases the total labeling times in the data plane to some extent, and aggregates INT data items in fewer labeled packets. This will greatly reduce the number of labeled packets and thus the processing time of these packets. The increase of labeling times also updates the device states faster.

Algorithm 3: Adaptive labeling (Pro)

```
   Input: INT database, labeling interval T, parameter k
   Output: Updated value S in Reg2
1  Initialize: Last time Reg2 is updated (i.e., LRT) = current time
2  Function Pro(INT database, T, k):
3      while True do
4          if current time − LRT > k ∗ T then
5              Get recorded time R_x.P_y.RT according to the key
                 (R_x, P_y) from the INT database.
6              if current time − R_x.P_y.RT > k ∗ T then
7                  Reg2 = Reg2 + 1
8                  LRT = current time
9              else
10                 if Reg2 > 0 then
11                     Reg2 = Reg2 − 1
12                 LRT = current time
```

3.2 Controller-driven adaptive labeling (Pro)

We leverage the adaptive labeling (Algo. 3) to counter the telemetry resolution degradation due to the loss of labeled packets or the large labeling interval. For example, if this resolution degradation is observed, the controller may reduce the labeling interval and thus improve telemetry resolution. In the worst-case scenario, INT-label may fall back to the standard INT labeling every single packet. Based on the *Base A/B* algorithm, the Pro algorithm adds additional modules to both controller and data plane for dynamic adjustment of the labeling interval T which is fixed in the *Base A/B* algorithm. We add a register *Reg2* to store a variable S at each port of a device. S is initialized to be 0 and can be updated by the controller. The value of S

indicates that the labeling interval T should be reduced by 2^S times. That is, the labeling interval of *Base A/B* is changed from T to $T/2^S$, which can be easily implemented with the right shift operation. By analyzing the network-wide device states, the controller can determine if the label frequency of some ports need to be changed. For example, if the recorded time of a port has not been updated for a long time (*i.e.*, $CT - RT > k * T$), the controller will increase S in *Reg2* to raise the label frequency (line 6-7 in Algo. 3). Otherwise, the controller will decrease S if $S > 0$ (line 10-12). In order to not change S too fast, we record the last timestamp as *LRT* when *Reg2* is updated, and only modify S after a period of time lapsed since *LRT* (line 4).

The recall of the INT tasks may suffer if the labeling interval is too large. This is particularly true for highly dynamic networks. With adaptive labeling, the labeling interval can be adjusted according to the network dynamics. If the recall is getting unacceptable, the labeling frequency needs to be increased, resulting in the increased overhead.

3.3 Specific flow telemetry

The distributed labeling process is difficult to collect telemetry data for a specific flow on its entire path, which is needed by some applications (*e.g.*, public cloud providers need the hop-by-hop latency data of specific users for QoE evaluation), and often requires dedicated probe packets [5, 16]. Fortunately, under the same distributed labeling algorithm framework, we can tune the *Base B* algorithm a little bit to make it support specific flow telemetry and seamlessly work with the normal labeling process. To achieve this, in the *Base B* algorithm, we need to ensure each switch's labeling probability to be 100% for any specific flow we want to monitor. Specifically, we use the same labeling probability function $f(INT\_num) = \frac{INT\_num}{MAX\_hop-1}$, but for a specific flow packet, when it is sampled and labeled at the first device, we initial its *INT_num* to *MAX_hop*. Then the labeling probability will be 100% along the packet's forwarding path, and thus the INT data will be written to the packet header with *INT_num* incremented by 1 at each hop. Such specific flows are easy to be identified at the last hop because only their *INT_num* are greater than *MAX_hop*. When extracting the INT data at the last hop, we get the real number of INT information by subtracting *MAX_hop* from *INT_num* (line 3-4 in Algo. 4). After that, the INT data can be extracted and exported (line 7).

Specific flow telemetry is integrated into the same code implementing the *Base B* algorithm. The switches on path follow the same processing procedure without any extra overhead. The data collected for specific flows also contribute to the network-wide telemetry coverage. The administrator specifies a flow by adding a flow table entry to the ingress device and sets a sampling frequency for it.

3.4 Telemetry result collection

Algo. 4 demonstrates the telemetry data collection algorithm. These steps are shared by both *Base A, B*, and *Pro*. At the controller, each INT export packet may contain multiple INT data items collected along the route, and data item is regarded as a separate entry to be installed into the INT database (*i.e.*, one telemetry data packet exported from the data plane may trigger a string of database insertions). The INT database is indexed by the unique key {device ID, port ID} with value filled with the telemetry data as well as the timestamp for the labeling time (we call the timestamp "recorded time" (RT)). The timestamp ensures that the INT database contains only the latest telemetry data (line 16-17 in Algo. 4).

Algorithm 4: Telemetry data collection

```
   Input: Packets with INT information
   Output: Updated INT database at the controller
1  Function ExtractINT(packet):
2      Read INT_num in the INT OPTION
3      if INT_num > MAX_hop then
4          n = INT_num − MAX_hop
5      else
6          n = INT_num
7      Extract n * 32B from packet
8  Function ResultCollection(packet):
9      while Receive a packet with INT information do
10         INT_info = ExtractINT(packet)
11         for R_x, P_y, R_x.CT, Latency in INT_info do
12             if the key (R_x, P_y) not in the INT database then
13                 Add database entry (R_x, P_y) : R_x.CT,
                       Latency into the INT database
14             else
15                 Get recorded time R_x.P_y.RT (i.e., previously
                       inserted R_x.P_y.CT) according to the key.
16                 if R_x.CT > R_x.P_y.RT then
17                     Update the INT database with R_x.CT
                           and Latency (i.e., the latest INT
                           information)
```

Case in point. Fig. 3 shows the labeling-based INT data collection and export process, as well as adaptive labeling. First, a host (Sender in the figure) sends a packet with the destination of another host. When forwarding the packet, the routers decide whether to label it with INT data according to the algorithm of Base A (line 10 in Algo. 1) or Base B (line 6 in Algo. 2). The routers also check if the next hop is the destination address. If so, they extract the INT data from the packet and restore the original packet (line 14-16), in order to make the INT process transparent to end hosts. When receiving the INT data exported by R6, the controller parses the data items (as described in line 10 of Algo. 4). The newer records are stored in the database via timestamp comparison (line 15-16). If a port has not been updated for a long time, the controller can modify *Reg2* of the port to increase its labeling frequency (line 6-7 in Algo. 3).

4 Theoretical Analysis

4.1 Proof of Correctness

First, we demonstrate that the *Base A* algorithm can achieve both network-wide coverage. The network-wide coverage means that all active links with traffic are monitored in a given time window.

Proposition: Base A can probe the device states of all the ports with traffic within the interval $T + T_s$, where T is the labeling interval, and T_s is the maximum packet arrival interval.

Proof. We construct a simple proof by contradiction. Assuming that Base A cannot achieve network-wide coverage within $T + T_s$, that is to say, the time interval between

TABLE 1
Symbols used in analysis.

Symbols	meaning
T	Labeling interval
T_s	Maximum packet interval on a port
T_t	Transmission latency
T_u	INT data export time
T_p	Time for first packet after T
T_i	Telemetry interval($\frac{1}{telemetry\ resolution}$)
P_t	Probability of probing any port more than once within T_i

the two labeling operations can be greater than $T + T_s$, indicating that some port $R_x.P_y$ does not write INT data to any packet during the time period $[t, t + T + T_s]$. This also means $t > R_x.P_y.LT$. On the other hand, there must be a packet forwarded to other network devices through this port within the time period $[t, t + T + T_s]$, since the maximum packet arrival interval is T_s. Now consider a packet with the arrival time of $t+T+T_p$, where $0 < T_p \leq T_s$. $\because t > R_x.P_y.LT\ and\ T_p > 0$, $\therefore t + T + T_p - R_x.P_y.LT > t + T + T_p - t = T + T_p > T$. According to the line 10 of Algo. 1, the port needs to write INT data to this packet at its arrival time $t + T + T_p$. Since $0 < T_p \leq T_s$, the port writes INT data to the packet during the time period $[t, t + T + T_s]$, contradicting the assumption. Hence, the assumption is incorrect and Base A can achieve network-wide coverage within the interval $T + T_s$.

In other words, we can complete probing all the ports with traffic at time $t + T + T_s$ from any given time t. The value of T_s is roughly inversely proportional to the rate of traffic passing through the port. So, when the traffic rate becomes higher, the time required to probe all the ports with traffic will approach T. Conversely, if the traffic rate is low, T_s will increase, suggesting that the port states cannot be monitored as frequently as expected. However, since the traffic rate is usually high in a data center network, T_s is very small and thus $T + T_s$ is close to T.

4.2 Impact of delay on coverage rate

We have known that, from the last labeling, a port would label another packet at a time between T and $T + T_s$. However, these labeled packets will take some time to reach the last-hop router, so the controller always receives the delayed data. We would like to know when the controller can complete a network-wide telemetry data update. The $coverage\ rate$ represents the ratio of active ports with at least one update to all the active ports at the controller over a period of time. Transmission delay in the network may lead to a suboptimal coverage rate. For example, if a data packet carrying multiple INT data items has experienced a long queuing delay on a congested link, its data cannot be exported to the controller in a timely manner. Consequently, the controller fails to obtain timely data of those affected ports. We establish a mathematical model to analyze the transmission delay's impact on the coverage rate. Table. 1 summarizes the variables used in the analysis.

Transmission delay fluctuates due to various factors such as the congestion level on a link and the number of traversed links on a path. We regard the delay as a random variable T_t and assume it follows a normal distribution with a mean of μ and a variance of σ^2. That is, $T_t \sim N(\mu, \sigma^2)$.

Taking the penultimate hop R5 in Figure 3 as an example, we analyze the time interval between two labeled packets. At time t, $R5$ writes INT data to a data packet. The transmission delay to the next hop is denoted as T_{t1}. So the packet reaches the last-hop router R6 at time $t+T_{t1}$. Assume the time required for the last-hop switch to export the INT data to the controller is a constant value T_u. The controller receives this INT data at time $t+T_{t1}+T_u$. Since the R5's port does not label other data packets within $labeling\ interval\ T$, the next time it will label another data packet will be after $t+T$. Assume this labeled packet leaves the port at $t+T+T_p$. The value of T_p is inversely proportional to the packet rate of this port. This value is smaller if the packet rate is high, as high packet rate implies an earlier labeling opportunity. Once the labeling conditions are met, the router will label the packet with INT data. Assume the transmission delay of this packet is T_{t2}. The packet reaches the last-hop router R6 at time $t + T + T_p + T_{t2}$ and finally reaches the controller at time $t + T + T_p + T_{t2} + T_u$. Assume the controller updates the database immediately after receiving new data. The time interval between two updates to the same port is therefore $(t+T+T_p+T_{t2}+T_u)-(t+T_{t1}+T_u) = T+T_p+T_{t2}-T_{t1}$.

If the controller expects telemetry data updates for all devices/ports in every time interval T_i (*i.e.*, network-wide telemetry coverage), it is essential to maintain $T+T_p+T_{t2}-T_{t1} \leq T_i$, and $\therefore T_{t1} - T_{t2} \geq T + T_p - T_i$. As a result, the probability that any port must be probed within interval T_i, denoted as P_t, is equal to $P(T_{t1} - T_{t2} \geq T + T_p - T_i)$.

In probability theory, if X and Y are normally distributed independent random variables, their sum is also normally distributed [32]. That is, if $X \sim N(\mu_1, \sigma_1^2)$, $Y \sim N(\mu_2, \sigma_2^2)$ and $Z = X + Y$, then $Z \sim N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Similarly, $Z = X - Y \sim N(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2)$. $\because T_{t1},\ T_{t2} \sim N(\mu, \sigma^2)$. Since T_{t1} and T_{t2} are two independent delay observations, $\therefore Q = T_{t1} - T_{t2} \sim N(\mu - \mu, \sigma^2 + \sigma^2)$. i.e., $Q \sim N(0, 2\sigma^2)$. In probability theory, if $X \sim N(\mu, \sigma^2)$, $P(X \geq a) = P(\frac{X-\mu}{\sigma} \geq \frac{a-\mu}{\sigma}) = 1 - \Phi(\frac{a-\mu}{\sigma})$ [32]. $\because Q \sim N(0, 2\sigma^2)$. $\therefore P(Q \geq T + T_p - T_i) = P(\frac{Q}{\sqrt{2}\sigma} \geq \frac{T+T_p-T_i}{\sqrt{2}\sigma}) = 1 - \Phi(\frac{T+T_p-T_i}{\sqrt{2}\sigma})$.

Although the labeling interval is T, experiments show that it often fails to reach 100% coverage within time interval T because when $T_i = T$, $T + T_p - T_i = T_p > 0$. $\therefore (\frac{T+T_p-T_i}{\sqrt{2}\sigma}) > 0$. $\because \Phi(x)$ is a monotonically increasing function. $\therefore \Phi(\frac{T+T_p-T_i}{\sqrt{2}\sigma}) > \Phi(0) = 0.5$. $\therefore P(Q > T + T_p - T_i) = 1 - \Phi(\frac{T+T_p-T_i}{\sqrt{2}\sigma}) < 1 - \Phi(0) = 0.5$. It indicates that even if telemetry data reach the controller in only one hop, the possibility of getting high coverage rate is less than 50% due to delay fluctuation. Usually multiple hops are required for telemetry data to reach the controller, implying a greater impact on coverage rate due to transmission delay. In general, if a port's data arrive at the controller by n hops, the probability of two export data packets for that port reaching the controller within an interval of T is no more than 0.5^n.

It has been theoretically proved that, the probability P_t that INT-label can achieve network-wide telemetry coverage under the condition of labeling interval = T and telemetry interval = T_i is equal to $1 - \Phi(\frac{T+T_p-T_i}{\sqrt{2}\sigma})$. When $\Phi(\frac{T+T_p-T_i}{\sqrt{2}\sigma})$

is smaller, higher coverage rate can be obtained. $\because \Phi(x)$ is a monotonically increasing function. $\therefore \frac{T+T_p-T_i}{\sqrt{2}\sigma}$ need to be small. It is difficult to control σ because it is subject to link states and traffic conditions. Since the traffic rate influences T_p and the traffic is also affected by the number of data export packets, T_p is volatile. Hence, we can make $\frac{T-T_i}{\sqrt{2}\sigma}$, or $T - T_i$, smaller to get a higher P_t. $\therefore T_i$ needs to be greater than T, suggesting that we should make $T < T_i$ for high coverage rate when determining the telemetry interval T_i. For instance, if we want a telemetry resolution of 100 times per second, the telemetry interval $T_i = 10ms$ and T should be set less than 10ms. The smaller the value of T, the higher the coverage rate is. However, a smaller T will result in more consumption of data plane and southbound interface bandwidth due to more labeled data packets. We provide the guidance for the choice of T through experiments.

4.3 INT labeling times distribution

We analyze the distribution of the number of INT data items carried by the packets of *Base A* and *Base B* under different scales of FatTree topology. The analysis is divided into three parts: (1) the relationship between the number of hops required for a packet to reach the destination and the size of the FatTree, (2) the labeling times distribution of *Base A*, and (3) the labeling times distribution of *Base B*.

End-to-end hop count vs. FatTree size. We analyze the number of hops required for a packet to reach the destination under a k-ary FatTree topology as introduced in [33, 34]. k is the pod number, each pod contains $k/2$ aggregation and edge switches, and each edge switch is directly connected with $k/2$ hosts. The total number of hosts is $k^3/4$. We assume that the *host1* located at <*pod1, edge1*> sends a packet to each other host. Since there are $k^3/4$ hosts in total, host1 sends $k^3/4 - 1$ packets. In the FatTree topology, the end-to-end hop number of a packet is only related to the positional relationship between the sender and the receiver. To ease the description, we assume that the location of the receiver is <*podR, edgeR*>. Let H be the number of hops required for a packet to reach its destination. We analyze the end-to-end hop number case by case.

1. If *edgeR* $=$ *edge1*, $H = 1$. Each edge connects $k/2$ hosts, so there are $k/2 - 1$ packets with an H value of 1.

2. If *edgeR* $\neq$ *edge1* and *podR* $=$ *pod1*, $H = 3$. A pod contains $k/2$ edge switches, and each edge switch connects $k/2$ hosts, so there are $(k/2 - 1) * k/2$ packets with an H value of 3.

3. If *podR* $\neq$ *pod1*, $H = 5$. There are k pods in the k-ary FatTree network, and each pod contains $(k/2)^2$ hosts, so there are $(k - 1) * (k/2)^2$ packets with an H value of 5.

So H is a discrete random variable whose value can be 1, 3, and 5. The probability distribution of H can be calculated as: $P(H = 1) = \left(\frac{k}{2} - 1\right) / \left(\frac{k^3}{4} - 1\right)$, $P(H = 3) = \left(\frac{k}{2} - 1\right) * \frac{k}{2} / \left(\frac{k^3}{4} - 1\right)$, $P(H = 5) = (k - 1) * \left(\frac{k}{2}\right)^2 / \left(\frac{k^3}{4} - 1\right)$.

Labeling times distribution of Base A. In the Base A solution, each switch determines if a packet needs to be labeled independently without considering whether or not the packet has already be labeled by some other switches. Therefore, the packet labeling actions by different switches are independent and unrelated events. Let X_i represent whether the packet is labeled after passing through the i-th switch. X_i is a discrete random variable with a value of 0 or 1 (representing unlabeled or labeled), so X_i is a typical *Bernoulli trial*. Assume that the maximum interval of packets passing through each port is T_s and the labeling interval is T. There will be a packet being labeled among T/T_s packets, so the probability of a packet being labeled is $p = 1/(\frac{T}{T_s}) = \frac{T_s}{T}$. Therefore, $X_1, X_2, ..., X_n$ all obey the *Bernoulli trial* with p being $\frac{T_s}{T}$.

A theorem in probability theory claims that, in a Bernoulli experiment with n trials, if the probability of success on each trial is p, and let A count the number of successes in n trials, then the probability of event $A = s$ is: $P(A = s) = C_n^s p^s (1 - p)^{n-s}, s = 0, 1, 2, \ldots, n$.

In the Base A solution, let Y_n count the number of labeling times after n hops. $\because Y_n \sim B(n, \frac{T_s}{T})$. $\therefore P(Y_n = s) = C_n^s \left(\frac{T_s}{T}\right)^s \left(1 - \frac{T_s}{T}\right)^{n-s}, s = 0, 1, 2, \ldots, n$.

Let Z_A represent the labeling times of a packet sent by host1 in the Base A solution, so the probability distribution of Z_A is: $P(Z_A = s) = \sum_{l=1,3,5} P(H = l) * P(Y_l = s), s = 0, 1, \ldots, 5$. Plugging the above relations into the expression of $P(Z_A = s)$, we get the probability distribution of labeling times in the Base A solution. Due to the space limitation, we do not expand it in this paper but present the results in the GitHub repository [28].

Labeling times distribution of Base B. In the Base B solution, each switch labels packets further according to the number of INT data items carried in the packets. The larger the number, the greater the labeling probability is. It is different from Base A in that the labeling decision on a subsequent switch will be affected by the labeling actions of previous switches. Therefore, each labeling action can no longer be regarded as an independent and unrelated event. We need to analyze the correlation between the two labeling actions.

In Base B, even if $CT - LT \leq T$, the switch may label the packet and refresh the last labeling time (*LT*), which reduces the probability of labeling based on the interval. We denote the probability based on the interval and the probability based on the INT data item as p_B and $f(INT\_num)$, respectively. The larger $f(INT\_num)$ is, the smaller p_B is. For example, when $f(INT\_num) = 1$, $p_B = 0$. The switch will only label packets based on the number of INT data items. Conversely, the smaller $f(INT\_num)$ is, the larger p_B is. When $f(INT\_num) = 0$, $p_B = \frac{T_s}{T}$. It can be seen that p_B is a function of $f(INT\_num)$, denoted as $p_B = g(f(INT\_num))$. According to the above analysis, $g(x)$ is a monotonically decreasing function as $g(x) \in \left[0, \frac{T_s}{T}\right], x \in [0, 1]$.

For convenience, we denote N and T as INT data number-based and interval-based labeling, respectively. $P(N) = f(INT\_num)$, $P(T) = g(f(INT\_num))$. Next, we analyze the probability of joint events case by case.

1. The packet is labeled based on the INT data item number, denoted as $N\bar{T}$. According to the *total probability rule*, $P(N) = P(T)P(N \mid T) + P(\bar{T})P(N \mid \bar{T})$. Because N and T cannot happen at the same time, $P(N \mid T) = 0$. Therefore $P(N) = P(\bar{T})P(N \mid \bar{T})$. According to the *formula for conditional probability* $P(B \mid A) = P(AB)/P(A)$, $P(N \mid \bar{T}) = P(N\bar{T})/P(\bar{T})$. Therefore, $P(N\bar{T}) = P(\bar{T})P(N \mid \bar{T}) = P(N) = f(INT\_num)$.

2. The packet is labeled based on the interval, denoted as $\bar{N}T$. Similar to the case 1, we get the probability: $P(\bar{N}T) = P(T) = g\left(f\left(INT\_num\right)\right)$.

3. The packet is not labeled, denoted as $\bar{N}\bar{T}$. Because N and T cannot happen at the same time, $P(NT) = 0$. Therefore, $P(\bar{N}\bar{T}) = 1 - P(NT) - P(\bar{N}T) - P(N\bar{T}) = 1 - f\left(INT\_num\right) - g\left(f\left(INT\_num\right)\right)$.

In the Base B solution, let W_n count the number of labeling times after n hops. Obviously, W_n is a discrete random variable with values of $0, 1, 2, \ldots, n$. It is easy to see that $P\left(W_{n+1} = i\right) = P\left(W_n = i\right) * (1 - g(f(i)) - f(i)) + P\left(W_n = i - 1\right) * (f(i-1) + g(f(i-1)))$, $i = 0, 1, \ldots, n+1$. The first term represents that when $W_n = i$, the packet is still not labeled by the $n+1$th hop switch, so $W_{n+1} = i$. The second term represents that when $W_n = i - 1$, the packet is labeled by the $n+1$th hop switch, so $W_{n+1} = i$. The boundary terms involved in this formula are all 0 (*e.g.*, $P(W_n = -1) = 0, P(W_n = n+1) = 0$). Then we analyze the probability distribution of W_1. Since the packet before entering the first-hop switch will not carry any INT data, the packet labeling at the first-hop switch is determined only by the interval, so the labeling probability is the same as that of Base A, which is $\frac{T_s}{T}$. Therefore, $P\left(W_1 = 0\right) = 1 - \frac{T_s}{T}$ and $P\left(W_1 = 1\right) = \frac{T_s}{T}$. According to the expression of $P(W_{n+1})$, we derive the probability distribution of W_2, and then the probability distribution of $W_3, W_4, \ldots$.

Similar to the analysis for Base A, we only need the distribution of W_1, W_3, and W_5. Z_B represents the labeling times of a packet sent by host1 in the Base B solution, so the probability distribution of Z_B is: $P\left(Z_B = s\right) = \sum_{l=1,3,5} P(H = l) * P\left(W_l = s\right), s = 0, 1, \ldots, 5$. By plugging the above relations into the expression of $P\left(Z_B = s\right)$, we get the probability distribution of label times of Base B. Again, the results are presented in the GitHub repository [28].

5 System Implementation

Network testbed. Mininet [35] is used to establish the virtual network environment. We use the simple switch model of BMv2 (a software-based P4 switch) [27] to realize the virtual switch so that a customized INT header format can be supported. The simple software switch BMv2 is good enough for our design verification despite its performance and functionality difference with an actual hardware P4 switch. Besides, we implement a remote controller through a separate OS process that receives notifications from the data plane and communicates with the database through socket.

INT header. Programmable data plane devices allow arbitrary header format. For example, Protocol Oblivious Forwarding (POF) [36] uses programmable $\{offset, length\}$ to identify packet header fields. According to the definition of packet format in §2, we need to add a new header *INT OPTION* to support packet labeling. As shown in Fig. 4, we define a new header *INT OPTION* in P4 language. The 8-bit *int_num* can support the packet to carry 2^8 INT data items in the *Base A* algorithm. In the *Base B* algorithm, we need to reserve half of 2^8 (*i.e.*, 2^7) to support the specific flow telemetry (line 3-4 of Algo. 4), while the maximum hop in real networks is far less than this value. The length of the *PROTOCOL* field in the IP header is 8 bits, so we reserve 8 bits for the *NH* field, which store the value of the *PROTOCOL* field for packet recovery at the last hop.

```
header int_option_t {
    bit<8>   NH;
    bit<8>   int_num;
}

control MyEgress {
    apply(get_dst_ip)
    apply(int_label_logic)
    if (next_hop_ip == dst_ip){
        apply(extract_int)
        apply(restore_data_packet)
    }
}
```

Fig. 4. INT header option format and the egress control block.

Algorithm 5: The P4 implementation of int_label_logic.

```
1  Function int_label_logic():
2      if hdr.int_option.isValid() then
3          set_flag_table.apply();
4          if meta.int_flag_md.int_flag==1 then
5              int_table.apply();
6              lt.write((bit<32>)port, ct);
7          else
8              bit<48> T = 50000;
9              bit<48> T_1 = T * 0/100;
10             T_1_value.write(1, (bit<48>) T_1);
11             bit<48> MAX_hop = 5;
12             lt.read(meta.lt, (bit<32>)port);
13             if ct - meta.lt >T then
14                 int_table_sampling.apply();
15                 lt.write((bit<32>)port, ct);
16             else if ct − meta.lt > T_1 then
17                 bit<8> int_num_val =
                     meta.int_num_md.int_num;
18                 bit<48> rand_val;
19                 random(rand_val,0,100);
20                 if rand_val < int_num * 100/MAX_hop
                     then
21                     int_table_sampling2.apply();
22                     lt.write((bit<32>)port, ct);
```

Egress control block of INT-label. INT data are added to the packet header at the egress port, which is depicted in Fig. 3. First, in order to forward the packet, we need to get the next hop to the destination IP address. Then, the *int_label_logic* needs to be executed, which is the most distinctive block in the INT-label. The *int_label_logic* block includes the interval-based labeling algorithm (*Base A*), the labeling times-based labeling algorithm (*Base B*) and the data-plane part of the adaptive labeling algorithm (*Pro*). Next, if the next hop IP address is the same as the destination IP address, execute *extract_int* for INT data extraction and *restore_data_packet* for original data packet restoration. Other conventional packet processing actions are omitted.

INT-label logic with P4. The Algo. 5 depicts the P4 implementation of *int_label_logic*. We simplify some variable names involved in the specific implementation of the algorithm (e.g., $standard\_metadata.egress\_port$ is simplified as $port$). The details can be found in the GitHub repository [28].

To simplify the implementation of the adaptive labeling

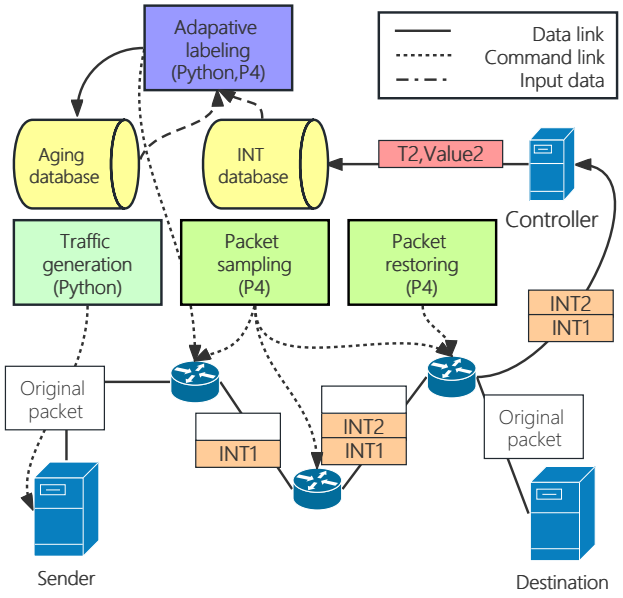


Fig. 5. INT-label modules and implementation.

algorithm (Algo. 3), we store a Boolean type $flag$ rather than an integer in the register *Reg2*. $flag$ is initialized to 0. If the value of $flag$ is 1, all the packets passing that port will be labeled (as implemented in line 4-5 in Algo. 5). Then, the last labeling time in the *Reg1* will be updated to the current time (line 6).

If the value of $flag$ is 0, we need to read the last labeling time (LT) in *Reg1* (line 12). If the interval between the current time and LT exceeds the labeling interval (T), the packet will be labeled and *Reg1* will be updated (line 13-15). Otherwise, we will execute the *Base B* labeling. In order to verify the impact of probability function $f(\cdot)$ on the *Base B* algorithm, we add optional items to the implementation of *Base B*. Specifically, we add a new parameter T_1 to specify the "rest" time after labeling a packet, which can prevent excessive labeling times due to the labeling times-based labeling. The value of T_1 is in the interval $[0, T]$ (line 9). If $CT - LT < T_1$, the packet will not be labeled. If $CT - LT \geq T_1$, the labeling interval-based probability labeling algorithm is executed by generating random numbers (lines 16-22).

Traffic generation. We write scripts for traffic generation and collection using Python's Scapy library. The route of the background traffic is specified by the source routing [23], and the background traffic rate is dominated by the sleep function at the sender (by pausing the sending for a while).

Adaptive labeling. The implementation of data-plane part of the adaptive labeling algorithm (*Pro*) has been discussed. Now we explain the control-plane part of the *Pro* algorithm. As mentioned above, the value of $Reg2$ affects the labeling frequency. The controller can directly modify the value of the register. As shown in Fig. 5, there are two databases in the controller: *INT database* storing the latest INT data collected from the data plane, and *aging database* used for adaptive labeling. If the controller finds one entry in *INT database* has not been updated as expected, it will set $Reg2$ of the corresponding port to 1 and insert an entry into the aging database for conducting *time-out* operation. The time-out function is built-in and implemented with the "subscribe-publish" mechanism in Redis [37]. Each entry in *aging database* will be deleted after $k * T$ and the process will publish a notification to another process which previously subscribed to the time-out notification. It indicates that the entry in INT database has been updated. If the time-out is triggered, controller will modify the value of $Reg2$ to 0. While the port in *INT database* is not updated even if the controller has set $Reg2$ to 1, the controller will again write the corresponding entry into aging database to refresh the time-out process to maintain $Reg2 = 1$. With the "subscribe-publish" mechanism, the value of $Reg2$ can be adjusted adaptively according to the data plane telemetry results.

6 Evaluation

6.1 Experiment Setup

We build an emulation-based network prototype to evaluate INT-label's performance. The prototype runs on a server with Ubuntu 18.04 OS and an Intel Xeon Silver 4116 CPU with 12 cores/24 threads and 32GB memory. The prototype is based on Mininet [35] and consists of 1 controller, 4 Spine switches, 4 Leaf switches, 4 ToR switches, and 8 servers, organized into the same FatTree topology as HULA [5]. Among the 8 servers, server1 and server8 randomly send traffic (packet size = 1kB) to others. The default traffic rate is around 5Mbps. With pre-installed flow tables, traffic passes through 24 ports (some ports are deliberately bypassed without traffic). The default labeling interval is 50ms. The maximum INT header length is 160B (32B*5). The default network-wide telemetry resolution is 10 times/s (the telemetry resolution is limited by BMv2's performance [27]). The default $f(\cdot)$ is $f(0) = 0, f(i) = 1, i = 1, \ldots, 4$. The prototype is implemented with 829 lines of Python and 317 lines of P4, available at the GitHub repository [28].

6.2 Results

Impact of the background traffic rate. Fig. 6 shows the impact of the background traffic rate on the network-wide coverage rate and the bandwidth occupation of INT encapsulation overhead. The *coverage rate* is defined as the labeled port number divided by the total number of ports with traffic passing by during every 100ms (the default telemetry resolution). *bandwidth occupation* $= \frac{bandwidth\ of\ INT}{bandwidth\ of\ traffic}$. In §4, we have demonstrated that our architecture can achieve network-wide probing if the background traffic rate is much larger than the labeling frequency. In other words, the coverage rate can be affected by the packet arriving rate of each port. As shown in Fig. 6, the coverage rate increases with the background traffic rate, which confirms our analysis. Furthermore, as the background traffic rate increases, the bandwidth occupation rate of the INT encapsulation with a constant labeling frequency decreases. The reason lies in that, as the traffic rate of the port increases, the average INT data items carried by each packet gradually decreases. When the background traffic rate reaches a certain scale, the coverage rate no longer increases significantly, achieving a turning point. Besides, the coverage rate and the bandwidth occupation of *Base B* are always higher than those of *Base A* due to the possibility of redundant labeling. We set the default background traffic rate as the rate at the turning point in Fig. 6.

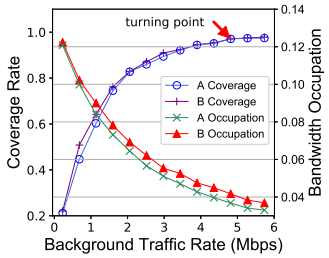


Fig. 6. The impact of labeling interval on coverage rate and bandwidth occupation.

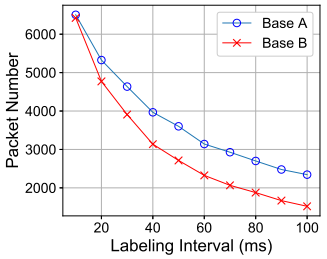


Fig. 7. The number of packets labeled with INT data under different labeling intervals.

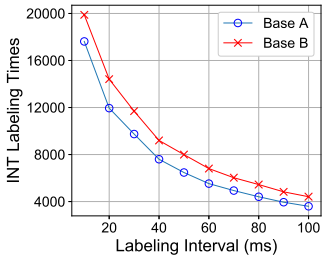


Fig. 8. The data plane labeling times under different labeling intervals.

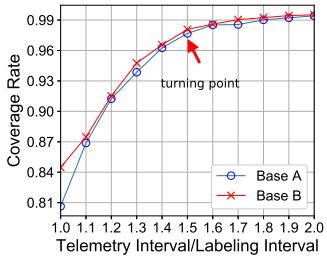


Fig. 9. How the relation between labeling interval and telemetry resolution affects the coverage rate.

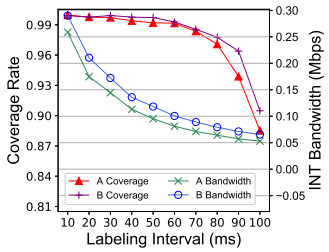


Fig. 10. The impact of labeling interval on coverage rate and INT bandwidth occupation.

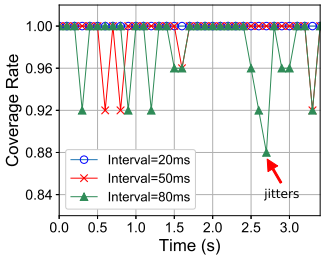


Fig. 11. The impact of data plane labeling interval on network-wide coverage rate changes over time.

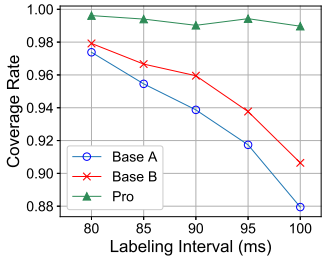


Fig. 12. Different coverage rates under Base A/B and Pro strategies.

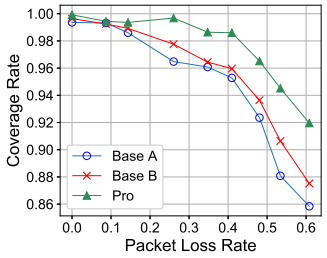


Fig. 13. Network-wide coverage degradation due to loss of packets under Base A/B and Pro strategies.

The number of packets carrying INT data: Base A vs. Base B. Fig. 7 shows the number of packets with INT data under different labeling intervals, with a total of 10,000 packets generated. It can be seen from the figure that as the labeling interval increases, the number of labeled packets of *Base A* decreases and the number of labeled packets of *Base B* is always lower than that of *Base A*. Specifically, the number of labeled packets of *Base B* decreases much faster than that of *Base A*. This is because the increase of T allows more probabilistic labeling on packets based on the number of INT data items, resulting in a more concentrated distribution of INT data, which in turn leads to a more obvious reduction in the number of labeled packets. When T = 100ms, compared with *Base A*, the number of labeled packets of *Base B* is reduced by 35.2%.

The number of labeling times: Base A vs. Base B. Fig. 8 shows the total labeling times in the data plane under different labeling intervals, with a total of 10,000 packets generated. As shown in the figure, *Base B* only causes a small number of redundant labels, which is insignificant compared with the reduction in the number of labeled/exported packets.

Right labeling interval for a given telemetry resolution. For network management, a network operator may need to determine an appropriate labeling frequency for cost-effective network-wide telemetry under a certain telemetry resolution. Overly-low labeling frequency can cause incomplete or untimely network-wide view, while overly-high labeling frequency is unnecessary and inefficient. Fig. 9 shows how to choose the appropriate labeling frequency according to the telemetry resolution requirement. For example, regardless of using *Base A* or Base B, the network-wide coverage rate in Fig. 9 has exceeded 0.98 when the ratio of telemetry interval/labeling interval is 1.6, which helps determine the right labeling interval if the required telemetry resolution (*i.e.*, 1/telemetry interval) is provided. Given the default telemetry resolution of 10 times/s, our system can achieve 99.72% measurement coverage under a labeling frequency of 20 times/s (telemetry interval/labeling interval = 2).

Impact of labeling interval on coverage rate and bandwidth overhead. Fig. 10 demonstrates the impact of labeling interval on coverage rate and bandwidth overhead. We set the telemetry interval to 100ms and make the background traffic rate fast enough. As shown in Fig. 10, the coverage rate increases as the labeling interval decreases. While a shorter labeling interval corresponds to a higher coverage rate, the bandwidth occupation also becomes greater. The labeling interval of 60ms is the turning point of the coverage curve. The same conclusion can be drawn from 100ms/60ms = 1.6 that telemetry interval/labeling interval = 1.6 can achieve a good trade-off between coverage rate and bandwidth overhead. In addition, compared to *Base A*, *Base B* always has a higher coverage rate and bandwidth occupation due to its redundant labeling property.

Impact of the data plane labeling interval on coverage rate changes over time. Fig. 11 shows the impact of labeling interval on network-wide coverage rate changes over time for *Base A* algorithm. As shown in Fig. 11, when T = 20ms, the coverage always remains at 100%, which means that a shorter labeling interval has a more stable and higher coverage. We find that a longer labeling interval causes more jitters on the network-wide coverage rate, which is triggered by the labeled packets queuing on congested links. Although the labeling intervals in Fig. 11 are all below the default telemetry interval (*i.e.*, 100ms), the labeled packets with longer labeling interval are more likely to miss the packet collection deadline of the latest telemetry round.

Performance of *Base A/B* and Pro under different labeling intervals. Fig. 12 shows the relationship between the coverage of the *Base A/B* and Pro algorithms and the labeling interval when labeling interval and telemetry in-

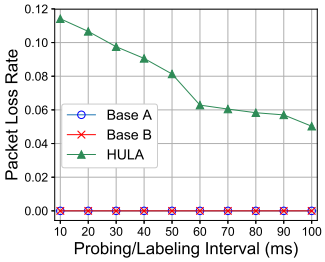


Fig. 14. Packet loss rate (with rate limit) at different labeling/probing intervals (Base A/B vs. HULA).

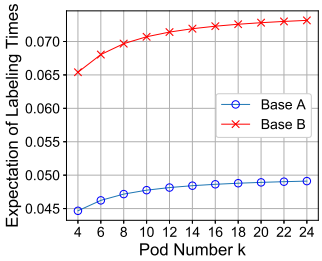


Fig. 15. The impact of FatTree topology size on theoretical labeling times of *Base A/B*.

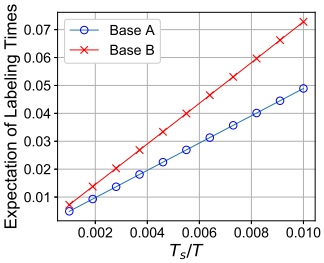


Fig. 16. The impact of T_s/T on theoretical labeling times of *Base A/B*.

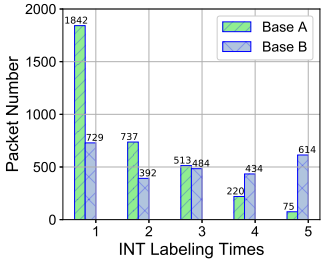


Fig. 17. Distribution of INT labeling times in Base A/B.

terval are closer. As shown in Fig. 12, the coverage of the Pro algorithm remains around 99%, while the coverage of *Base A/B* algorithm gradually decreases with the increase of labeling interval, because the Pro algorithm can automatically increase the labeling frequency when coverage decreases. This indicates that the Pro algorithm can always achieve a higher coverage, while the coverage of the Base A/B algorithm is affected by the relationship between the labeling interval and the telemetry interval.

Network coverage rate degradation due to traffic loss. Fig. 13 shows coverage rate degradation due to traffic loss. We deliberately change the background traffic rate, and limit the forwarding rate and queue depth of BMv2 to cause packet loss. Fig. 13 suggests that network coverage rate will degrade due to packet loss since the labeled packets will get lost as well. Specifically, the Pro algorithm outperforms the Base A/B algorithm thanks to the built-in adaptive labeling mechanism. With adaptive labeling enabled, the coverage rate can still reach 92% even if 60% of the packets are lost.

Packet loss due to rate limit. Fig. 14 compares the packet loss rate of Base A/B and HULA due to the deliberate rate limit. We limit the bandwidth and queue depth of BMv2 so that there is no packet loss while sending background traffic alone. It can be seen that there is no packet loss of Base A/B because of its ultra-low bandwidth occupation. In contrast, HULA induces packet loss, which decreases from 12% to 5% as the probing interval increases from 10ms to 100ms.

Distribution of INT labeling times. In §4.3, we analyze the labeling times distribution of the *Base A* and *Base B* algorithms. According to the derived formula, we can see that the labeling times are affected by the FatTree topology size (k) and the values of T_s/T (T is the labeling interval and T_s is packet arrival interval). Specifically, the topology scale affects the average end-to-end hop number and the values of T_s/T affects the labeling times of a single packet. In order to intuitively observe their influence on the theoretical labeling times, we obtain a set of numerical solutions by replacing $\{k, \frac{T_s}{T}\}$ with a set of data. Fig. 15 shows the impact of FatTree topology size on the mathematical expectation of labeling times. The value of T_s/T is $1/100$ and $f(i) = i/5, i = 0, 1, 2, 3, 4$. With the increase of topology scale, the number of hosts that do not belong to the same pod increases sharply, so the average end-to-end hop count increases (as analysis in §4.3). Because the labeling times of a single packet of *Base A/B* is not affected by the topology scale, the increase of average end-to-end hop count will lead to the increase of the mathematical expectation of labeling times. As a result, the mathematical expectation of labeling times increases with the increase of topology scale as shown in Fig. 15. Furthermore, due to the redundant labeling, *Base B*'s mathematical expectation of labeling times is always higher than that of *Base A*.

Fig. 16 shows the impact of T_s/T on the mathematical expectation of labeling times. The value of k is 20 and $f(i) = i/5, i = 0, 1, 2, 3, 4$. As shown in Fig. 16, the mathematical expectation of labeling times increases with the increase of T_s/T. The value of T_s/T represents the number of packets arriving within a single labeling interval. In the *Base A* algorithm, T_s/T is also the probability of a packet being labeled on a single device. When T_s/T is small, the more packets arrive within a labeling interval, the lower the probability of a packet being labeled in *Base A*. With the increase of the labeling probability on a single device, the mathematical expectation of labeling times of *Base A* increases. In the *Base B* algorithm, the probability of packet being labeled on the first hop is only affected by the time interval because $INT\_num = 0$. With the decrease of T_s/T, the probability of packet being labeled on the first hop decreases, which will lead to the decrease of the labeling probability of subsequent devices, so the mathematical expectation of labeling times of *Base B* decreases. Similarly, due to the redundant labeling, the mathematical expectation of labeling times of *Base B* is higher than that of *Base A*.

Next, we evaluate the real distribution of labeling times of *Base A/B*. Fig. 17 shows the number of packets under different labeling times, with a total of 10,000 packets. It can be seen that the number of packets with labeling times of 1 in *Base A* is the largest. And the number of packets gradually decreases with the increase of labeling times. The number of packets whose labeling times is 5 is very small. Although the number of packets with labeling times of 1 in *Base B* is also the largest, the number of packets with labeling times of 4 and 5 is large as well. The number of packets whose labeling times is 5 is second only to those whose labeling times is 1. In addition, the packets with labeling times 1 and 2 in *Base B* are much less than those in Base A, and the packets with labeling times 4 and 5 are much more. In summary, the *Base B* algorithm does make INT data more concentrated in fewer packets, which coincides with the conclusion in §4.

The impact of "rest" time. In order to explore the impact of probability function $f(\cdot)$ on the *Base B* algorithm, we added the concept of "rest" time in the implementation (§5). During the "rest" time, no matter how many INT data items the packet carries, the probability of packet labeling is 0. The "rest" time can reduce the redundant telemetry. In this experiment, the labeling interval T is fixed to 50ms

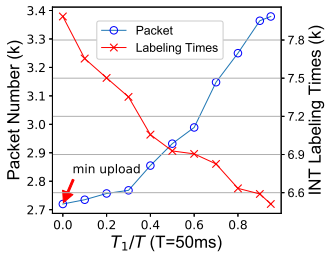


Fig. 18. The impact of "rest" time.

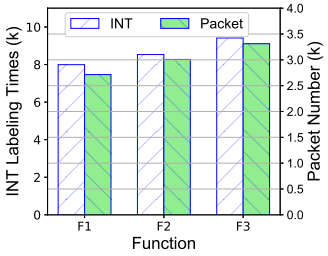


Fig. 19. The impact of probability function.

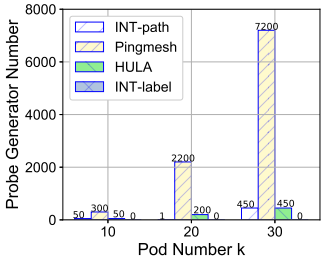


Fig. 20. The number of probe generators required under different scale FatTree topologies.

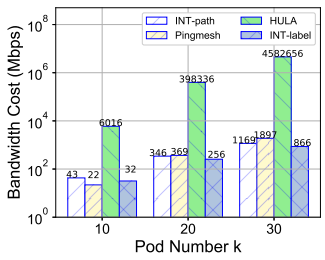
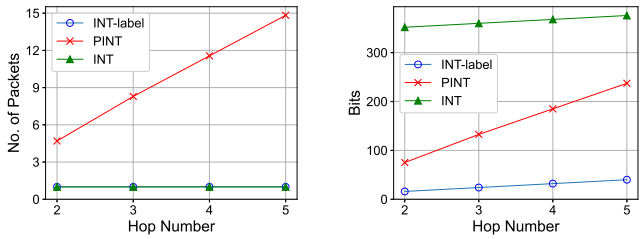


Fig. 21. The bandwidth overhead for network-wide telemetry under different scale FatTree topologies.

and the total packet number is 10,000. Fig. 18 shows the impact of T_1/T on the number of packets with INT data and total INT labeling times. As T_1 gradually approaches T, the total labeling times decreases. Because a lot of packets with INT data are forwarded during the "rest" time when the operation of labeling is blocked, which will inevitably reduce the total labeling times. However, as the total labeling times decreases, the number of packets carrying INT data increases, which shows that the per-packet labeling times decreases. The more the total labeling times, the higher the data-plane telemetry overhead and resolution. The smaller the number of packets carrying INT data, the lower the CPU overhead of controller. It can be seen from this figure that the "rest" time has a great impact on the performance of the *Base B* algorithm. If we want to get the least CPU overhead, we need to set T_1 to 0 (*i.e.*, no "rest" time). For a trade-off between the overhead of data plane and control plane, we can make $T_1 = \frac{T}{2}$ (*i.e.*, the cross point in Fig. 18).

The impact of probability function of the Base B algorithm. In order to further observe the impact of probability function of the *Base B* algorithm, we deployed several different probability functions, as shown in Fig. 19. F1 represents the probability function we currently use (*i.e.*, as long as the packets carry INT data, the labeling probability is 100%. $F1(0) = 0, F1(i) = 1, i = 1, \ldots, 4$). F2 function is a linear function, which can be expressed as: $F2(i) = \frac{INT\_num}{MAX\_hop-1}$. F3 function considers both time interval and INT_num, which is expressed as $F3(i) = \frac{INT\_num}{MAX\_hop-1} * \frac{interval}{T}$. As shown in the figure, the probability functions affect the INT labeling times and the number of exported packets. Because there is some redundancy in the *Base B* algorithm, which consumes a little extra data-plane transmission bandwidth. In addition, exporting INT data also consumes the processing resources of the controller. Therefore, the less the redundant telemetry data and the number of exported packets, the better. Fig. 19 shows that the performance of F1 function is the best, F2 is the second, and F3 is the worst. Because F3 considers the factor of time interval, which reduces the influence of INT_num, resulting in a more random labeling process. Although the F1 function introduces a large number of labeling based on the INT_num, these labeling will also refresh the last labeling time (LT) of the port, which reduces the number of labeling based on time interval, so the total number of labeling does not increase sharply. Therefore, we choose F1 function as the default probability function of the *Base B* algorithm.

The number of probe generators required. Fig. 20 shows the number of probe generators required by different



(a) INT packets (b) Extra bits

Fig. 22. The average number of INT packets and extra bits needed to probe the data path for a specific flow with INT-label/PINT/INT.

methods under different FatTree scales. INT-label does not need to allocate any probe generator due to its "probeless" mechanism, so the number is 0 for any topology. The number of probe generators of INT-path [16] is positively correlated with the number of odd vertices in the topology. And the odd vertices in the FatTree depends on its scale. When there is no odd vertex, only one probe generator is needed to send and receive probes. When there are $2k$ odd vertices, $2k$ probe generators are needed. The number of Pingmesh is positively correlated with the sum of the number of hosts and ToR switches. The number of HULA is positively correlated with the number of ToR switches. So the number of Pingmesh is the most, and the number of INT-path and HULA are lower. Therefore, INT-label has many advantages in deployment as it does not require any probe generator.

Network-wide telemetry bandwidth overhead. Fig. 21 shows the bandwidth cost comparison under different FatTree scales. Because INT-label only needs to adds a small INT header to some packets, it consumes little bandwidth. HULA has a large number of redundant probes due to its broadcast mechanism, so its bandwidth cost is always much higher than the other methods. Pingmesh's performance is better when the topology size is smaller, but gradually deteriorates as the topology scale increases. It should be noted that Pingmesh is incapable of measuring hop-by-hop link latency. Although INT-path realizes redundancy-free telemetry, it needs to generate extra probe packets, resulting in more bandwidth cost than INT-label by $\sim 35\%$. It can be concluded that INT-label can achieve network-wide telemetry coverage with the minimal bandwidth overhead.

Path tracing for a specific flow. To better showcase the effectiveness of INT-label, we have conducted comparisons with PINT [21] and the traditional sampling-based INT (*e.g.*, Sel-INT [18]). The experiments were conducted using the source code of PINT and the simple topology structure

provided within the code, which consists of only a single path between the source and destination. We varied the number of hops between the source and the destination, and measured the number of packets and extra bits needed to complete the path probing for a specific flow. The maximum number of bits that can be carried by a single packet in PINT was set to 16. For the sampling-based INT [18], each device will label INT information on the packet with a specific header, which is sampled at the ingress of the topology.

Fig. 22(a) illustrates the variation in the number of packets as the number of hops changes. Both INT and INT-label require only a single packet to complete the path probing task. However, PINT requires an increasing number of packets as the number of hops increases. Fig. 22(b) depicts the extra bits needed by path probing with INT-label, PINT and INT, respectively. Both INT and INT-label collect the INT information with a single packet. However, INT uses an active probing approach by sending probe packets, which introduces additional overhead in terms of packet headers. PINT introduces lots of redundancy as each packet is independently labeled, resulting in increased overhead that escalates with the number of hops. Based on the aforementioned points, INT-label can accomplish path probing with the minimum number of required INT packets and extra bits.

7 Related Work

Network telemetry provides deep visibility into the intricate networks, making it easier to maintain and troubleshoot the widespread distributed systems [6, 38, 39]. However, traditional management protocols, such as SNMP [3], fail to achieve fine-grained monitoring due to its inefficient controller-driven device state polling, which cannot adapt to the traffic dynamics in data center networks. Other approaches such as NetFlow [31], sFlow [40], and IPFIX [41] maintain per-flow counters instead of monitoring the low-level packet queuing behaviors. The protocol-independent forwarding architectures [11, 36] unleash the power of data plane programmability, enabling network devices to customize their functions. In-band Network Telemetry (INT) [10, 29] takes advantage of this capability, allowing probe/user packets to query device-internal states as they pass through the device data plane.

In some cases, INT is activated to monitor specific user flows by labeling some or all packets of these flows [21, 42], while in some other cases, network-wide telemetry coverage is desired for network-wide traffic load balancing [5] or failure detection/localization [4, 7]. For example, PINT [21] is designed to bound the per-packet overhead according to limits set by the user. PINT approximates and spreads the INT information over multiple data packets, and assumes that it is not required to know all of the per-packet-per-hop information that INT collects. PINT uses a fixed-sized digest instead of the standard INT telemetry header. The per-packet overhead can be as little as 1 bit. A combination of techniques have to be in place to make it work, including Global Hashes, Distributed Coding and Value Approximation. This results in the architectural complexity, the loss of some INT information and the significant processing delay and overhead such as reassembling the INT information. While PINT works well for applications that tolerate some loss of INT information with per-packet overhead constraints, it is not suitable for network-wide telemetry that INT-label aims to address. As INT is essentially an underlying primitive for device-internal state exposure, network-wide telemetry coverage requires high-level orchestration on top of INT [16]. Some network telemetry systems [4, 5, 16] collect telemetry results by sending probe packets, but the probe packets lead to extra traffic load and potentially different forwarding treatment. For example, INT-path [16] is based on active probing. It generates a set of non-overlapped probing paths covering the entire network and then sends probe packets over these paths through source routing. INT-path is particularly useful, if there is a lack of user traffic in the network or when active probing is desired.

For in-situ measurement architectures, if all traffic carries INT telemetry data, it will incur significant telemetry redundancy. The redundancy can be mitigated by packet sampling. However, sampling from the network ingress [17, 18] may still suffer considerable telemetry redundancy. Besides, it is difficult to achieve the network-wide telemetry coverage in a short interval. The postcard method [19] realizes the redundancy-free network-wide telemetry, but a large number of postcard packets can overwhelm the controller's CPU. Based on the observation that the device-internal states do not change frequently, some solutions only collect the information with drastic changes [17, 24]. LINT [43] tries to estimate the error of collector if the INT data is not piggybacked. However, these distributed labeling solutions cannot collect hop-by-hop data and provide network-wide telemetry coverage.

8 Conclusion

We propose *INT-label*, a lightweight In-band Network-Wide Telemetry architecture without using explicit probe packets. INT-label periodically labels the sampled user packets with device-internal states, which is cost-effective with a minor bandwidth overhead. In INT-label, the network-wide monitoring is decoupled from the network topology, allowing seamless adaptation to topology changes. INT-label is ready to be deployed in mega-scale data centers.

References

[1] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of little minions: Using packets for low latency network programming and visibility," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 3–14.

[2] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao *et al.*, "Packet-level telemetry in large datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 479–491.

[3] C. Hare, "Simple Network Management Protocol (SNMP)," 2011.

[4] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[5] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*. ACM, 2016, p. 10.

[6] C.-W. Chang, G. Huang, B. Lin, and C.-N. Chuah, "LEISURE: Load-balanced network-wide traffic measurement and monitor placement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 1059–1070, 2013.

[7] C. Jia, T. Pan, Z. Bian, X. Lin, E. Song, C. Xu, T. Huang, and Y. Liu, "Rapid Detection and Localization of Gray Failures in Data Centers via In-band Network Telemetry," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[8] Y. Zhou, J. Bi, T. Yang, K. Gao, C. Zhang, J. Cao, and Y. Wang, "KeySight: Troubleshooting Programmable Switches via Scalable High-Coverage Behavior Tracking," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 291–301.

[9] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

[10] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[12] "Barefoot Deep Insight," https://barefootnetworks.com/products/brief-deep-insight/, 2017.

[13] "1000-Fold Increase in Detection Precision of Packet Loss," https://www.huawei.com/ke/press-events/news/2019/6/first-ifit-pilot-5g-transport-network-beijing-unicom-huawei, 2019.

[14] "Broadcom trident 3," https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series, 2019.

[15] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2017, pp. 323–335.

[16] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, and Y. Liu, "INT-path: Towards Optimal Path Planning for In-band Network-Wide Telemetry," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 487–495.

[17] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. IEEE, 2018, pp. 1–3.

[18] S. Tang, D. Li, B. Niu, J. Peng, and Z. Zhu, "SelINT: A Runtime-Programmable Selective In-Band Network Telemetry System," *IEEE Transactions on Network and Service Management*, 2019.

[19] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 55–60.

[20] H. Song, T. Zhou, and Z. Li, "Export User Flow Telemetry Data by Postcard Packets," IETF Secretariat, Internet-Draft draft-song-ippm-postcard-based-telemetry-00, October 2018. [Online]. Available: http://www.ietf.org/internet-drafts/draft-song-ippm-postcard-based-telemetry-00.txt

[21] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: probabilistic in-band network telemetry," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 662–680.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[23] C. A. Sunshine, "Source routing in computer networks," *ACM SIGCOMM Computer Communication Review*, vol. 7, no. 1, pp. 29–33, 1977.

[24] S. Sheng, Q. Huang, and P. P. Lee, "DeltaINT: Toward general in-band network telemetry with extremely low bandwidth overhead," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–11.

[25] S. Zhu, J. Lu, B. Lyu, T. Pan, C. Jia, X. Cheng, D. Kang, Y. Lv, F. Yang, X. Xue *et al.*, "Zoonet: a proactive telemetry system for large-scale cloud networks," in *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, 2022, pp. 321–336.

[26] C.-H. He, B. Y. Chang, S. Chakraborty, C. Chen, and L. C. Wang, "A zero flow entry expiration timeout p4 switch," in *Proceedings of the Symposium on SDN Research*, 2018, pp. 1–2.

[27] "P4 Switch Behavioral Model," https://github.com/p4lang/behavioral-model, 2018.

[28] "Int-label repository," https://github.com/graytower/INT_LABEL, 2020.

[29] "In-band Network Telemetry (INT) Dataplane Specification," https://p4.org/p4-spec/docs/INT_v2_1.pdf, 2020.

[30] E. Song, T. Pan, C. Jia, W. Cao, T. Huang, and Y. Liu, "INT-filter: Mitigating Data Collection Overhead for High-Resolution In-band Network Telemetry," in *2020 IEEE Global Communications Conference (GLOBECOM)*.

[31] B. Claise, "Cisco systems netflow services export version 9," Tech. Rep., 2004.

[32] D. S. Lemons and P. Langevin, *An introduction to stochastic processes in physics*. JHU Press, 2002.

[33] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[34] M. A. Mollah, X. Yuan, S. Pakin, and M. Lang, "Rapid calculation of max-min fair rates for multi-commodity flows in fat-tree networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 1, pp. 156–168, 2017.

[35] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[36] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 127–132.

[37] "The documentation of Redis," https://redis.io/documentation/, redis documentation.

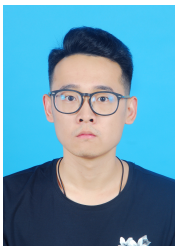
[38] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: enabling record and replay troubleshooting for networks," in *USENIX Annual Technical Conference*. USENIX Association, 2011, pp. 327–340.

[39] D. Yu, Y. Zhu, B. Arzani, R. Fonseca, T. Zhang, K. Deng, and L. Yuan, "dShark: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces," in *NSDI*, 2019, pp. 207–220.

[40] P. Phaal, S. Panchen, and N. McKee, "InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks," 2001.

[41] J. Quittek, T. Zseby, B. Claise, and S. Zander, "Require-

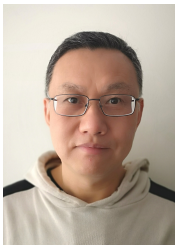
ments for IP flow information export (IPFIX)," RFC 3917 (informational), Tech. Rep., 2004.
[42] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "HPCC: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 44–58.
[43] S. R. Chowdhury, R. Boutaba, and J. François, "Lint: Accuracy-adaptive and lightweight in-band network telemetry," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 349–357.



Enge Song received the Ph.D. degree from the Department of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2022. His research interests include cloud network protocol and architecture, and programmable data plane.

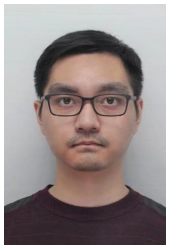
Tian Pan received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, in 2015. He is currently an associate professor with Beijing University of Posts and Telecommunications. His primary research interests include public cloud networks, data center networks, programmable data plane, and satellite networks.

Haoyu Song received the B.E. degree in electronics engineering from Tsinghua University in 1997 and the M.S. and D.Sc. degrees in computer engineering from Washington University in St. Louis in 2003 and 2006, respectively. He is currently a Senior Principal Network Architect with Futurewei Technologies, USA. His research interests include software defined networks, network virtualization and cloud computing, high performance networked systems, algorithms for packet processing, and intrusion detection.

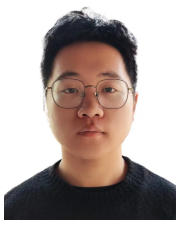


Qiang Fu received his Ph.D. from The University of Queensland. He is currently a Senior Lecturer in Cloud, Systems and Security with the School of Computing Technologies, RMIT University, Melbourne, Australia. His research interests are broadly in the areas of Internet and Cloud based systems including wireless and mobile systems. In particular, he has a focus on content delivery networks, data center design and analysis, virtualization as well as network programmability.

Yingjiang Liu received the B.S. degree from the Department of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2023. His research interests include data center networking and software-defined networking.



Chenhao Jia received the B.S. degree from the Department of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2019, and the M.S. degree from the Department of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2023. His research interests include software-defined networking, data center networking and protocol-independent forwarding.



Chuanying Yuan received the B.S. degree from the Department of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2022. His research interests include data center networking, protocol-independent forwarding, multicast routing for distributed training.

Minglan Gao received the M.S. degree from the Department of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2023. Her research interests include software-defined networking and protocol-independent forwarding.

Jiao Zhang received the B.S. degree from the School of Computer Science and Technology, Beijing University of Posts and Telecommunications in July 2008, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, in July 2014. She is a Professor with Beijing University of Posts and Telecommunications. Her research interests include data center networks, software-defined networking, and network function virtualization.

Tao Huang received the B.S. degree in communication engineering from Nankai University, in 2002, and the M.S. and Ph.D. degrees in communication and information system from Beijing University of Posts and Telecommunications, in 2004 and 2007, respectively. He is currently the Professor with Beijing University of Posts and Telecommunications. His research interests include network architecture, routing and forwarding, and network virtualization.

Yunjie Liu received the B.S. degree in technical physics from Peking University, in 1968. He is currently an Academician with the China Academy of Engineering, the Chief of the Science and Technology Committee of China Unicom, and the Dean of the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications. His research interests include next generation networks, network architecture, and management.