Taking detours: An in-network fault-tolerant probing planning for In-band Network Telemetry

Ariel G. Castro*, Igor Capelletti, Arthur F. Lorenzon*, Fabio D. Rossi+,
Roberto I. T. da Costa Filho$^\Psi$, Christian E. Rothenberg$^\gamma$, Marcelo C. Luizelli*
*Federal University of Pampa (UNIPAMPA), +Federal Institute Farroupilha (IFFAR)
$^\Psi$Federal Institute Sul-rio-grandense (IFSUL),$^\gamma$Campinas State University (UNICAMP)

***Abstract*—In-band Network Telemetry (INT) is a novel network monitoring approach mainly fostered by programmable network devices. Despite existing efforts toward the orchestration of INT, little has yet been done to provide fault-tolerant mechanisms in the data plane (e.g., to address hardware failure). In this paper, we introduce *InPatching* – an in-network approach to fast recover INT-based monitoring from network link failures. *InPatching* is implemented in the data plane and allows the application of detours in an autonomous and coordinated manner without the control plane intervention. To provide efficient detours to INT solutions, we formalize the fault-tolerant probing planning for INT by means of a MILP (Mixed-Integer Linear Programming) model. We prototype *InPatching* in P4 and we show that it can recover from fault conditions much faster than control plane solutions (up to 18X), while not imposing substantial overhead.**

I. Introduction

In-band Network Telemetry (INT) is an emerging network monitoring approach in programmable networks [1] that allows increasing network visibility of fine-grained network events (e.g., micro-bursts [2], load imbalance [3])). In short, INT works by continuously collecting low-level data plane statistics (a.k.a. telemetry data) from the infrastructure in a per-packet manner. These telemetry data include internal data plane statistics such as queue occupancy, per-packet processing time, and aggregated/computed statistics such as inter-packet gap [4].

In the classical INT operation (also known as INT-MD: eMbed Data), network packets are instructed to properly collect telemetry data as they are routed through the network. The instructions are added into an INT header, which can be embedded into either active network flows [5] or specially-crafted probing packets [6]. These packets are then interpreted by INT-enabled forwarding devices, which collect the requested telemetry data. Figure 1 illustrates the whole INT procedure using probing packets. Observe there are three probing cycles collecting data from the network, i.e., probing cycle f_1 collects telemetry data from nodes A, E, F, G, H, and I, returning to origin A (i.e., steps (1) – (5)), and then to an INT collector.

Recent research [1], [5], [7], [8] have made consistent efforts regarding the INT orchestration. The problem consists of efficiently using available resources (in this case, spare space on network packets) to collect data plane network statistics. In this context, Liu et al. [7] and Pan et al. [1] have focused on optimizing the usage of probing packets to collect INT data, while Marques et al. [8] and Hohemberger et al. [5] have

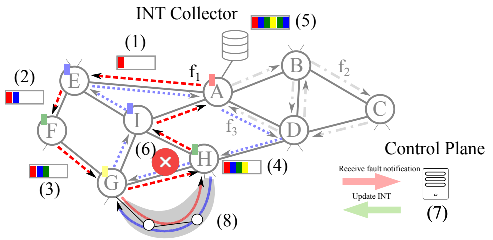


Fig. 1. Overview of INT planning.

focused on the embedding of telemetry data into production network packets. Castro et al. [9], in turn, leverages a shortest-path algorithm to reconstruct probe paths from link failures in the control plane. Despite the efforts toward INT orchestration, little has been done to provide fault-tolerant mechanisms for INT in the data plane. In case a network link fails, all of the INT monitoring mechanism that relies on that device is compromised. In Figure 1, for example, the failure of network link G–H directly affects probing cycles f_1 and f_2 (step 6).

A naive solution to provide fault-tolerance to this problem consists of computing a novel solution or adapting existing ones upon a failure (e.g., [1] [9]). In this case, a control plane application (step (7)) would be triggered to compute a new telemetry solution. In Figure 1, for instance, the new solution comprises a detour of probing cycles f_1 and f_2 through an alternative/updated path (step (8)). Despite this solution, the recovery of the INT monitoring approach would take, in the best case, a few hundred milliseconds. This is mainly due to the time required to identify the fault, the time spent to react (i.e., compute a new solution), and recovery (i.e., update the data plane). Consequently, the network-wide visibility required by monitoring applications might degrade in terms of coverage and freshness [8] during the faulty period.

To fill this gap, in this paper, we propose `InPatching`: an in-network approach to fast recovery INT-based monitoring approaches. In the event of faulty devices, `InPatching` autonomously (and without the control plane intervention) fix monitoring cycles by identifying the faulty device and applying detours in affected probing cycles to ensure the required INT data is collected correctly. `InPatching` is mainly offloaded to the data plane and, therefore, the recovery time of INT-based monitoring mechanisms can be made faster than existing control plane strategies. To provide efficient detours to existing probing cycles, we formalize the fault-tolerant probing planning for INT by extending the existing orchestration model [6]. Results show that *InPatching* outperforms control plane solutions by a factor of 18X. The main contributions of this paper can be summarized as follows:

- Proposal of an in-network strategy to quickly react to faulty network conditions;
- Formalization of the fault-tolerant INT probing planning;
- Open-source software artifacts for reproducibility.

The remainder of this paper is organized as follows. In Section 2, we discuss related work in the area of in-band network telemetry. In Section 3, we introduce the `InPatching` design in programmable data planes. In Section 4, we present and discuss the results of an evaluation of the proposed approach. Last, in Section 6, we conclude the paper with final remarks and perspectives for future work.

II. Related Work

Existing In-Band Network Telemetry Orchestration (INTO) approaches rely either on using (i) active flows [10], [8], [11], [12] or (ii) probe flows [13], [14], [1], [15], [16], [17] to collect telemetry demands across the network topology. Marques et al. [8] propose two heuristic strategies for collecting telemetry data, namely, *concentrate* and *balance*. The first strategy strives to aggregate telemetry data on a restricted number of flows, while the second tries to distribute equally the telemetry data over available network flows. Hohemberger et al. [10] is the first attempt to collect telemetry items in real-time coordinately. They design a machine-learning-based model and formalize the problem of collection to satisfy both spatial and temporal requirements, i.e., consider items must be collected from specific devices and at a certain rate to properly feed machine-learning applications on top of the network to detect anomalies (e.g., DDoS). Similarly, SDProber [13] performs a random decision for embedding INT data into probe packets, while Pan et at. [1] utilizes *Euler Circuits* and DFS-like strategies to orchestrate probing packets across the network. However, these works construct a static solution and do not consider devices may fail.

Scano et al. [12] extend P4 INT to 5G. Packet flows carrying selected latency-sensitive services are proposed to encompass the INT header already from the user equipment, allowing for rerouting packets when soft failures are detected (e.g., increased bit errors, occasional packet loss, link congestion). Patcher [9] is a fault-tolerant mechanism that leverages the shortest path algorithm [18] performing "patches" on affected probe flows where device faults occurred - e.g., due to energy failure, misconfiguration. Still, the failures must be communicated to a control plane where the "patches" are performed and informed to data plane devices, incurring high latency - intolerable for low-latency applications (e.g., VoIP).

More recently, Fast Rerouting (FRR) mechanisms have been used to implement fault-tolerant routing schemes directly on the fast path in the data plane. This approach aims to reduce path recovery time by requiring minimal or no control plane intervention. PURR [19] is an FRR primitive that supports multiple failures and avoids re-circulations. First, the switches send the packet on the first active port in a sequence. In case of failure, multiple mechanisms may be used to re-establish the connection. Blink [20] passively monitors non-negligible TCP re-transmissions to detect remote link failures that disrupt end-to-end connectivity. Basically, it probes all the next hops for availability and chooses a new working one. Subramanian et al. [21] present D2R, which provides policy-compliant paths. They logically split the topology into domains to reduce the memory overhead of alternative path computation. Nevertheless, if a domain becomes internally disconnected due to multiple failures, D2R may not find a route to the destination - even if it exists.

Despite recent efforts [9], [20], [19], [21] on providing routing fail-over mechanisms in the data plane, none of these solutions properly work to recover INT data collection. In particular, these solutions are likely to fail to keep an uninterrupted collection of INT data as the fail-over mechanisms are not designed for that. It turns out that the INT downtime is dominated by the time needed to compute path updates in the controller, which can take several hundreds of milliseconds depending on the size of the network. Even employing pre-computed paths, the delay in receiving the failure notification at the controller, as well as in propagating and installing the new forward entries in the data plane devices is still significant, leading to loss of network visibility. To the best of our knowledge, `InPatching` is the first approach to offload recovery decisions to the data plane entirely. Our approach can react quickly to faulty network conditions in an autonomous and coordinated manner, outperforming state-of-the-art control plane solutions by a factor of 18X.

III. `InPatching`: An in-network fault-tolerant strategy for INT

A. Overview

`InPatching` is an in-network fail-over approach to INT-based monitoring mechanisms. Figure 2 depicts the in-network approach in a programmable network infrastructure. In the example, three probing cycles f_1, f_2, and f_3 are responsible for continuously (i) collecting telemetry data and (ii) checking network connectivity. For simplicity, we omit the telemetry data collected at each device in the figure and assume a single independent failure of network links. In the example, network link G–H has failed, and therefore affected probing cycles f_1 and f_3 – that is, INT packets are not returning to the INT Collector with collected telemetry data.

`InPatching` aims to offload the fail-over mechanism to the data plane in order to reestablish the monitoring without the control plane intervention. For that, it relies (i) on alternative paths computed in advance, and (ii) on a data plane heuristic strategy to select the proper alternative path. Upon a link failure, probing packets start to time out in device A (in the

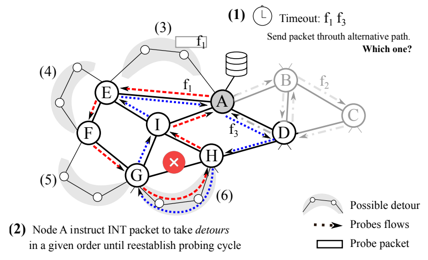

(1) Timeout: f1 f3
Send packet throuth alternative path.
Which one?
(3)
f1
(4)
E
A
B
f2
C
F
I
f3
D
H
G
(5)
(6)
(2) Node A instruct INT packet to take detours in a given order until reestablish probing cycle
Possible detour
Probes flows
Probe packet

Fig. 2. Overview of the in-network `InPatching` strategy.

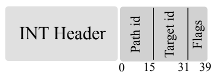


Fig. 3. `InPatching` header structure.

examples, probing packets from f_1 and f_3), indicating to the data plane that probing packets have not arrived back on time.

In turn, the data plane autonomously reacts to that by instrumenting the next packet of affected probing flows to take a detour in the main probing path as a way to circumvent the connectivity problem. In the Figure 2, the data plane of device A is in charge of taking that decision. The first attempt is to perform a detour between nodes A and E. In this case, the packet sent through this alternative path does not return (since the failure is on network link G–H). Then, the data plane attempts to use an alternative detour on nodes E and F, which also does not solve the connectivity problem. Eventually, the data plane uses an alternate path between nodes G and H, reestablishing the INT monitoring mechanism. Last, from time to time, the data plane attempts to utilize back the original forwarding path. Note that the data plane is taking heuristic decisions, and in the worst case, the number of attempts is $O(n)$, where n is the length of the probing path. As we discuss next, we can optimize these decisions to minimize the number of attempts, minimizing the overall recovery time.

Two building blocks are required to realize the potential of `InPatching`. The first is the data plane itself. As mentioned, the data plane needs to react to time-out conditions and to instrument other data planes to take a detour. The second is the orchestration model (implemented in the control plane) that is responsible for constructing appropriate probing cycles and detours for network links.

B. Data plane design

The proposed data plane follows a distributed design similar to the well-known master-slave approach. One data plane logic is set as master and takes the main decisions (e.g., whether or not to take a detour). The others (i.e., the slaves) implement a simplified data plane logic to forward probing packets. It is important to mention that all data plane logic has primary and alternative paths installed in advance by the control plane (we will discuss how we do that in the following subsection).

Our design works by extending the classical INT-based header struct. All probing packets have an INT header instructing which INT telemetry data must be collected at each forwarding device. The `InPatching` header struct is appended just after the INT header and is 5 Bytes long (i.e., 0.003% of a regular 1500B MTU frame). These Bytes are used to monitor time-outs and to instruct the slave data plane on how to react. Figure 3 depicts the `InPatching` header struct. The first 16 bits represent the path id of a probing cycle. Then, the target id represents the slave data plane id being targeted by the master data plane. In other words, it represents the data plane in which the detour decision is made. Last, there is an 8-bit word to represent possible control flags. `InPatching` utilizes these flags to notify the slave data plane to attempt to use the main path or to force them to stay in the alternative path. Our approach periodically injects a packet trying to return to the main path.

The master data plane logic needs to (i) keep track of probing cycle time-outs and (ii) react in case of time out by instrumenting data plane slaves on possible detours. Figure 4 illustrates the whole procedure. The INT header is processed (steps 1-2) whenever a packet ingresses the device pipeline. It interprets INT instructions and collects INT data at a given device. The master data plane is uniquely identified by an *id* and it is responsible for (i) encapsulating packets in the `InPatching` header (step 4), (ii) keeping track of probing packet time-outs (steps 5 and 6), and (iii) choosing a slave/target data plane to perform a detour (step 7). For each probing cycle, the master data plane keeps an array of $|P|$ register (each of 48 bits) to store data plane timestamps, where P is the set of probing cycles in the network. When a packet ingress the master data plane, the data plane compares the packet ingress timestamp with the last seen packet timestamp of the same probing cycle P. If this difference is greater than a fixed threshold, the data plane assumes a time-out has happened. Then the master data plane notifies a slave to follow an alternative path (step 6). For that, the master stores a list of device *ids* in a probing cycle using an array of registers. The order of this *ids* in the array defines the order in which the data plane tries to apply a detour. This order is defined in advance by the control plane and can be adjusted based on failure probabilities, for example. When a detour is applied, the `InPatching` header is updated with the target data plane *id* (step 7). That information in the header field is used by the slaves in order to either apply alternative paths (steps 8-9) or not (step 10). Furthermore, whenever a packet returns to the master data plane (steps 11-12), we update the time-out data structure for each probing cycle to keep track of the last seen probing packet. To properly control returning packets to the master, we use the field flags in the `InPatching` header. Last, the packet is sent out to a specific port in step 13.

C. Control plane design

As previously mentioned, the main decision of the `InPatching` approach is taken in the data plane. However,

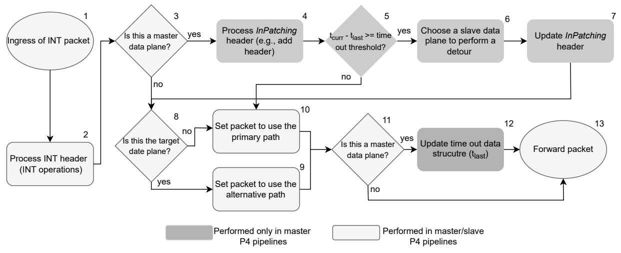

1
Ingress of INT packet
2
Process INT header (INT operations)
3
Is this a master data plane?
yes
no
4
Process *InPatching* header (e.g., add header)
5
t_{curr} - t_{last} >= time out threshold?
yes
no
6
Choose a slave data plane to perform a detour
7
Update *InPatching* header
8
Is this the target date plane?
no
yes
10
Set packet to use the primary path
9
Set packet to use the alternative path
11
Is this a master data plane?
yes
no
12
Update time out data strucutre (t_{last})
13
Forward packet
Performed only in master P4 pipelines
Performed in master/slave P4 pipelines

Fig. 4. Overview of the `InPatching` data plane procedure.

the control plane still plays an important role by defining the probing paths [1], [5], [7], [8], the detours, and the order in which they are applied by the data plane. In particular, how the control plane defines the probing paths directly impacts the efficiency of `InPatching`.

To understand the impact of probing paths on the efficiency of `InPatching`, let's consider the example given in Figure 2. In case a failure occurs at a network link covered by only one probing path (e.g., network links E–F or B–C), the heuristic strategy taken by the `InPatching` data plane is in the worst case $O(n)$ (where n is the probing cycle length). This happens because affected probing paths f_1 and f_2 are disjointed, and independently of the order `InPatching` applies a detour, the worst case remains $O(n)$. However, if the failure happens at a network link shared by multiple probing paths, the decision can be optimized by cooperatively applying the heuristic by the data plane.

Let's suppose the master data plane is aware of the subset of overlapped probing paths. In the example of Figure 2, probing cycles $F_s = \{f_1, f_3\}$ share network links $S = \{(A-E), (G-H)\}$. In case of a network failure in S, the master data plane would reduce the search space to $O(\frac{|S|}{|F_s|})$. In that case, network flows in F_s would time out, and the master data plane could distribute the decision amongst them cooperatively. Whenever a probing flow finds a valid detour, it updates the master data plane accordingly.

Defining Optimized Probing Paths. To implement this unified solution by the data plane, the control plane runs an INTO model that is able to generate a valid set of probing cycles in such a way the network infrastructure links are covered by multiple probing cycles. Therefore, in the event of a network link failure, multiple affected INT cycles are promptly noticed and repair these links.

The optimization problem described next ensures a valid INTO solution with a minimal number of probing cycles while ensuring at least a $\mathcal{K} \in \mathbb{N}^+$ cycle coverage per network link. We adopt a revised and extended version of the model proposed by [6]. Similar to them, we consider a programmable network infrastructure $G = (D, L)$ and a set of telemetry items V. Set D of network G represents P4-enabled forwarding devices $D = \{1, ..., |D|\}$, while set L links interconnecting devices $(d_1, d_2) \in (D \times D)$. There is a set of available telemetry data V, where each $v \in V$ has its size defined by the function $S : V \to \mathbb{N}^+$. Each device $d \in D$ can embed a subset of items $V_d \subseteq V$ into a probing cycle packet $p \in P$. Packets in a probing cycle $p \in P$ have limited spare space to embed V data, defined by the function $U : P \to \mathbb{N}^+$. The set of probing cycles P is routed within the network G – that is, the packet is generated at an INT sink, routed through a subset of devices, and returns to its origin. A probing packet can visit a device $d \in D$ and not collect any associated telemetry items. We denote the origin of each cycle $p \in P$ as a fixed forwarding device $d_o \in D$.

The variable set of the optimization model is defined as follows. Variable $z_{p,v,i}$, $(\forall\, p \in P, v \in V, i \in D)$ indicate that a device i embeds INT data v into a probing packet from cycle p. Variable $x_{p,i,j}$ $(\forall p \in P, (i,j) \in L)$ indicate that network link $(i,j) \in L$ is being used to route probing cycle $p \in P$. Last, variables y_p $(\forall p \in P)$ and $w_{p,i,j}$ $(\forall p \in P, (i,j) \in L)$ are used, respectively, to count probing cycles used by the solution and by network link.

$$\textbf{Minimize} \quad \sum_{p=1}^{P} y_p \tag{1}$$

Subject to:

$$z_{p,v,i} \leq \sum_{j \in D} x_{p,j,i} \qquad \forall p \in P, i \in D, v \in V_i \tag{2}$$

$$z_{p,v,i} + x_{p,i,j} \leq 2 \cdot y_p \qquad \forall p \in P, (i,j) \in L, v \in V_i \tag{3}$$

$$\sum_{j \in D} x_{p,i,j} - \sum_{j \in D} x_{p,j,i} = 0 \qquad \forall p \in P, i \in D \tag{4}$$

$$\sum_{i \in S} \sum_{j \in S} x_{p,i,j} \leq |S| - 1 \qquad \forall p \in P, S \subseteq \{D - d_o\}, |S| \geq 2 \tag{5}$$

$$\sum_{p \in P} x_{p,i,j} + x_{p,j,i} \geq 1 \qquad \forall (i,j) \in L \tag{6}$$

$$\sum_{i \in D} \sum_{v \in V_i} z_{p,v,i} \cdot S(v) + \sum_{i \in D} \sum_{j \in D} x_{p,i,j} \leq U(p) \qquad \forall p \in P \tag{7}$$

$$x_{p,i,j} \leq \mathcal{B} \cdot w_{p,i,j} \qquad \forall p \in P, (i,j) \in L \tag{8}$$

$$\sum_{p \in P} w_{p,i,j} \geq \mathcal{K} \qquad \forall (i,j) \in L \tag{9}$$

$$z_{p,v,i} \in \{0, 1\} \qquad \forall p \in P, v \in V_i, i \in D \tag{10}$$

$$y_p \in \{0, 1\} \qquad \forall p \in P \tag{11}$$

$$x_{p,i,j} \geq 0 \qquad \forall p \in P, (i,j) \in L \tag{12}$$

$$w_{p,i,j} \in \{0, 1\} \qquad \forall p \in P, (i,j) \in L \tag{13}$$

Constraint set (2) ensures that if telemetry item v is collected from forwarding device i, then a probe should be routed through i. Constraint set (3) accounts for the number of probing cycles in use. In turn, constraint sets (4) and (5) ensure that cycles are well crafted. That is, constraint set (4) ensures flow conservation, while constraint set (5) is the well-known sub-tour elimination constraint. Then, constraint set (6) guarantees a probing cycle that covers at least one link direction. Constraint set (7) ensures the available probing packet capacity is not violated by the telemetry items collected or the network links being covered. Constraint sets (8) and (9) ensure a network link is covered by at least $\mathcal{K}$ probing cycles, where $\mathcal{B}$ is a big natural number. Last, constraint sets (10)–(13) define the domains of output variables, while Equation

(1) aims at minimizing the number of probing cycles.

Defining Detours. Another key element of `InPatching` is the detours taken by probing cycles in the data plane in case of network link failure. We define detours as simple paths in G between two forwarding devices belonging to the same probing cycle P. We denote by $N_p = \{d_1, d_2, ..., d_{|}N_p|\}$ the ordered set of forwarding device of a probing path $p \in P$. For each ordered pair (d_1, d_2), (d_2, d_3), ..., $(d_{|N_p|-1}, d_{|N_p|})$ we define a simple alternative path between devices of a pair.

As previously mentioned, in the event of a failure, the data plane heuristically selects one alternative path in order to address the faulty condition. The control plane also defines the order to instruct the master data plane on how to proceed. As for the experiments shown next, we adopt a simplified round-robin alternative following the order of nodes in the probing path P. However, other approaches are easily implemented, such as prioritizing high-probability faulty nodes first.

IV. Experimental Evaluation

A. Setup

We implemented our `InPatching` data plane approach in P4 using BMv2 virtual switches, and we evaluated it in a Mininet environment. Probing cycles are defined according to our optimal model (Section III.C), while probing packets are generated using a custom-made Scapy code. All experiments were performed on a machine equipped with an AMD Ryzen Threadripper 3990X with 64 physical processors, and 32 GB of memory, running Ubuntu 18.04. Each experiment was repeated 30 times, which was enough to guarantee a confidence level higher than 95%. Our source code is available on GitHub [22].

B. *`InPatching` Data Plane vs. Control Plane Approach*

We first evaluate the effectiveness of `InPatching` in recovering the INT-based monitoring mechanism from single-link failures compared to a control-plane based solution [9]. For that, we deploy a set of disjoint probing cycles in Mininet and inject link failures to network links in P. For this experiment, we set all network links in Mininet to have a 1ms delay and we varied the time out in the data plane from 10ms to 100ms. The time out is the reaction time, that is, the time the data plane waits for an injected probing packet before reacting to it. Therefore, when time out occurs, `InPatching` is applied over the next packet or sent to the control plane.

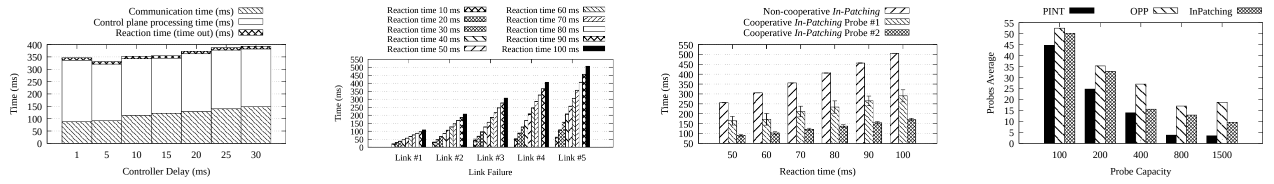
Figure 5(a) and Figure 5(b) illustrate, respectively, the recovery time for the control and data plane approaches. Figure 5(b) depicts the proposed `InPatching` data plane approach considering an increasing value of time out. For this experiment, we consider a 5-hop probing cycle (i.e., link #1 to link #5). As we observe, the data plane increases the recovery time linearly as we increase the time out. For instance, for a data plane time out of 10ms, the recovery time of the entire INT monitoring system is ∼19ms. Further, we also can observe that the round-robin heuristic approach also affects the data plane recovery time. The more distant the failure happens from the probing cycle origin, the more time is required to `InPatching` to find a valid detour. However, even in the more distant link (e.g., link #5), the recovery time is below 60ms for a time out of 10ms. In contrast, Figure 5(a) illustrates the recovery time in the control plane. In this experiment, we fixed the time out as 10ms. The control plane recovery time depends on (i) the control plane processing time (i.e., the time required to compute a solution), (ii) communication time (i.e., the time to send and receive information between infrastructure and controller), and (iii) reaction time (i.e., the time out). When a packet is timed out by the data plane, it clones the packet and sends it to the control plane. In the case of the control plane, the network link failure order does not affect the reaction time since we are sending them to the control plane. We varied the control plane distance from the probing path origin (from 1 to 30ms) in order to observe the communication time. In the best case (for 1ms control plane latency), `InPatching` is up to 18x quicker than the control plane (i.e., $\sim$ 350ms to the control plane vs. $19ms$ of `InPatching`).

C. *The gain of overlapping probing INT cycles*

Next, we evaluate the gains attained to `InPatching` when probing cycles are constructed by our model considering a given overlapping (i.e., $\mathcal{K} > 1$). In our P4 implementation, we allow $\mathcal{K} = 2$ and, therefore, we ensure that at least two probing paths cover the same network link. Figure 5(c) illustrates the recovery time for a reaction time (time out) varying from 50ms to 100ms. In case of a failure, two probing paths time out and the `InPatching` data plane coordinately attempt to solve the failure by applying a detour simultaneously. In the experiment, we consider two probing cycles sharing a network link. In the event of a network link failure, the two probing cycles attempt to recover by applying a detour. The first one to find a valid detour notifies the other in the origin data plane. The Figure 5(c) illustrates the non-cooperative and cooperative versions of `InPatching`. In the cooperative version, the first probing cycle (probe #1) is the one that finds a valid detour, notifying the second probing cycle (probe #2). The cooperative `InPatching` achieves a recovery time up to 50% and 300% faster than the non-cooperative version of `InPatching` for the first and second probing cycles, respectively.

D. *The cost of overlapping*

Last, we evaluate how our proposed optimal model impacts the number of probing cycles generated in an INTO solution (in particular, Equations 8, 9, and 13). Our model is implemented using IBM CPLEX Optimization Studio 12.9 to obtain optimum solutions and our results are compared to [1], [6]. We follow a similar approach to them to generate a workload, setting $\mathcal{K} = 2$ as this is the parameter evaluated in the previous subsection. Figure 5(d) illustrates the number of probing cycles for increasing the size of probing capacity (from 100B to 1500B). As we increase the probing capacity, we observe a decrease in the number of deployed INT probing cycles – as we do have more space on packets to collect INT data. For small packets (100B, 200B, 400B), our model requires on average 15% more probing cycles than the solution provided by [6]. To larger packets (800B and 1500B), our solutions require on



(a) Control plane approach.

(b) `InPatching` data plane approach.

(c) Cooperative `InPatching` data plane vs non-cooperative.

(d) `InPatching` optimal model

Fig. 5. Control Plane vs. `InPatching` Data Plane Approach

average two cycles more. Yet, our model produces 40% fewer cycles than [1]. Due to space constraints, we omitted other analytical evaluations of our model.

V. Final Remarks

In this paper, we introduced *InPatching*, an in-network fault-tolerant approach to INT monitoring solutions. *InPatching* offloads the fail-over mechanism to the data plane in order to reestablish the monitoring without the control plane intervention. For that, it relies on alternative paths computed in advance and on a data plane heuristic strategy to select the proper alternative path. As shown, our solution *(i)* outperforms control plane solutions by a factor of 18X *(ii)* finds a valid a detour solution up to 3X quicker when coordinated with other probing flow, and (iii) requires no more probing cycles on average than state-of-the art In-Band Network Telemetry Orchestration approaches. As future work, we intend to evaluate other data plane heuristics and implement our solution on a programmable hardware.

Acknowledgements

This work was partially funded by National Council for Scientific and Technological Development (CNPq 404027/2021-0), Foundation for Research of the State of Sao Paulo (FAPESP 2021/06981-0, 2021/00199-8, 2020/05183-0), and Foundation for Research of the State of Rio Grande do Sul (21/2551-0000688-9).

References

[1] 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*, Apr 2019, pp. 1–9.

[2] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, and O. Rottenstreich, "Catching the microburst culprits with snappy," in *Proceedings of the SelfDN*. New York, NY, USA: ACM, 2018, p. 22–28.

[3] P. Tammana, R. Agarwal, and M. Lee, "Distributed network monitoring and debugging with switchpointer," in *15th USENIX NSDI 18*, Renton, WA, 2018, pp. 453–456.

[4] S. K. Singh, C. Rothenberg, M. C. Luizelli, G. Antichi, and G. Pongracz, "Revisiting heavy-hitters: Don't count packets, compute flow inter-packet metrics in the data plane," in *ACM SIGCOMM Poster*. New York, NY, USA: ACM, 2020, pp. 1–4.

[5] R. Hohemberger, A. G. Castro, F. G. Vogt, R. B. Mansilha, A. F. Lorenzon, F. D. Rossi, and M. C. Luizelli, "Orchestrating in-band data plane telemetry with machine learning," *IEEE Communications Letters*, vol. 23, no. 12, pp. 2247–2251, 2019.

[6] A. G. Castro, A. F. Lorenzon, F. D. Rossi, R. I. da Costa Filho, F. M. Ramos, C. E. Rothenberg, and M. C. Luizelli, "Near-optimal probing planning for in-band network telemetry," *IEEE Communications Letters*, vol. 25, no. 5, pp. 1630–1634, 2021.

[7] Z. Liu, J. Bi, Y. Zhou, Y. Wang, and Y. Lin, "Netvision: Towards network telemetry as a service," in *IEEE ICNP*, Sep. 2018, pp. 247–248.

[8] J. A. Marques, M. C. Luizelli, R. I. T. Da Costa, and L. P. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *Journal of Internet Services and Applications*, no. 1, p. 16, Jun 2019.

[9] A. G. Castro, V. H. Lopes, F. G. Vogt, F. D. Rossi, A. F. Lorenzon, and M. C. Luizelli, "Patcher: Towards fault-tolerant probing planning for in-band network telemetry," in *2020 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE, 2020, pp. 1–6.

[10] R. Hohemberger, A. G. Castro, F. G. Vogt, R. B. Mansilha, A. F. Lorenzon, F. D. Rossi, and M. C. Luizelli, "Orchestrating in-band data plane telemetry with machine learning," *IEEE Communications Letters*, vol. 23, no. 12, pp. 2247–2251, 2019.

[11] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," ser. SIGCOMM '20. New York, NY, USA: ACM, 2020, p. 662–680.

[12] D. Scano, F. Paolucci, K. Kondepu, A. Sgambelluri, L. Valcarenghi, and F. Cugini, "Extending p4 in-band telemetry to user equipment for latency-and localization-aware autonomous networking with ai forecasting," *Journal of Optical Communications and Networking*, vol. 13, no. 9, pp. D103–D114, 2021.

[13] S. Ramanathan, Y. Kanza, and B. Krishnamurthy, "Sdprober: A software defined prober for sdn," in *Proceedings of the Symposium on SDN Research*, 2018, pp. 1–7.

[14] Z. Liu, J. Bi, Y. Zhou, Y. Wang, and Y. Lin, "Netvision: Towards network telemetry as a service," in *IEEE ICNP*, Sep. 2018, pp. 247–248.

[15] D. Bhamare, A. Kassler, J. Vestin, M. A. Khoshkholghi, and J. Taheri, "Intopt: In-band network telemetry optimization for nfv service chain monitoring," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[16] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Simon: A simple and scalable method for sensing, inference and measurement in data center networks," in *NSDI 19*, 2019, pp. 549–564.

[17] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, and J. Wu, "Netview: Towards on-demand network-wide telemetry in the data center," *Computer Networks*, p. 107386, 2020.

[18] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[19] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, and S. Schmid, "Purr: a primitive for reconfigurable fast reroute: hope for the best and program for the worst," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 1–14.

[20] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 161–176.

[21] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, "D2r: Policy-compliant fast reroute," in *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, 2021, pp. 148–161.

[22] InPatching. (2022, Nov) Inpatching - github repo. Accessed on Nov, 2022. [Online]. Available: https://anonymous.4open.science/r/InPatching-E16B/README.md