IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 3, 2021, accepted January 22, 2021, date of publication January 28, 2021, date of current version February 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3055462

Passive In-Band Network Telemetry Systems: The Potential of Programmable Data Plane on Network-Wide Telemetry

PILAR MANZANARES-LOPEZ, JUAN PEDRO MUÑOZ-GEA, AND JOSEMARIA MALGOSA-SANAHUJA, (Senior Member, IEEE)
Department of Information Technologies and Communications, Universidad Politecnica de Cartagena, 30202 Cartagena, Spain

Corresponding author: Pilar Manzanares-Lopez (pilar.manzanares@upct.es)

This work was supported by the Research State Agency (RSA)/European Regional Development Funds (ERDF) through the European Union Project under Grant TEC-2016-76465-C2-1-R (AIM).

ABSTRACT In the last few years, the emergence of Programmable Data Planes and the appearance of programming protocol-independent languages such as P4 have offered powerful tools to define new network protocols, as well as to redesign existing network applications and systems. Network telemetry is one of the main areas of interest identified by the P4 Application Working Group. The collection of network-wide, fine-grained network information in real-time is a critical requirement for the design of useful and adequate monitoring tools that can be integrated into complex Operations, Administration & Maintenance applications. Recent research has focused on the definition and implementation of in-band monitoring systems, where specifically dedicated monitoring packets are not required. Even though the In-Band Network Telemetry specification proposed by the P4 Language Consortium is the starting point of many of the in-band monitoring systems, this is not the only alternative. Therefore, in this work, we will describe and compare other P4-based in-band passive telemetry proposals.

INDEX TERMS In-band network telemetry, network monitoring, P4, passive monitoring, programmable data plane, remote network management.

I. INTRODUCTION

Efficient solutions to monitor the performance of a network, to detect congestion, failures and anomalies, and the ability to respond to them in real-time have been, and will remain, a key point in current networks. Many passive and active monitoring solutions have been proposed and developed over the years to provide a network-wide perspective of the network status to Operations, Administration & Maintenance (OAM) applications [1].

Active monitoring (which is also known as synthetic monitoring) involves the injection of test traffic and then the measurement of network performance. This offers a proactive approach to detect any potential problems before they happen. However, active monitoring does not always offer an accurate view of the network's performance. In contrast, passive monitoring analyses real network traffic, which involves the capturing of all, or just some part, of the traffic flowing through the network, which offers a more accurate view of the network.

Passive monitoring is a common approach, which collects counters and statistics directly from network devices using protocols such as SNMP [2] or NETCONF [3]. However, passive monitoring's polling-based nature and high processing overhead can lead to performance limitations. Consequently, large-scale networks require an alternative approach, which has led to the proposal of network telemetry solutions, where network devices push specific network metrics in real-time (or not) to a collector.

In the last few years, the emergence of programmable data plane (PDP) [4], [5] has led to a new line of network telemetry solutions. PDP is an emerging technology for programming packet processing tasks, which works by means of a domain-specific high-level language and programmable switches. POF [6] and P4 [7] are two of the most prominent programming languages that enable data plane

The associate editor coordinating the review of this manuscript and approving it for publication was Zehua Guo.

programmability. The recent advances of data plane programmability have led to possible the proposal of new telemetry methods that can perform end-to-end monitoring directly in the data plane. This network telemetry is based on in-band measurement where monitoring information is embedded into data packets as they traverse the network rather than being sent within specifically dedicated packets.

Programming Protocol- independent Packet Processor (P4) [7] is a representative data plane programming language that provides packet processing abstractions in networking elements in a target-independent way. Although POF and P4 have the same goal, the P4 language provides higher level abstractions [8]. In P4, a program codes how packets are going to be processed; that is, the P4 program defines the packet forwarding behaviour within a network element, and later a compiler generates a configuration for a specific and protocol-independent switch.

At this point, it is interesting to clarify the difference between P4 and OpenFlow (which is one of the most widespread SDN (Software Defined Networking) protocols). The OpenFlow protocol [9] was developed as a communication protocol to allow interoperability between the control plane and the data plane. OpenFlow functions consider a set of predefined protocol headers that must be supported by the OpenFlow-compatible switches. Although the OpenFlow specification has grown from a reduced set of matching fields to dozens of fields, multiple tables, meters and groups [10], the inclusion of new protocols or protocol changes is not trivial or immediate. In contrast, P4 avoids the need to extend the OpenFlow specification, providing adaptable header definition and matching functionality. The P4 language offers end-users the ability to create custom protocols and algorithms in an agile and generic way. Another key feature of the P4 language is that thanks to the definition of advanced programming functions, such as custom pipelines and memory registers that can be accessed when processing a packet, P4 offers stateful programming [11]. Thus, forwarding procedures can be defined based on the network state, variations, and history of flow statistics at node level, without requiring the intervention of a controller. On the other hand, stateful processing without controller intervention is minimal in OpenFlow.

The P4 framework offers researchers a powerful tool to enhance existing network applications. The P4 Application Working Group identifies, among others, the following as some areas of interest: forwarding-plane telemetry; flow monitoring using sketches; heavy-hitter detection in the data plane; low latency congestion control; big data aggregation inside the network; middlebox functions (e.g. layer-4 connection load balancing); fast in-network cache of distributed services; and, consensus protocol at network speed.

In this survey, we will focus on the first line, and we will review the main network-wide in-band telemetry solutions that are implemented in P4 (see table 3). It is important to remark that *in-band* refers to the fact that monitoring and data collection is directly performed in the data plane.

TABLE 1. A list of the most used acronyms.

Abb.	Definition
AM-PM	Alternate Marking-Performance Marking
BMv2	Behavioral Model
eBPF	extended Berkeley Packet Filter
FS-INT	Flexible Sampling-based INT
GRE	Generic Routing Encapsulation
INT	In-Band Network Telemetry
INTO	In-Band Network Telemetry Orchestration
KDN	Knowledge-Defined Networking
ML-INT	Multi-Layer In-band Network Telemetry
MTU	Maximum Transmission Unit
OAM	Operation, Administration & Maintenance
OVS	Open vSwitch
ONOS	Open Network Operating System
P4	Programming Protocol-independent Packet Processor
PAINT	Policy-Aware In-band Network Telemetry
PISA	Protocol Independent Switch Architecture
PDP	Programmable Data Plane
POF	Protocol-oblivious forwarding
PNA	Portable NIC Architecture
PSA	Portable Switch Architecture
SAI	Switch Abstraction Interface
SDN	Software Defined Networking
sINT	Selective INT
SNMP	Simple Network Management Protocol
SR	Segment Routing
VXLAN	Virtual eXtensive LAN
XDP	eXtended Debug Port

As will be described in the following sections, In-Band Network Telemetry (INT) is perhaps the main network monitoring framework implemented using P4, and is developed by P4 Language Consortium [12]. However, alternative network telemetry approaches that make use of P4 have also been proposed. The main objective of this article is to survey the academic publications and projects related to this topic.

The remainder of the paper is organised as follows. Section II offers a description of the background and fundamental information of P4 ecosystem. Next, Section III to V present the new trends in network-wide passive in-band telemetry solutions using P4. The INT approach is described in Section III, which brings together the majority of the works. Based on the use of INT dataplane specification, different monitoring systems have been proposed in the last few years. However, this is not the only in-band network-wide telemetry proposal. Section IV describes another recent proposal called Alternate Marking-Performance Measurement (AM-PM) that has been implemented in P4 network elements to implement a monitoring system. The last reviewed approach is called FlowStalker and is described in Section V. Section VI presents a comparison of the described proposals considering some network telemetry challenges. Finally, Section VII concludes this article and it also identifies future lines of work. Table 1 provides a list of most used acronyms in the paper.

II. P4 ECOSYSTEM REVIEW

A. INTRODUCTION

P4 is a high-level language for programming protocol-independent packet processors. P4 is used to program how

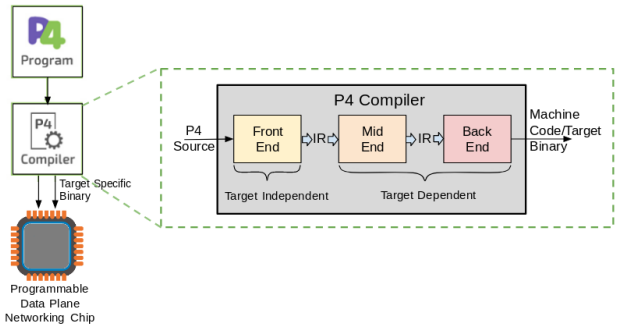

FIGURE 1. P4 Compiler Architecture. From [13].

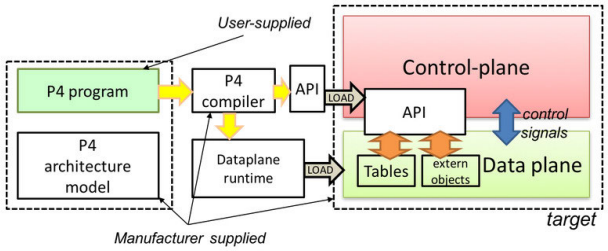

FIGURE 2. Workflow for programming a network device using P4.

packets are to be processed by the data plane of programmable forwarding elements (e.g. hardware or software switches, network interface cards, routers, or network appliances), which are commonly known as p4 targets. The name P4 comes from the original paper that introduced the language [7]. The P4 specification is open and public.

The P4 language is the main component of the P4 ecosystem. It is a high-level domain-specific language that is dedicated to the programming of network elements. It allows the user to specify the format of packets (protocol's headers) to be recognised by network devices and the actions to be performed on incoming packets (e.g. forwarding, headers modification, adding protocol header). P4 also allows the definition of stateful forwarding behaviours based on the use of memory registers that can be accessed when processing a packet.

As represented in Fig. 1, the P4 source code is not directly consumed by the network elements. A P4 program must be compiled for a particular platform. These platforms can be hardware-based (e.g. Barefoot Tofino, FPGA) or software-based (e.g. BMv2, eBPF/XDP, PISCES or P4rt-OVS). P4c is a reference p4 compiler [14], that supports both P4 language versions P4-14 [15] and P4-16 [16]. A survey of P4 compilers can be found in [17].

Fig. 2 shows a typical workflow when programming a P4 target. Target manufacturers provide a model of a specific hardware or software implementation, an architecture definition (that is, the P4 programmable blocks and an API to program the target) and a P4 compiler for the target. The P4 compiler maps the P4 source code into the target model, and it then produces a data plane configuration that implements the forwarding logic and an API to manage the state of the data plane objects from the control plane.

Programming the data plane using P4 offers several advantages, as summarised in [18], as follows:

- Flexibility: P4 makes many packet-forwarding policies expressible as programs, in contrast to traditional switches, which expose fixed-function forwarding engines.
- Expressiveness: P4 can express sophisticated, hardware-independent packet processing algorithms using solely general-purpose operations and table look-ups. These programs are portable across hardware targets that implement the same architectures (assuming that sufficient resources are available).
- Resource mapping and management: P4 programs describe storage resources abstractly (e.g. IPv4 source address), while the compilers map such user-defined fields to available hardware resources and manage low-level details such as allocation and scheduling.
- Software engineering: P4 programs provide important benefits, such as type checking, information hiding, and software reuse.
- Component libraries: Component libraries supplied by manufacturers can be used to wrap hardware-specific functions into portable high-level P4 constructs.
- Decoupling hardware and software evolution: Target manufacturers may use abstract architectures to further decouple the evolution of low-level architectural details from high-level processing.
- Debugging: Manufacturers can provide software models of an architecture to aid in the development and debugging of P4 programs.

B. PROTOCOL INDEPENDENT SWITCH ARCHITECTURE

Protocol Independent Switch Architecture (PISA) [20] introduced a new generation of high-performance forwarding architectures that defined the need to program the data plane, which is the origin of P4. Following this architecture, the V1Model P4 switch architecture [21] can be used as an excellent example to review many of the main concepts.

As described in the abstract model defined by V1Model (see Fig. 3), the input packets are first handled by P4 programmable *Parser Stage*. The packet parser in P4 is defined as a state machine. Each state in the state machine extracts header fields (e.g. Ethernet, IPv4, TCP/UDP), which are declared in the P4 program and, depending on the field value, the state will change to another state.

After the packet is parsed, it passes through the P4 programmable *Ingress pipeline* where Match&Action tables are used. A table compares the matching fields in flow entries with packet headers or metadata to determine the appropriate actions (i.e. add/remove/change fields or passing the table without any action). These actions include the possibility of

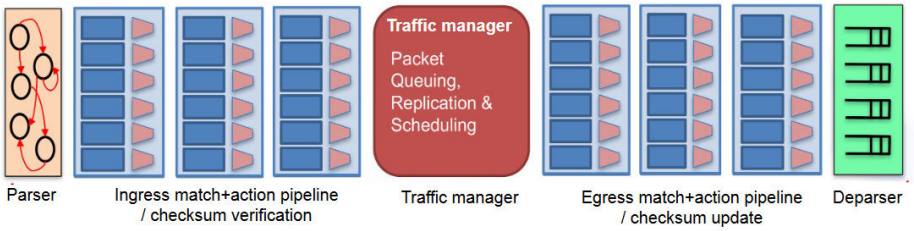

FIGURE 3. V1Model P4 switch Architecture.

TABLE 2. P4 switch Architectures.

PISA (Protocol-Independent Switch Architecture) [20] PISA defined the need for programming the data plane.
V1Model. Version 1.0 P4 Switch Architecture [21] V1 defines 4 programmable P4 blocks and 1 fixed-function block. Supported by BMv2 open-source software switch.
PSA (Portable Switch Architecture) [22] PSA is a switch architecture specification that may be implemented in any switch target. Community-developed architecture (P4.org Arch WG). It defines 6 programmable P4 blocks and 2 fixed-function blocks.
FlexSAI [23] FlexSAI is a hybrid programmable/fixed-function switch based on the Switch Abstraction Interface (SAI).
PNA (Portable NIC Architecture) [24] PNA is a Work in progress by the P4.org Arch WG.
SimpleSumeSwitch Architecture [25] Protocol-Independent Switch Architecture for NetFPGA SUME.

modifying the header fields of the current packet (stateless or transient information) and perform stateful operations that store state persistently, i.e. for more than a single packet. P4 allows the programmer to maintain stateful or persistent information in the form of counters (for counter events associated to entries in tables), meters (for data rate measurement) and registers (counters that can be operated from actions in a general way).

Between the Ingress pipeline and the Egress pipeline stages, the V1Model includes a non-p4 programmable block that represents the non-programmable *traffic manager* including fixed functions provided by target vendor. Next, the packets pass through the *Egress pipeline*, which is also defined as Match&Action tables for further modifications. Finally, the packets pass through the *Deparser* for serialization and then they exit the switch.

Table 2 shows a list of the existing P4-programmable switch architectures.

C. P4RUNTIME

P4Runtime is another component of the P4 ecosystem [19]. P4Runtime is defined as a standard, open and silicon-independent API to enable runtime-control of P4 forwarding planes. It is open (i.e. it can be used to control any switch ASIC), extensible (i.e. it is designed to make it easy to add new features over time), and customisable (i.e. different networks can use different protocols and features while still using the same API).

Although both the P4 language and the P4Runtime API include the term P4, they are different projects. The P4 language is used to define how a switch processes packets, specifying the switch pipeline. Meanwhile, P4Runtime is an API that is used to control fixed, semi-programmable or completely programmable switches whose behaviour has been specified in the P4 language.

III. IN-BAND NETWORK TELEMETRY (INT)

In-Band Network Telemetry (INT) is one of the main P4 applications that has been developed by the P4 Applications Working Group [26]–[29]. Based on the INT Dataplane Specification, different monitoring systems have been proposed in the last few years. In subsection III-A, we will describe the basic concepts of INT, and we will then review the most relevant INT-based monitoring systems in the literature.

A. DESCRIPTION AND TERMINOLOGY

As defined in the INT Dataplane Specification, In-Band Network Telemetry (INT) is a framework that is designed to allow the collection and reporting of network state by the data plane, without requiring intervention or work by the control plane. This 'in-band' feature offers the possibility to attach real-time network state to every packet at the line rate, which allows fine-grained monitoring [30].

INT packets are defined as packets that contain header fields (INT headers) that include instructions (INT instructions) interpreted by network devices. Following these instructions, network devices attach network information (INT metadata; e.g. switch id, ingress port id, hop latency, queue status or link utilisation) to the packets, and this information will be used to provide real-time, end-to-end network telemetry, with packet-level granularity.

An INT system consists of INT sources (entities that create and insert INT headers into packets and send them out), INT transit hops (network devices that add their own INT Metadata to the INT packets by following the INT Instructions in the INT header) and INT traffic sinks (network devices that extract the INT headers and collect the path state contained in the INT headers).

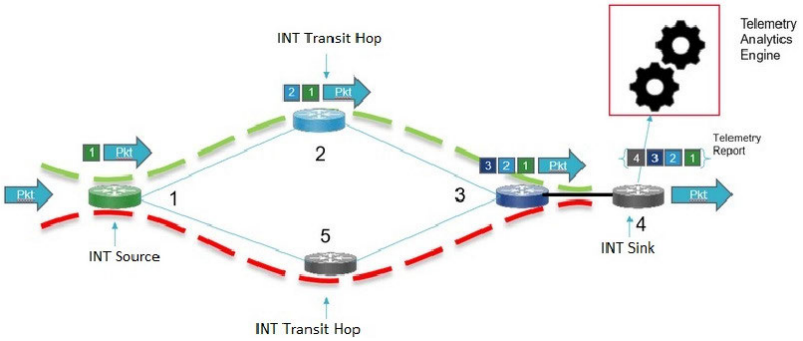

FIGURE 4. INT System. Example of INT-capable network.

TABLE 3. Summary of network-wide passive in-band telemetry solutions using P4.

	Proposals	Implementation details
INT-based solutions	IntMon [30]- [39]	INT over UDP; P4-14 language; BMv2 switches + ONOS controller KDN included in [36] [37] IntController defined in [38] [39]
	Pre-filtering at INT Sinks [40]	INT encapsulation not defined; P4-16 language; Netronome Agilio NFP-4000 SmartNIC; INT report pre-filtering algorithms implemented in P4 switches
	IntPath [41]	INT over UDP; P4-14 language; BMv2 switches in Mininet This uses Segment Routing (SR) and two path policies: depth-first search (DFS) and Eulier.
	Tracking Matched Rules [43]	INT over TCP; Real testbed: Edge-core Wedge100BF switches This enhances monitored information adding matched rule tracking This proposes two schemes to reduce the number of INT reports, implemented in P4 switches
	FS-INT [45]	INT over UDP; Hw/Sw implementation details not given This proposes two sampling strategies to reduce INT report overload
	PAINT [47]	INT over UDP; BMv2 switches + ONOS controller INT service module to improve SDN fault location system.
	ML-INT [48]	INT over UDP/TCP; IP-over-optical networks; P4-enabled SmartNICs and programmable Tofino ASIC in Barefoot switch This employs a rate-based sampling approach to include INT header into a selected portion of IP packets of a flow ElectricalOptical parameters are monitored by INT reports
	AM-PM based Monitoring [53]	This implements a Double Marking method and Time-Multiplexed Marking method for delay and packet loss measurements Evaluated using Mininet. Code available in [54]
	FlowStalker [55]	Mixed monitoring approach; Custom packet; BMv4 switches

INT-capable network elements are programmed to match on particular network flows and they then execute the programmed instructions on these flows. Thus, the INT packets are periodically generated by INT sources and injected into the network, where they will be queued and forwarded together with the background traffic. In each INT transit hop along the forwarding path, the INT packet will extract device-internal states and push them into the INT packet stack. At the last hop, the INT packet containing the end-to-end monitoring data will be sent to a centralised controller for further analysis.

Fig. 4 describes an INT capable network. Switch 1 (the INT source) inserts telemetry headers into the packet. Switches 2 and 3 (acting as INT Transit Hops) append telemetry data related to the green flow, and switches 5 and 3 (acting as INT Transit Hops) append telemetry data related to red flow. Switch 4 acts as the INT Sink for both flows.

Neither INT Dataplane Specification v0.5 [26] nor v1.0 [27] specify a specific INT header location (although potential encapsulations are described). An INT header could be inserted as an option or payload to any encapsulation type as long as the encapsulation header provides sufficient space to carry the INT information and that all INT switches agree on the location of the INT Headers. INT Dataplane Specification v2.0 [28] and v2.1 [29] specify four encapsulation formats, covering different scenarios, with and without network virtualisation: INT over IPv4/GRE (the INT headers are carried between the GRE header and the encapsulated GRE payload), INT over TCP/UDP (a shim header is inserted following TCP/UDP header, and INT Headers are carried

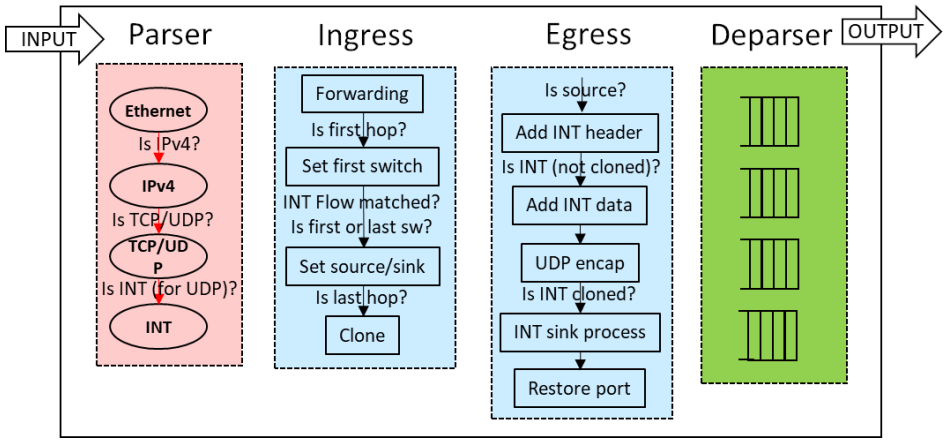

FIGURE 5. Processing Flow of an IntMon data plane. Adapted from [30].

between this shim header and TCP/UDP payload), INT over VXLAN (VXLAN generic protocol extensions are used to carry INT Headers between the VXLAN header and the encapsulated VXLAN payload) and INT over Geneve (using Geneve options).

INT over TCP/UDP are the most common encapsulation formats used in the INT-based telemetry systems found in the literature. As stated earlier, the INT Dataplane Specifications define the encapsulation format as an INT shim header, and the INT metadata header and INT metadata stack being encapsulated between the shim header and the TCP/UDP payload. In parallel to INT specification evolution, the telemetry report format has also been subject to changes, from v0.5 Nov. 17 [31] to the last version v2.0 June 20 [33].

The first prototype of the INT system was described in [34].

B. INTMON

IntMon [30] was proposed as an INT-based monitoring system implemented in SDN networks using Open Network Operating System (ONOS) controller. This solution uses the INT over UDP encapsulation format, defining the INT packet as a shim header in UDP packets. The INT header includes an INT port (to differentiate the INT UDP flows from others), an INT length field and a bit field is used to indicate if the INT packet must be sent to the controller or just forwarded. The INT data, after the INT header, will carry monitoring data.

Fig. 5 shows the processing flow of an INT-capable data plane in the proposed IntMon system. The Parser stage identifies the presence of INT packets in incoming packets. After packet parsing, the packets are fed to the Ingress pipeline for packet forwarding based on MatchAction tables. It is then determined whether the switch is source or sink, and the corresponding flags in the INT metadata are set. If the switch is a sink switch, then the packet is cloned. After completing Ingress pipeline, the packets are sent to Queues/Buffers and then sent to the Egress pipeline. Egress pipeline checks the

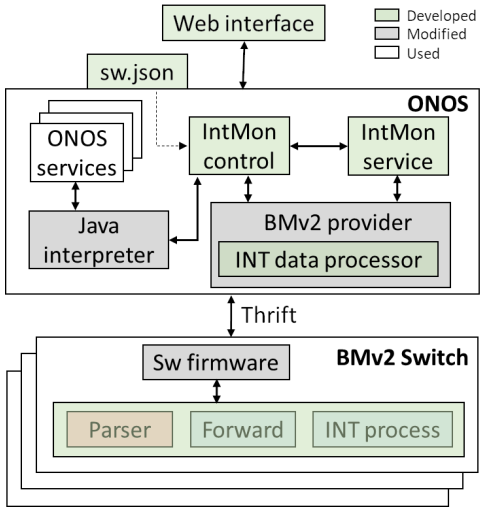

FIGURE 6. ONOS-based IntMon Monitoring Architecture. Adapted from [30].

source flag and adds an INT header if the packet matches a MatchAction table used to identify packets that are going to be monitored. Then, the INT information of the switch is attached to the end of the INT data. Finally, the packet is sent to the Deparser and sent out of the switch. If the switch is the last switch, then two packets are processed (the original and the cloned packets). One of them is sent to the controller, while the other suffers the INT sink process to remove the INT header and the INT data, and is then sent to the destination. Thus, monitoring tasks are transparent to end hosts.

The ONOS-based IntMon monitoring architecture is shown in Fig. 6. IntMon controller is the module that is in charge of populating the flow rules in the tables of INT network elements. The rules for setting up which flows and which fields to monitor are entered through a Web Interface. INT packets sent from INT sinks to ONOS controller, through the Thrift protocol, are processed at low-level layer of the ONOS core by the INT data processor. To reduce the huge

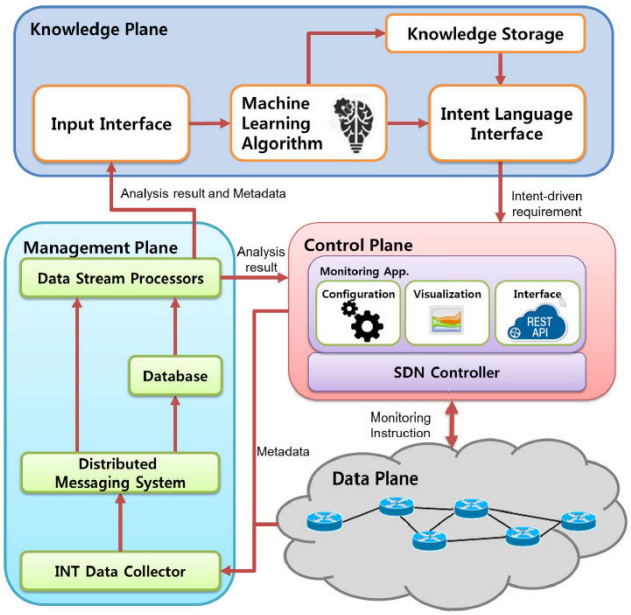

FIGURE 7. Knowledge-Defined networking using in-band network telemetry. From [36], [37].

amount of packet, the INT data processor decides if an INT packet must be analysed by the IntMon service or just discarded. In addition, IntMon service provides interfaces for other applications to offer real-time data.

The IntMon system was implemented using ONOS controller and BMv2 software switches [35] that are configured using P4 language, and evaluated using Mininet in a simple tree topology. The average CPU usage of the ONOS controller was analysed, concluding that the CPU usage increases linearly with the number of INT packets sent to the ONOS core. Bandwidth overhead for carrying INT data was also evaluated. The paper concludes that both CPU usage and bandwidth overhead are obstacles to useful and practical large-scale use of INT.

In an effort to improve the initial system, [36], [37] propose the use of IntMon architecture to define an INT monitoring system for SDN networks that includes Knowledge-Defined Networking (KDN). The new proposed architecture is composed of four planes: control and data planes, management plane and knowledge plane (see Fig. 7). The SDN controller works as control plane and deploys INT functionality to programmable network elements. It controls network monitoring using INT and converts requirements from the knowledge plane into specific network policies. In the data plane, INT metadata is generated, extracted and transmitted to the management plane. In the management plane, INT metadata is collected, stored and aggregated. The result contains a trajectory of packets, latency at each hop, queue occupancy and congestion status for each queue it passed through. This information is sent to the control plane (events that require immediate actions, such as link failures or loop detection) or to the knowledge plane (for actions such as resource planning, optimisation, performance management, and verification).

As pointed out earlier, although INT monitoring works directly at the data plane, high INT report rates in real-time and complex scenarios imply the need for a high-performance INT collector. To improve the INT-based telemetry system, the authors of IntMon propose a high-performance collector called INTCollector [38] that would replace the INT controller services implemented by ONOS controller. IntCollector defines two processing paths: a fast path and a normal path. The fast path processes every INT report, and therefore it requires a high packet processing rate. The normal path processes special events sent from the fast path (e.g. variation of a defined metric), and stores INT metric values in a database.

A complete and formal description of the INT monitoring architecture including INTCollector is published in [39].

To sum up, IntMon embeds telemetry data in real-time, covering the entire network. The knowledge plane offers to the management plane the intelligence to optimise the monitoring tasks, and the implementation of INT collector optimises the collector's resources. The extension of IntMon system to other encapsulation approaches will be necessary to evaluate the performance of the proposed monitoring system in current scenarios.

C. PREFILTERING APPROACH IN INT SINKS

Although INTCollector [38], [39] improves the INT controller performance, all INT reports have to be processed by the controller, which results in a very high load on it. With the aim of reducing this load, [40] proposes a programmable event detection mechanism, programmable in P4, that allows customisation of which events at the network elements are allowed to give rise to the generation of an INT report to be sent to the INT controller.

Through an SDN controller, the control plane configures the network element role (INT source, INT transit or INT sink). In the case of INT sources, they insert telemetry headers and an instruction bitmask to determine which telemetry items to collect. INT transits insert INT metadata (queue occupancy or hop latency) into the header. INT sinks receive the packets with telemetry, remove the INT headers and run the configured event detection algorithm. If the configured threshold is exceeded (end-to-end latency or queue occupancy threshold), then the INT report is sent to the INT controller; otherwise, it does not. This proposal also introduces the use of a Kafka cluster and an Elastic Search stack for processing the received telemetry reports, where advanced monitoring tasks can be executed (see Fig. 8). Thus, the SDN controller can use the obtained data to reconfigure and fine-tune the threshold and algorithm settings.

Two event detection algorithms are defined in this work: fast_detection and complex_detection. The algorithm and the related parameters are obtained by the INT sink device in the P4 egress block from a match-action table. When fast_detection is selected, three options are possible: per-flow, per-hop and moving average. In the per-flow case, an event is detected if the difference between the sum of the INT

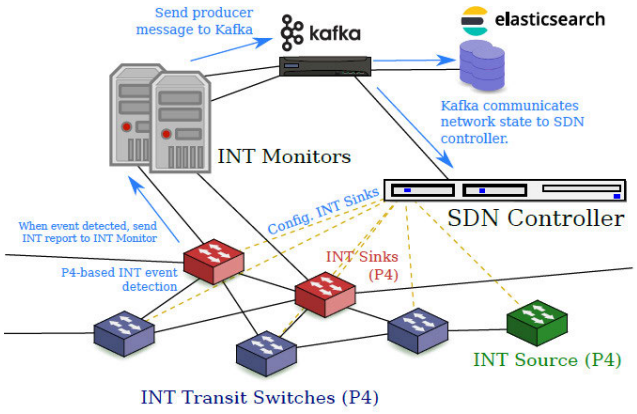

FIGURE 8. INT monitor system proposed in [40].

metadata for all hops in a specific flow and previous sums is greater than the threshold. In the per-hop case, the metadata is analysed for each switch id. If the difference between the incoming INT metadata and the previous metadata is greater than the threshold, then the new value is stored and an event is considered. The moving average is similar to the per-flow algorithm, but a simple exponential moving average between the current and the last received sum is performed. Meanwhile, a complex_detection algorithm allows the definition of complex logical operations.

The proposed event detection algorithms, which are implemented in P4_16 language, have been evaluated in a real testbed consisting of an INT sink equipped with a Netronome Agilio 2 × 40G NFP-4000 SmartNIC, an INT monitor and a NetFPGA-SUME running a traffic generator. The results show that the proposal is highly scalable reducing the processor load due to the pre-filtering in the switch data plane. While the INT report collector can process around 3 Mpps telemetry reports per core, using event pre-filtering increases the capacity by 10-15x.

In summary, this proposal is also able to embed telemetry data in real-time, covering the entire network (managed by the monitoring application). Considering telemetry traffic optimisation, event detection algorithms are defined to reduce telemetry report sent from INT sink. As far as resource optimisation is concern, the same event detection algorithms are also designed to optimise the controller load. The proposal of more complex event detection mechanisms and the performance evaluation on switches have been identified as new lines of work.

D. INT-PATH

INT-Path [41] proposed another INT-based network-wide traffic monitoring, including two new approaches. The first novelty of this proposal is the use of Segment Routing (SR) [42] to include the route that it must take through the network into the INT packet. This removes the need to populate the Match/Action tables of the network elements with forwarding rules. Meanwhile, focusing on obtaining a network-wide view, the INT-Path defines two path planning policies to generate multiple non-overlapped INT paths that cover the entire network.

INT-Path uses INT over UDP encapsulation to carry the SR-INT payload (see Fig. 9). The destination port is used to include the SR-INT port and, above the UDP header, a 512-bit field is reserved for the SR label stack. Above the SR label stack, a variable-length INT label stack is allocated. Each INT label occupies 22 bytes containing information such as device ID, ingress/egress port, egress queue depth. The destination IP (DIP) address of the probe packet is set using controller's IP to guarantee that the probe packet will finally be forwarded to the controller for further analysis.

From the forwarding behaviour, INT network elements perform three different functionalities. The INT source (called INT generator in the paper) is responsible of injecting the SR-INT probe packets into the network. Periodically, an INT generator receives 'empty' probes from an attached host, and it then rewrites the packet header to allocate the SR label stack and includes its local INT information. The SR label stack (that is, the packet route) is determined at the controller applying a centralised route calculation. The INT transit nodes (called INT forwarders in the paper) perform packet forwarding of SR-INT packets and background traffic. Finally, at the last hop of the monitoring path, the INT sink (called INT collector in the paper) forwards the probe packet to the controller for analysis.

As stated earlier, the INT-Path proposes two path planning policies. The first is based on depth-first search (DFS), which is straightforward but time-efficient. The second is an Euler trail-based algorithm that can optimally generate non-overlapped INT paths with a minimum path number.

INT-Path proposal was tested and evaluated using Mininet and BMv2 software switches. To achieve the capability of protocol independent header rewriting and forwarding of customised INT packets, which contain additional source routing metadata, the switches were programmed in P4. The two path planning policies were evaluated considering the number of generated INT paths and INT path length variance under different network topologies. The INT telemetry overhead was also evaluated, differentiating controller overhead caused by INT probe collection, and switches overhead measured as total number of INT probes processed by switches per second. The execution time of path planning algorithms and the impact of telemetry granularity were also studied.

The main novelty introduced by INTPath with respect to the previously described proposals is the use of SR to optimise the resource utilisation of the network elements. Again, only the INT over UDP encapsulation approach has been considered.

E. INT-BASED SYSTEM FOR TRACKING MATCHED RULES

The work presented in this section [43] defines an INT-based system to track the rules matches by the packets of a flow in real or past time. In this way, this proposal extends the network information that can be monitored using In-Band Network Telemetry (INT).

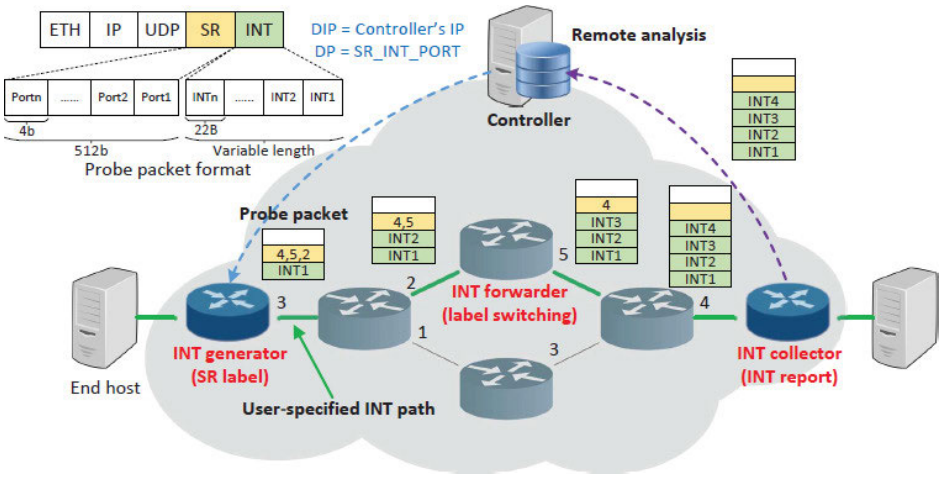

FIGURE 9. INT-Path monitoring system. From [41].

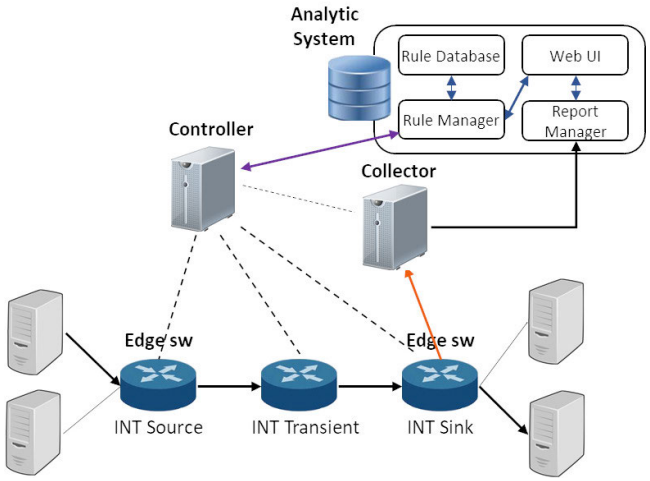

FIGURE 10. INT-based system for tracking matched rules. Adapted from [43].

The proposal implements INT over TCP encapsulation considering the telemetry report format specified by the last version (Telemetry Report Format v2.0 [33]). It defines a 16-bit metadata bitmap to indicate which optional metadata is present in the telemetry report header, where bit 1 to bit 8 indicate the network states that are included in the report (e.g. ingress and egress interface ID, hop latency, queue id and queue occupancy, etc.). This work proposes the use of the 9th bit, which is reserved in the specification, to indicate the matched rule tracking.

Fig. 10 shows the system architecture. Using the numeration generated by a Rule Manager, the controller assigns a globally unique ID (a 32-bit unsigned integer counter variable) to every rule that is installed in a programmable network element. The Rule Manager associates the rule ID with the switch ID and table ID, and stores the record in a Rule Database. In the P4-programmable network element, each rule is composed of the match keys, the action function to be invoked when the rule is matched, and the data that should be passed into the action function as its arguments. For tracking rules, the rule ID is one of these arguments, and the action function copies the value into the INT header of the packet.

With the aim of reducing the rate of INT reports, two INT report traffic reduction schemes are proposed: enumerable network state traffic reduction and threshold-based network state traffic reduction. The first scheme is applied when the network states to be monitored are enumerable (e.g. switch ID, output port ID and rule ID). The second scheme is applied if the monitored network states change all the time (e.g. timestamps and hop latency). In both cases, programmable network elements maintain a P4 register array where the network state last seen associated to a flow is stored. A flow is indexed in the array using a Flow ID, which is obtained as the hash value of 5-tuple information of packet's header.

By applying enumerable network state traffic reduction, when a switch receives an INT packet, it obtains the flow ID and the hash value of the INT metadata. If there is no difference between the obtained value and the stored value in the P4 register, then the network state has not changed and therefore there is no need to add the network state into the packet. The INT Sink receives the packet, but the INT report is not sent to the collector. In the case of threshold-based network state traffic reduction, the system employs a threshold to identify a change when the difference between the current network state and the previous stored value is greater than the threshold. In this case, the state is added into the INT header of the packet.

Both reduction schemes are evaluated in a real testbed consisting of three Edge-core Wedge100BF series switches [44]. It is demonstrated that both schemes save network bandwidth consumed by INT reports and reduce the processing requirements at the INT collector.

This proposal defines an INT-based monitoring system that allows the tracking of the rules that have been matched by the packets (in real-time and in the past). This monitoring information, that is not offered by other INT-based proposals,

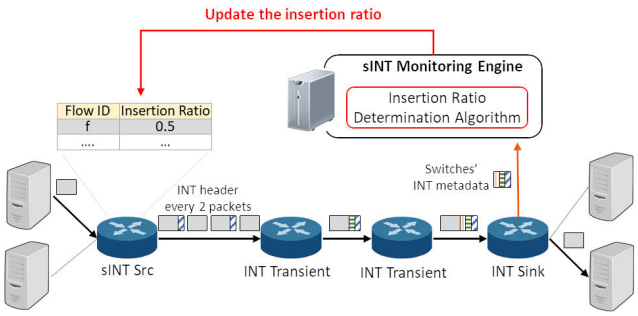

FIGURE 11. sINT architecture. Adapted from [46].

can be useful to detect unexpected behaviours related to network security problems.

F. FS-INT

Previous works [40] and [43] have proposed their respective methods to reduce the number of INT reports sent to the controller in the defined systems. Briefly, in the first case, the INT sink executes an event detection algorithm to determine if the INT report should be sent to the controller. In the second case, each INT device executes a traffic reduction scheme to determine if state information should be included in the INT header. Consequently, if the INT sink receives an INT packet without state information, then the INT report will not be sent to the controller.

In this section, Flexible Sampling-based INT (FS-INT) system [45] is presented, which includes a different approach to reduce the INT report traffic. This work is an extension of a previous work by the same authors, which is called the Selective INT (sINT) system [46] (see Fig. 11).

The basic idea of sINT is to limit the number of packets belonging to a monitored flow that will include INT information. Therefore, an sINT monitoring engine determines the ratio of INT header insertion associated to a certain flow, and it informs the sINT ingress node. The sINT ingress node maintains an sINT table where each entry consists of a flow ID (a hashed value of the 5-tuple: source IP, source port, destination IP, destination port, protocol type) and the corresponding insertion ratio. Thus, upon each packet's arrival, the sINT ingress switch extracts the flow ID from the 5-tuple hash function and checks the sINT table. If the matched entry exists, then the INT header will be inserted according to the corresponding insertion ratio. Then, the INT transit and sink switches insert the INT metadata only for incoming packets to which the INT headers are inserted.

As stated earlier, reference [45] extends the sINT and proposes a flexible sampling-based INT (FS-INT) system that includes two different sampling strategies: a rate-based strategy and an event-based strategy (see Fig. 12). Using the rate-based strategy (similarly to sINT), the INT source inserts the INT header into every R-th packet. If a packet includes the INT header, then all of the specified INT metadata are inserted at each hop. In contrast, using the event-based

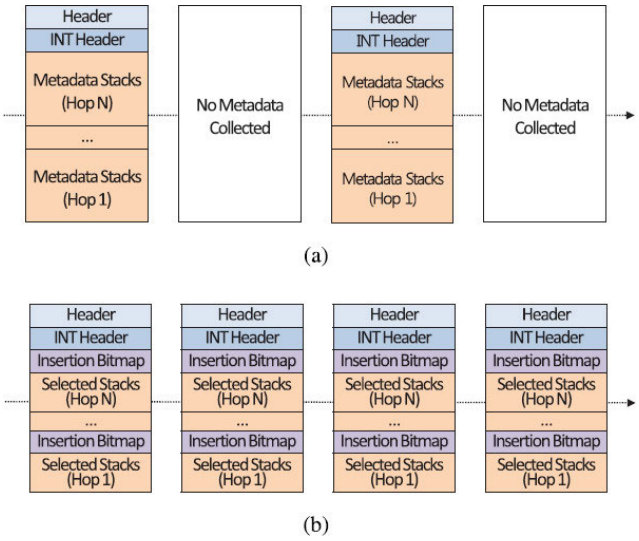

FIGURE 12. Sampling strategies defined in FS-INT: (a) rate-based sampling strategy; (b) event-based sampling strategy. From [45].

strategy, the INT source inserts the INT header into all the packets. For every incoming packet with INT headers, each INT transit node determines whether a sample value for the specified INT metadata type should be inserted based on a certain criterion (e.g. the queue length exceeds a prespecified threshold). Consequently, the number of values inserted at each hop can vary. For that reason, an insertion bitmap will also be added at the beginning of the metadata stack inserted by an INT node. This metadata bitmap is defined as a new INT metadata type. According to the specification [33], bit 1 to bit 8 in the instruction bitmap field are used to the standard INT metadata type, and the others bits (bit 9 to bit 14) are reserved for future use. In this proposal, bit 14 is used.

Both sampling strategies are evaluated and compared to a non-sampled INT approach using the INT over UDP encapsulation method. In FS-INT-rate-based, metadata is inserted for every R packets, and in FS-INT-event-based, metadata is inserted only when the difference between the last hop latency measurement and the current hop latency measurement exceeds a pre-defined deviation threshold. As expected, the protocol overhead in non-sampled INT approach is much higher than in the proposed FS-INT alternatives. In addition, the protocol overhead in the FS-INT-rate-based decreases linearly to the increase of R, whereas the protocol overhead in the FS-INT-event-based decreases as the threshold increases. In addition, accuracy is analysed comparing the measured average hop latency in the non-sampled INT to FS-INT-rate-based and FS-INT-event-based results. This work shows that the average hop latency measured in FS-INT-event-based is closer to non-sampled INT compared with FS-INT-rate-based.

Although the original work does not raise the possibility, in our opinion both strategies could be applied together: not all of the packets will include INT header, and those that include it will include event-based monitor information.

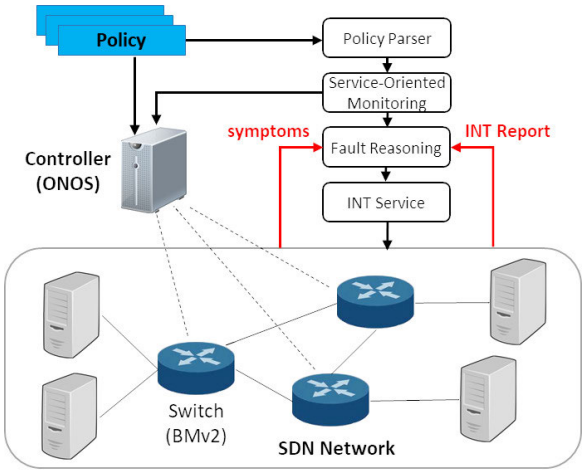

FIGURE 13. PAINT system Architecture. Adapted from [47].

G. PAINT

Policy-Aware In-band Network Telemetry (PAINT) [47] proposes the use of In-Band Network Telemetry (INT) framework as one of the blocks to deploy a policy-aware SDN fault localisation system, as shown in Fig. 13.

Network operators define network services using a high-level Service Provisioning Language (blue boxes in the figure). From the service policy, the SDN controller decides (and performs) how the service is deployed in the network, and the policy parser module parses the policy to identify a casual relationship between network elements and service-level symptoms that would allow the detection of service fault. Based on this model, the service-oriented monitoring module defines monitoring schemes to detect service-level symptoms, which will be submitted to the controller for deployment. When the symptoms are received, then the fault reasoning module tries to locate the root causes. If the received symptoms offers sufficient information to identify specific problematic network elements, the fault reasoning process can stop. Otherwise, the INT Service module is used to collect additional and relevant information directly from appropriate network elements.

The PAINT system is implemented in Python 2.7 and Erlang 16.2.1. Python is used to map the provisioned services to the corresponding monitoring schemes. Erlang is used as a backend engine for symptom-fault-telemetry model construction and distributed monitoring. The INT over UDP encapsulation scheme is used by the INT service module.

The PAINT system is evaluated by comparing its performance with the Symptom-Fault bipartite model based approach (in particular, the Risk Share model based approach) in two different network topologies (binary tree and fat tree topologies) simulated in Mininet, where BMv2 switches connected with ONOS controller through Thrift protocol are employed. Two parameters are considered to evaluate the accuracy of the PAINT system: the fault detection rate and the false positive ratio. The obtained fault detection rate is very high with an average value of 95% for different fault scenarios; and the false positive ratio is very low (3%).

PAINT -unlike the rest of proposals- does not work collecting INT-based monitoring information continuously. INT Service module is only used to collect punctual information. Therefore, the proposal of resource optimisation methods or telemetry traffic optimisation solutions have not been considered in this work.

H. MULTILAYER INT

The last INT-based contribution reviewed in this section focuses on IP-over-optical networks. Multi-Layer In-band Network Telemetry (ML-INT) [48] defines a P4-based flexible INT-based system that treats IP and optical levels jointly for monitoring and troubleshooting in real-time.

The architecture of an ML-INT system is shown in Fig. 14-(a). The IP layer consists of P4 network elements (two types are considered: Linux servers equipped with P4-enabled SmartNICs, and programmable ASIC switches) interconnected by lightpaths established in the optical layer. The optical layer consists on BV-WSS (Bandwidth-variable wavelength selective switches) and fiber lines. To implement ML-INT, an Optical Performance Monitor (OPM) is implemented on each BV-WSS.

Similarly to the approach previously introduced by sFlow, when an IP flow is involved in ML-INT monitoring tasks, not all of the packets will include INT header. The ingress switch selects a portion of the flow's packet according to a preset sampling rate and INT fields are inserted in them. Meanwhile, to limit the bandwidth overhead, only part of the statistics of all the electrical/optical network elements will be included in the INT packet along the path. The egress switch mirrors the INT packet to a data analyser for data analysis and storage, and converts the INT packet back to a regular one by removing the INT fields to be sent to the destination host.

Fig. 14-(b) shows the implementation details of the Data Analyser, a P4 network element and the OPM on each BV-WSS. The Optical INT module of the P4 switch receives from the Optical Channel Monitor (OCM) the lighpaths' optical parameters, and the Packet INT module collects the statistics of the flow in real-time. Both parameters (IP layer and Optical layer values) will be aggregated by the INT Agent. On the other hand, the Packet Processing Pipelines defined in the P4 switch will be used to define: 1) the sampling rate for INT insertion, 2) the maximum number of INT fields that can be inserted in an INT packet, 3) the electrical/optical network elements that are selected for statistic collection, and 4) the required statistics of each selected network element.

In ML-INT system, the INT over TCP/UDP encapsulation method is adopted. The INT header consists of an INT Info field and an INT data stack that includes a series of INT Fields, each one corresponds to a hop on the routing path. ML-INT limits the number of statistics that can be included in an INT field to its smallest value (in particular, two) to avoid long INT headers. One of these two values must be the switch ID, and the other can be selected from all the supported

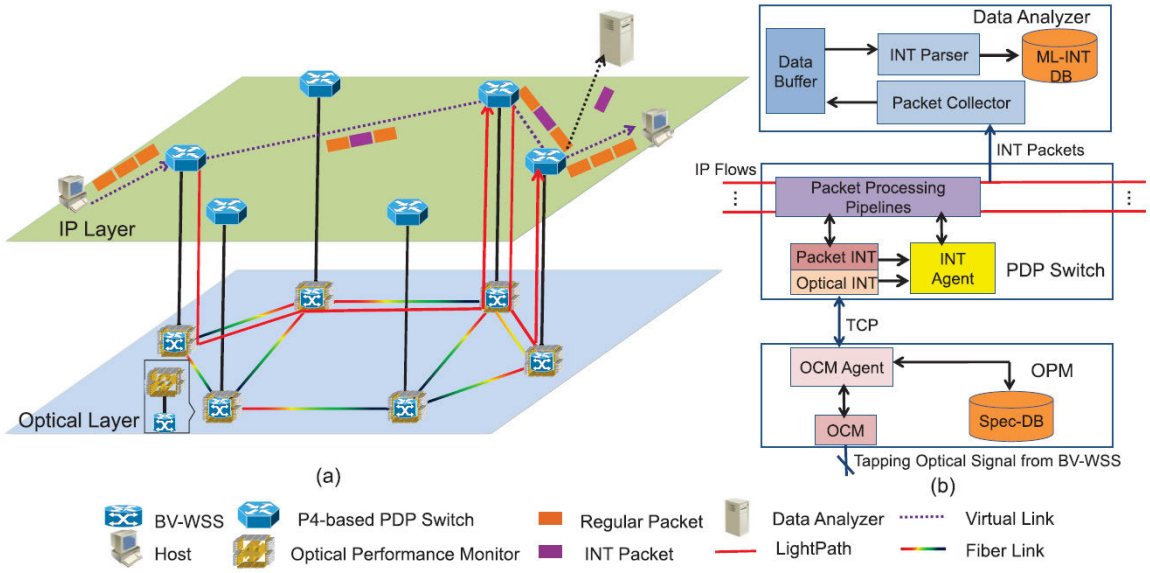

FIGURE 14. (a) System architecture of ML-INT system. (b) Details inside the data analyser, P4-based Programmable-Data-Plane (PDP) switch and Optical Performance Monitor (OPM). From [48].

statistics of the electrical/optical network element. ML-INT proposes that if more than two values are required, then the system distributed them in different INT packets.

Fig. 15 describes the packet processing pipelines implemented in (a) INT ingress (source) switches, (b) INT transient switches and (c) INT egress (sink) switches. Common modules are the INT Arbiter and INT Selection modules. The INT Arbiter determines the sampling rate of the selective insertion and the type of statistics to be encoded in the INT Field, which are the statistics that will be given by the INT Agent. The INT Selection module obtains tokens from the INT Arbiter to execute the INT header insertion.

The ML-INT system implementation is validated in small-scale real IP-over-optical network testbed. In addition, three use-cases are considered to demonstrate the usefulness of the proposed system. In a further work [49], the ML-INT system is expanded to integrate AI-assisted data analytic to process the obtained telemetry data for network monitoring and troubleshooting.

IV. AM-PM NETWORK TELEMETRY

As described in the previous section, In-Band Network Telemetry (INT) is a P4-based application that was designed by the P4 alliance to allow the collection and reporting of network state directly by the data plane. To that end, the INT framework makes use of P4 language to program programmable data plane (PDP) network elements. However, the potential of the P4 language and PDP switches enables implementation of other types of solutions that are not based on the INT standard and which also offer in-band telemetry solutions.

One of these in-band telemetry approaches is Alternate Marking-Performance Measurement (AM-PM) [50], [51]. AM-PM is an efficient method that measures network flows by using a very small number of bits in each data packet: one or two bits, or even as few as zero bits per packet. In this section, we will review this recent proposal.

A. DESCRIPTION AND TERMINOLOGY

AM-PM monitoring action is executed between two or more Measurement Points (MPs) in the network, which can either be end-hosts or network devices. The basis of AM-PM is the use of marking bits, which are used for signalling and coordinating the measurement between the measurement points. The AM-PM method, which was documented and published as an RFC by the IETF [51], is considered in the context of various encapsulations, including Geneve, SFC NSH, BIER, MPLS, and QUIC. AM-PM can also be deployed over IPv4 or IPv6 by using reserved values of fields in the IP header.

Alternate marking offers a method to measure packet loss, packet delay, and packet delay variation. Consequently, AM-PM makes use of the marking bits and two basic abstractions: step detection and pulse detection. As shown in Fig. 16, a step is detected when the value of the marking bit is toggled, and remains at the new value in the following packets; and a pulse is detected when the value of the marking bit is toggled in a single packet, and returns to its previous value in the following packets.

Step marking can be used to define loss measurement solutions. At the initiating MP, the inclusion of a marking bit (usually called color indication) in the header of every packet allows the division of the traffic into equal length blocks of data. The reception and count of consecutive packets marked with a certain value at the terminating MP allows the detection of losses comparing the counters at a central controller. Step marking can also use to define

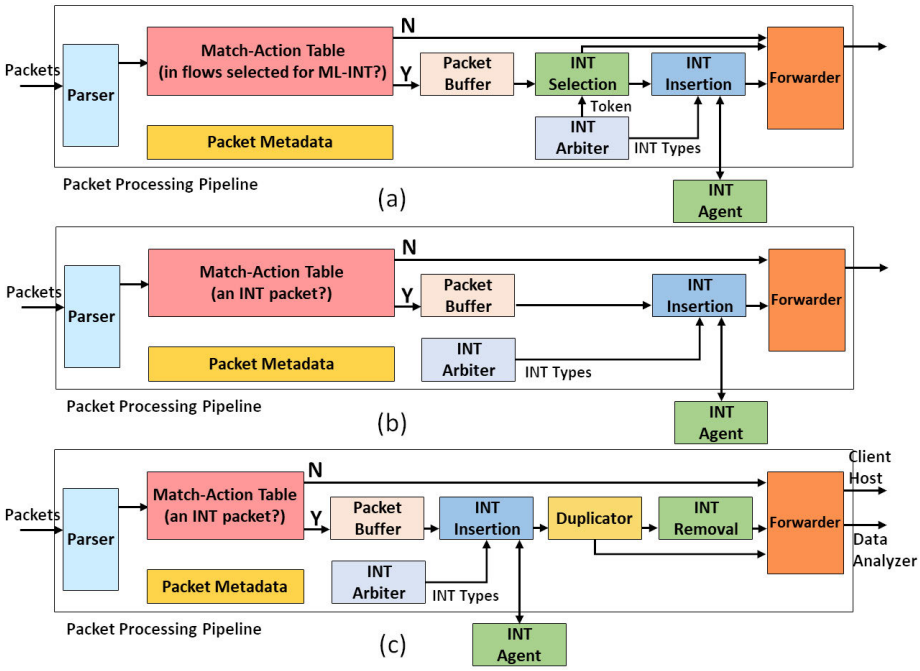

FIGURE 15. ML-INT. Packet processing pipeline in (a) an INT ingress switch, (b) and transient switch and (c) an egress switch. From [48].

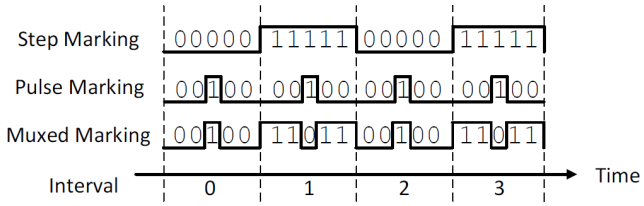

FIGURE 16. AM-PM marking bits. From [53].

the single-marking methodology to measure one-way delay. Whenever the color changes (which means that a new block has started), a network device can store the timestamp of the first packet of the new block. That timestamp can then be compared with the timestamp of the same packet on a second router to compute packet delay.

Step marking in coordination with pulse marking can be used to define the double-marking methodology that allows the implementation of fuller delay measurement solutions. The pulse bit is used to mark specific packet that are used by MPs to obtain delay/jitter measurements. This method offers to the central controller more information about delay.

These basic measurement methods are extended in [52], where additional methods based on single-bit or two-bit marking are defined.

B. AM-PM BASED NETWORK TELEMETRY IMPLEMENTATION IN P4

An AM-PM based time-multiplexed parsing approach that enables a practical and accurate implementation of AM-PM methodology in network devices in presented in [53]. This work details a P4-based implementation of AM-PM double marking method in P4 network elements.

As described earlier, to develop the AM-PM double marking method, it is necessary to implement a step marking and a pulse marking solution.

Starting with the first, step marking implemented in this solution requires the identification of periodic time range at network elements. Therefore, this proposal makes use of time-bit-as-a match criterion in the match-action lookup. Fortunately, this programming abstraction is provided by P4 as the packet's ingress time is included in the P4 metadata, which allows the timestamp to be used in ternary matches. For example, if the timestamp consists of two fields (a Seconds field and a Second Fraction field), then the least significant bit of the Second field can be used to determine periodic 1 second intervals defining a ternary match rule that considers this TimeBit and masks the rest of the bits.

In this case, the match-action rules at the initiating MP to implement step marking are indicated in Table 4a and the match-action rules at the terminating MP are shown in Table 4b.

For pulse marking, which is implemented in this proposal, it is necessary for the initiating MP to detect the first packet of the interval to be able to mark just one packet at a certain point of the interval. To implement this, the TimeBit and a register that stores the previous value of the TimeBit are used.

The match-action rules at the initiating MP to mark the first packet at the beginning of each interval and store the

TABLE 4. Step marking.

Match	Action
TimeBit=0; TimeBit=1;	MarkBit=0; counter0; MarkBit=1; counter1;

(a) Match-action table at the initiating MP

Match	Action
Mark=0; Mark=1;	counter0; counter1;

(b) Match-action table at the terminating MP

TABLE 5. Pulse marking.

Match	Action
TimeBit=1; register=0; TimeBit=1; register=1; TimeBit=0; register=1; TimeBit=0; register=0;	MarkBit=1; register=1; timestamp; MarkBit=0; register=1; MarkBit=1; register=0; timestamp; MarkBit=0; register=0;

(a) Match-action table at the initiating MP

Match	Action
MarkBit=1;	timestamp;

(b) Match-action table at the terminating MP

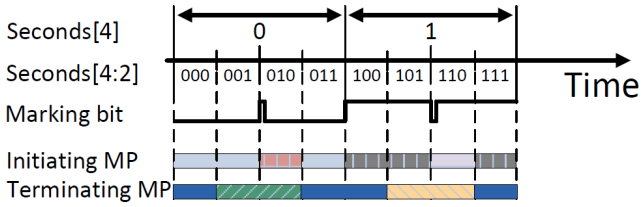

FIGURE 17. Time-muxed parsing. From [53].

timestamp are shown in Table 5a. The match-action rules at the terminating MP are simpler, as shown in Table 5b.

In addition to the implementation of the AM-PM double marking method using two-bit marking, this work defines a multiplexed (or time-muxed) marking method (see Fig. 17). The advantage of time-muxed marking is that it requires just a single bit per packet on the wire, while providing the same measurement resolution and accuracy as the double marking approach. This technique divides time into slots, and defines a meaning of the value of the marking bit depending on the time slot.

As an example, and due to software implementation restrictions, the implementation in [53] considers intervals of 16 seconds. Therefore, as can be seen in Fig. 17, the 5-th bit of the Seconds field (or Seconds[4]) defines 16-seconds intervals. In turn, each interval is divided into four sub-intervals using the following two bits (Seconds[4:2]). At the initiating MP, a pulse bit is located at the middle of the 16-seconds interval. TimeBits=010 or TimeBits=110 and the value of the register are used to detect that point using adequate match-action rules. At terminating MP, the second and third quarter of each interval are used to detect a pulse, whereas the rest of the time is used for detecting steps. To implement this method in p4 network elements, Table 6a and Table 6b show

TABLE 6. Muxed marking.

Match	Action
TimeBits=010; Register=0;	MarkBit=1; Register=1; counter0; timestamp;
TimeBits=010; Register=1;	MarkBit=0; Register=1; counter0;
TimeBits=0**; Register=*;	MarkBit=0; Register=0; counter0;
TimeBits=110; Register=1;	MarkBit=0; Register=0; counter1; timestamp;
TimeBits=110; Register=0;	MarkBit=1; Register=0; counter1;
TimeBits=1**; Register=*;	MarkBit=1; Register=1; counter1;

(a) Match-action table at the initiating MP

Match	Action
TimeBits=001;MarkBit=1;	counter0; timestamp;
TimeBits=010;MarkBit=1;	counter0; timestamp;
TimeBits=101;MarkBit=0;	counter1; timestamp;
TimeBits=110;MarkBit=0;	counter1; timestamp;
TimeBits=***;MarkBit=0;	counter0;
TimeBits=***;MarkBit=1;	counter1;

(b) Match-action table at the terminating MP

the match-actions at the initiating MP and the terminating MP, respectively.

The evaluation of the P4 software implementation was performed in Mininet and the code is available in [54]. Loss and delay were synthetically configured in the Mininet environment to compare the measured performance and the expected performance. The delay measurement error was on the order of 1 millisecond compared to the anticipated delay, and the loss measurement error was on the order of 10%. These results are reasonable given the software-based emulation environment that runs on a single machine.

As described in this section, AM-PM does not embed flow related metadata (e.g. switch-level information, ingress and egress information, buffer information) into packets, as it does INT-based solutions. Instead, AM-PM defines efficient monitoring methods using only a few bits in each packet for not wasting bandwidth. However, it is important to point out that AM-PM is a compatible and complementary method that can be used along with INT-based solutions. The implementation and evaluation of a monitoring system that integrates both approaches is an unsolved and interesting task.

V. FLOWSTALKER

To conclude this review, we will describe FlowStalker [55], a monitoring system that makes use of the data plane programmability offered by P4. Although this work has been included here, FlowStalker is not a passive in-band monitoring approach but is instead a mixed solution. As was described earlier, FlowStalker monitors real traffic flows, but also requires the injection of some packets from the controller (out-band traffic). These packets, called Crawler Packets (CPs), will only be processed by network elements in the data plane to obtain network element information (in-band telemetry).

The proposed monitoring scheme consists of two phases: a proactive and lightweight stage, and a reactive and

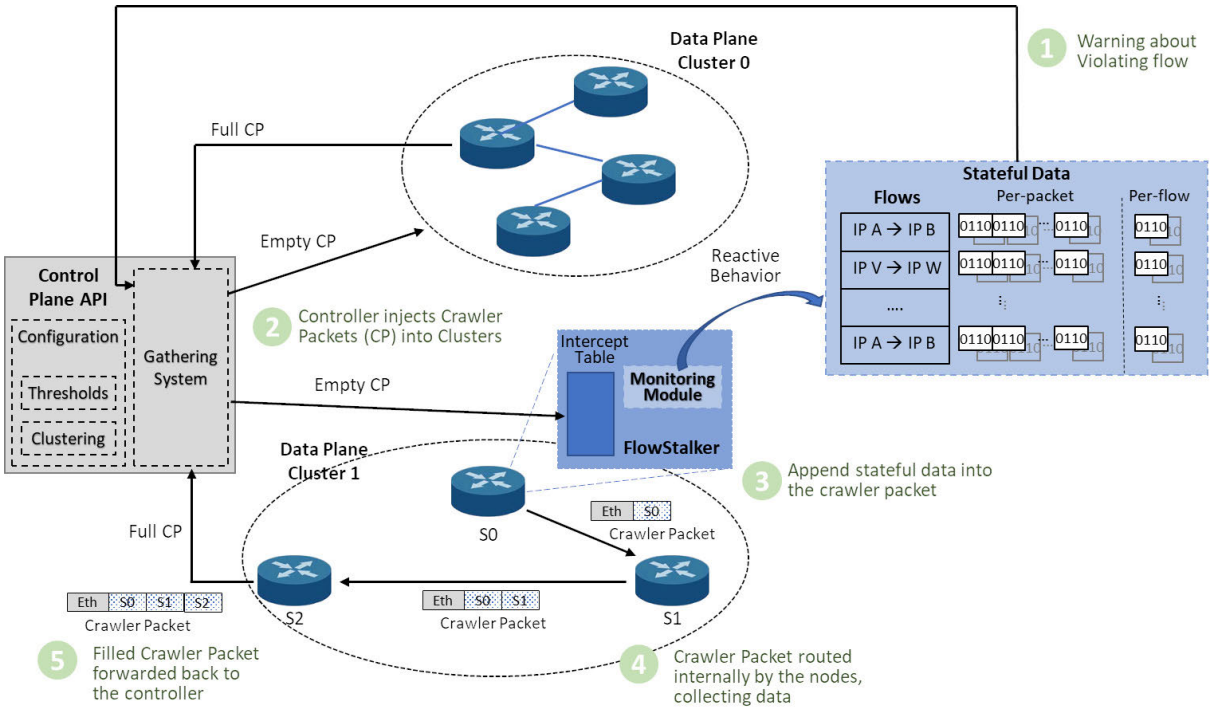

FIGURE 18. FlowStalker System. Steps 1 to 5 in the figure describe the steps after the monitoring warning of a flow goes off. From [55].

heavyweight stage. On the one hand, the proactive and lightweight phase is in charge of detecting target flows based on heavy hitter detection strategies. On the other hand, the reactive and heavyweight phase will be used to capture metadata (per-packet and per-flow metrics) about these detected
flows.

In the proactive phase, each switch intercepts all the incoming packets of each flow, increments a packet counter, and classifies a flow as a target flow when the number of received packets is higher than a certain threshold. At this point, the reactive phase is activated for the identified target flows, during which they are monitored by the switch. When a defined threshold associated to a metric of interest of a target flow is exceeded, the switch sends a 'warning' to the controller informing it of the violation (step 1 in Fig. 18).

To store the flow metrics, each switch maintains a hash table composed of an array of P4 registers where the lookup key is the source and destination IP address of a flow. Due to the limited length of the P4 data structures, the monitored per-flow and per-packet metrics are space-efficiently encoded.

As a result of the warning notification, the controller injects an empty Crawler Packet through the control path in a specific and predefined entry point in the network (or cluster) (step 2 in Fig. 18). Then, the Crawler Packet is routed through the data plane in the network (or cluster), appending stateful data into the Crawler Packet hop-by-hop (step 3 and step 4 in Fig. 18) until the Crawler Packet reaches a predefined

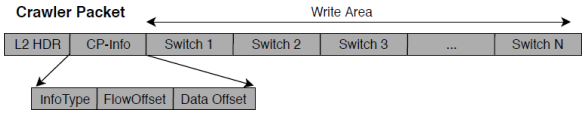

FIGURE 19. FlowStalker. Structure of a crawler packet.

exit point for that flow, and it is then forwarded back to the controller (step 5 in Fig. 18).

As mentioned earlier, FlowStalker defines an aggregated data plane information gathering strategy based on the partitioning of the network into clusters (based on the closeness of nodes). The definition of logical groups enables a quick and efficient cluster data collection. Fig. 18 described a FlowStalker system where two clusters have been defined.

The structure of a Crawler Packet is shown in Fig. 19. A Crawler Packet consists of a Layer 2 header, a CP header that stores control information for the switches, and a read/write area to be filled by the switches within the cluster. This read/write area is divided into equal chunks, and each fragment will contain information of a single switch. The type of the measurement is specified by the InfoType field, avoiding mixing measurements.

FlowStalker has been implemented in BMv2 P4 software switches and it has been evaluated considering a ring topology. Two experiments were performed to compare the communication overhead, defined as the total number of bytes exchanged through the control path, using FlowStalker system and the BMv2 Control Plane API that pools every single switch in the topology; and to evaluate the

TABLE 7. Comparison of the solutions considering six features that define the in-band network systems.

Proposals	Embedding telemetry metadata	Real-time	Fine-grained	Network Coverage	Telemetry traffic optimisation	Resource optimisation	Comments
IntMon [30]- [39]	✓	✓	✓	• Total network coverage, managed by the monitoring application	✗	• INT collector is defined to improve collector performance	
Pre-filtering at INT Sinks [40]	✓	✓	✓	• Total network coverage, managed by the monitoring application	• Event detection algorithms are defined to reduce telemetry report sent from INT sink	• Event detection algorithms optimise the controller load	
IntPath [41]	✓	✓	✓	• Path planning policies are proposed to optimise the number of monitored network elements	• Path planning policies are proposed to optimise the number of monitored network elements	• Use of Segment Routing to reduce Match/Action tables in the network elements • Optimised monitored paths to reduce the controller load	• Required Segment Routing support
Tracking Matched Rules [43]	✓• It extends the INT spec. defining Rule Matched (see comments)	✓	✓	• Total network coverage, managed by the monitoring application	• Two traffic reduction schemes are defined to reduce network bandwidth consumption	• 'Empty' INT reports are sent to the controller, reducing the processing load	• Use of P4 registers to maintain past state information • Bit 9 in the Instruction Bitmap Field is used
FS-INT [45]	✓• It extends the INT spec. defining Insertion (see comments)	✓	✓	• Total network coverage, managed by the monitoring application	• Flexible Sampling: Only a fraction of packets of a monitored flow include INT information (rate-based and event-based strategies)	• Rate-based strategy reduces the controller load	• Bit 14 in the Instruction Bitmap Field is used
PAINT [47]	✓	✗ INT module is used to collect punctual info	✓	• Punctual information required by the Fault Reasoning Module	✗	✗	
ML-INT [48]	✓• It extends the INT spec. to include optical layer values	✓	✓	• Total network coverage, managed by the monitoring application	• Sampling strategy: Not all the flow packets will include INT header	✗	• IP-over-optical network monitoring proposal
AM-PM based Monitoring [53]	✗	✓	✗	• Total network coverage, managed by the monitoring application	• Efficient method: Only one or two bits in each packet are required	✗	• It is compatible with INT-based monitoring solutions
FlowStalker [55]	✓• Crawler Packets (CP) are defined	✓	✓	• Clustered solution	• Use of threshold to identify in-band monitored flows • Only a fraction of the target flows initialise the in-band monitoring phase (using CP)	• Use of threshold to identify in-band monitored flows	

end-to-end delay of crawler packets depending on the cluster size.

As the authors indicates, the FlowStalker proposal is mainly focused on the definition of the data plane. Thus, the definition and implementation of a more complex control plane (e.g. using KDN) which uses the monitored data provided by the control plane is a pending task.

VI. COMPARISON

The state-of-art of network-wide passive in-band telemetry approaches is an open issue thanks to the emergence of data plane programmability. This survey reviews the main works that have been proposed to date.

Table 7 lists the recent contributions described in this article summarising how each fulfils different features of in-band telemetry, which we have identified from [56]:

- Embedding telemetry data: information obtained by network elements is sent in the body of packets to be used by the monitoring application.
- Real-time: a telemetry system provides real-time network status to feed controllers and users with monitoring information.
- Fine-grained: a telemetry system provides fine-grained network information (e.g. queue length, link utilisation, etc.)
- Network Coverage: a telemetry system covers the full network topology or just part.
- Telemetry traffic optimisation: a telemetry system optimises the bandwidth consumption due to telemetry overhead to scale solutions to large amount of traffic and network sizes.
- Resource optimisation: network elements and telemetry controller requirements to respond to large amount of traffic and network sizes.

VII. CONCLUSION AND FUTURE WORK

The main objective of a network telemetry system is to collect data to report information of the network in a real-time and fine-grained way to operators and OAM (Operation, Administration & Maintenance) applications. With the inclusion of SDN, network management solutions have changed from human operator based systems to programmable and automated systems. Therefore, the integration of an in-band telemetry system with existing network monitoring and management systems, or with particular applications (e.g. network anomaly detection, heavy hitter detection, NFV Service Function Chain planning and monitoring, etc.) is a promising avenue for further research work.

Another interesting line of work is focused on the adaptation or extension of in-band telemetry systems to wireless sensors networks (WSNs). For example, recent work [57] proposes a novel monitoring telemetry solution based on In-Band Network Telemetry (INT) for 6TiSCH (IPv6 over the TSCH mode of IEEE 802.15.4e) networks [58], [59], where TSCH stands for Time-Slotted Channel Hopping.

As summarised in Table 3, most of the reviewed in-band telemetry proposals rely on UDP/TCP encapsulation. Although other options are possible (e.g. GRE, Generic Routing Encapsulation; VXLAN, Virtual eXtensive LAN encapsulation; or GENEVE, GEneric NEtwork Virtualisation Encapsulation), the performance evaluation of in-band telemetry system considering overlay networks using encapsulation protocols has not been studied.

Closely related to the in-band encapsulation, the space required for the inclusion of monitoring actions and/or data into the packets is in conflict with the MTU (Maximum Transmission Unit) value of links. From INT specification v1.0, two different ways to address this problem are proposed: the first recommends the adequate configuration of the MTU of links between INT sources and sinks; the second proposes the use of dynamic discovery of Path MTU for flows being monitored. It would be interesting to evaluate both proposed alternatives and also the impact of IPv4 fragmentation on applications.

Finally, linked to any in-band network telemetry solution, using P4 framework or not, the In-Band Network Telemetry Orchestration (INTO) problem is currently under study. This problem consists in obtaining the most efficient and effective set of network elements to be monitored to provide full monitoring coverage while minimising the overhead. In particular, works such as [60]–[62] propose heuristic solutions to this challenging problem, which will undoubtedly and directly affect the design of in-band telemetry systems. This problem will be extremely important in Internet of Things (IoT) networks, which will require millions of traffic flows between IoT devices to be monitored.

REFERENCES

[1] T. Mizrahi, N. Sprecher, E. Bellagamba, and Y. Weingarten, *An Overview of Operations, Administration, and Maintenance (OAM) Tools*, document RFC 7276, IETF, 2014, doi: 10.17487/RFC7276.

[2] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3rd ed. Boston, MA, USA: Addison-Wesley, 1998.

[3] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, *Network Configuration Protocol (NETCONF)*, document RFC 6241, IETF, 2011, doi: 10.17487/RFC6241.

[4] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *Proc. HPSR*, Bucharest, Romania, 2018, pp. 1–7.

[5] S. Han, S. Jang, H. Choi, H. Lee, and S. Pack, "Virtualization in programmable data plane: A survey and open challenges," *IEEE Open J. Commun. Soc.*, vol. 1, pp. 527–534, 2020, doi: 10.1109/OJCOMS.2020.2990182.

[6] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, New York, NY, USA, 2013, pp. 127–132.

[7] P. Bosshart, G. Varghese, D. Walker, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, and A. Vahdat, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[8] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond OpenFlow: Opportunities and challenges for network and service operations and management," *J. Netw. Syst. Manage.*, vol. 25, no. 4, pp. 784–818, Oct. 2017.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[10] OpenFlow Specifications. *The Open Networking Foundation (ONF)*. Accessed: Jan. 28, 2020. [Online]. Available: https://opennetworking.org/software-defined-standards/specifications/

[11] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *J. Opt. Commun. Netw.*, vol. 11, pp. 84–95, Jan. 2019.

[12] *P4 Language and Related Specifications*. Accessed: Jan. 28, 2020. [Online]. Available: https://p4.org/specs/

[13] M. Budiu. *Programming Network With P4*. Accessed: Jan. 28, 2020. [Online]. Available: https://volansys.com/p4-programming-networks-forwarding-plane/
[14] *P4c*. Accessed: Jan. 28, 2020. [Online]. Available: https://github.com/p4lang/p4c
[15] *The P4 Language Specification Version 1.0.5*. The P4 Language Consortium. Accessed: Nov. 2018. [Online]. Available: https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf
[16] *The $p4_{16}$ Language Specification Version 1.0.0*. The P4 Language Consortium. Accessed: May 2017. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf
[17] H. Stubbe, "P4 compiler & interpreter: A survey," in *Proc. Seminars FI/IITM*, Munich, Germany, 2017, pp. 1–72.
[18] M. Budiu. *Programming Networks With P4*. Accessed: Jan. 28, 2020. [Online]. Available: https://blogs.vmware.com/research/2017/04/07/programming-networks-p4
[19] *P4Runtime Specification. Version 1.2.0*. The P4.org API Working Group. Accessed: Jul. 2020. [Online]. Available: https://p4.org/p4runtime/spec/master/P4Runtime-Spec.html
[20] P. Bosshart, G. Gibb, H. S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 99–110, Aug. 2013.
[21] *Version 1.0 Switch Architecture Model*. Accessed: Jan. 28, 2020. [Online]. Available: https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4
[22] *$P4_{16}$ Portable Switch Architecture (PSA)*. The P4.org Architecture Working Group. Accessed: Jul. 2020. [Online]. Available: https://p4.org/p4-spec/docs/PSA.html
[23] *FlexSAI*. Accessed: Jan. 28, 2020. [Online]. Available: https://github.com/opencomputeproject/SAI/tree/master/flexsai/p4
[24] *Portable NIC Architecture*. Accessed: Jan. 28, 2020. [Online]. Available: https://github.com/p4lang/pna
[25] *P4—NetFPGA: A Low-Cost Solution for Testing P4 Programs in Hardware*. Accessed: Jan. 28, 2020. [Online]. Available: https://p4.org/p4/p4-netfpga-a-low-cost-solution-for-testing-p4-programs-in-hardware.html
[26] *In-band Network Telemetry (INT) Dataplane Specification. Version 0.5*. The P4.org Applications Working Group. Accessed: Dec. 2017. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v0_5.pdf
[27] *In-Band Network Telemetry (INT) Dataplane Specification. Version 1.0*. The P4.org Applications Working Group. Accessed: Apr. 2018. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v1_0.pdf
[28] *In-Band Network Telemetry (INT) Dataplane Specification. Version 2.0*. The P4.org Applications Working Group. Accessed: Feb. 2020. [Online]. Available:https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_0.pdf
[29] *In-band Network Telemetry (INT) Dataplane Specification. Version 2.1*. The P4.org Applications Working Group. Accessed: May 2020. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf
[30] N. Van Tu, J. Hyun, and J. W.-K. Hong, "Towards ONOS-based SDN monitoring using in-band network telemetry," in *Proc. 19th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Seoul, South Korea, Sep. 2017, pp. 76–81.
[31] *Telemetry Report Format Specification. Version 0.5*. The P4.org Applications Working Group. Accessed: Apr. 2017. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v0_5.pdf
[32] *Telemetry Report Format Specification. Version 1.0*. The P4.org Applications Working Group. Accessed: Apr. 2018. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v1_0.pdf
[33] *Telemetry Report Format Specification. Version 2.0*. The P4.org Applications Working Group. Accessed: Jun. 2020. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v2_0.pdf
[34] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *Proc. ACM SOSR*, London, U.K., 2015.
[35] *Behavioral Model (BMV2)*. Accessed: Jan. 28, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model
[36] J. Hyun, N. Van Tu, and J. W.-K. Hong, "Towards knowledge-defined networking using in-band network telemetry," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp. (NOMS)*, Taipei, Taiwan, Apr. 2018, pp. 1–7.
[37] J. Hyun and J. W. Hong, "Knowledge-defined networking using in-band network telemetry," in *Proc. APNOMS*, Seoul, South Korea, 2017, pp. 54–57.
[38] N. V. Tu, J. Hyun, G. Y. Kim, J. Yoo, and J. W. Hong, "INTCollector: A high-performance collector for in-band network telemetry," In *Proc. CNSM*, Rome, Italy, 2018, pp. 10–18.
[39] J. Hyun, V. T. Nguyen, J.-H. Yoo, and J. W.-K. Hong, "Real-time and fine-grained network monitoring using in-band network telemetry," *Int. J. Netw. Manag*, vol. 29, no. 6, Oct. 2019, Art. no. e2080.
[40] J. Vestin, A. Kassler, D. Bhamare, K.-J. Grinnemo, J.-O. Andersson, and G. Pongracz, "Programmable event detection for in-band network telemetry," in *Proc. IEEE 8th Int. Conf. Cloud Netw. (CloudNet)*, Coimbra, Portugal, Nov. 2019, pp. 1–6.
[41] 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 *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Paris, France, May 2019, pp. 487–495.
[42] C. A. Sunshine, "Source routing in computer networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 7, no. 1, pp. 29–33, 1977.
[43] S. Wang, Y. Chen, J. Li, H. Hu, J. Tsai, and Y. Lin, "A bandwidth-efficient INT system for tracking the rules matched by the packets of a flow," In *Proc. GLOBECOM*, Waikoloa, HI, USA, 2019, pp. 1–6.
[44] *Edge-Core Wedge100BF Series Switches*. Accessed: Jan. 28, 2020. [Online]. Available: https://www.edgecore.com/
[45] D. Suh, S. Jang, S. Han, S. Pack, and X. Wang, "Flexible sampling-based in-band network telemetry in programmable data plane," *ICT Exp.*, vol. 6, no. 1, pp. 62–65, Mar. 2020.
[46] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," in *Proc. CloudNet*, Tokyo, Japan, 2018, pp. 1–3.
[47] Y. Tang, Y. Wu, G. Cheng, and Z. Xu, "Intelligence enabled SDN fault localization via programmable in-band network telemetry," in *Proc. HPSR*, Xi'An, China, 2019, pp. 1–6.
[48] B. Niu, J. Kong, S. Tang, Y. Li, and Z. Zhu, "Visualize your IP-over-optical network in realtime: A P4-based flexible multilayer in-band network telemetry (ML-INT) system," *IEEE Access*, vol. 7, pp. 82413–82423, 2019.
[49] S. Tang, J. Kong, B. Niu, and Z. Zhu, "Programmable multilayer INT: An enabler for AI-assisted network automation," *IEEE Commun. Mag.*, vol. 58, no. 1, pp. 26–32, Jan. 2020.
[50] T. Mizrahi, G. Navon, G. Fioccola, M. Cociglio, M. Chen, and G. Mirsky, "AM-PM: Efficient network telemetry using alternate marking," *IEEE Netw.*, vol. 33, no. 4, pp. 155–161, Jul./Aug. 2019.
[51] G. Fioccola, A. Capello, M. Cociglio, L. Castaldelli, M. Chen, L. Zheng, G. Mirsky, and T. Mizrahi, *Alternate-Marking Method for Passive and Hybrid Performance Monitoring*, document RFC 8321, IETF, Jan. 2018.
[52] T. Mizrahi, C. Arad, G. Fioccola, M. Cociglio, M. Chen, L. Zheng, and G. Mirsky, *Compact Alternate Marking Methods for Passive and Hybrid Performance Monitoring*, document Internet-Draft draft-mizrahi-ippm-compact-alternate-marking-05, IETF, 2019.
[53] A. Riesenberg, Y. Kirzon, M. Bunin, E. Galili, G. Navon, and T. Mizrahi, "Time-multiplexed parsing in marking-based network telemetry," in *Proc. SYSTOR*, Haifa, Israel, 2019, pp. 80–85.
[54] *P4 Alternate Marking Algorithm*. Accessed: Jan. 28, 2020. [Online]. Available: https://github.com/AlternateMarkingP4/FlaseClase
[55] L. Castanheira, R. Parizotto, and A. E. Schaeffer-Filho, "FlowStalker: Comprehensive traffic flow monitoring on the data plane using p4," in *Proc. ICC*, Shanghai, China, 2019, pp. 1–6.
[56] H. Song, T. Zhou, Z. Li, Z. Li, P. Martinez-Julia, L. Ciavaglia, and A. Wang, *Network Telemetry Framework*, document Internet-Draft draft-song-opsawg-ntf-02, IETF, 2018.
[57] A. Karaagac, E. De Poorter, and J. Hoebeke, "In-band network telemetry in industrial wireless sensor networks," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 1, pp. 517–531, Mar. 2020.
[58] *IEEE Standard for Local and Metropolitan Area Networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC Sublayer*, IEEE Standard 802.15.4e-2012, Apr. 2012.
[59] P. Thubert, Ed., *An Architecture for IPv6 Over the TSCH Mode of IEEE 802.15.4*, document Internet-Draft draft-ietf-6tischarchitecture-24, IETF, Jul. 2019.

[60] J. Marques, M. Luizelli, R. T. da Costa Filho, and L. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *J. Internet Serv. Appl.*, vol. 10, no. 12, pp. 1–20, 2019.
[61] 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 Commun. Lett.*, vol. 23, no. 12, pp. 2247–2251, Dec. 2019.
[62] R. Hohemberger, A. F. Lorenzon, F. D. Rossi, and M. C. Luizelli, "A heuristic approach for large-scale orchestration of the in-band data plane telemetry problem," in *Advanced Information Networking and Applications* (Advances in Intelligent Systems and Computing), vol. 1151, L. Barolli, F. Amato, F. Moscato, T. Enokido, and M. Takizawa, Eds. Cham, Switzerland: Springer, 2020.

PILAR MANZANARES-LOPEZ received the master's degree in telecommunication engineering from the Universidad Politécnica de Valencia (UPV), Spain, in 2001, and the Ph.D. degree in telecommunication engineering from the Universidad Politécnica de Cartagena (UPCT), Spain, in 2006.

She started working as an Assistant Professor with UPCT, in 2001, where she currently works as an Associate Professor with the Department of Information and Communication Technologies. Her research interests include software-defined networking, P2P networks, and distributed systems.

JUAN PEDRO MUÑOZ-GEA received the master's and Ph.D. degrees in telecommunication engineering from the Universidad Politécnica de Cartagena (UPCT), Spain, in 2005 and 2011, respectively.

He started working as a Research Assistant with UPCT, in 2006, where he currently works as an Associate Professor with the Department of Information and Communication Technologies. His research interests include software-defined networking and large-scale distributed networks.

JOSEMARIA MALGOSA-SANAHUJA (Senior Member, IEEE) received the degree in telecommunication engineering from the Technical University of Catalonia, Spain, in 1994, and the Ph.D. degree in telecommunication from the University of Zaragoza, in November 2000. In September 1999, he joined the Universidad Politecnica de Cartagena as an Associate Professor. Since 2001, he has been an Assistant Professor with the Department of Information and Communication Technologies, UPCT. His research interests include switching technologies and distributed systems.