Closed-loop Network Automation with Generic Programmable Data Plane (G-PDP)

(*Invited Paper*)

Shaofei Tang, Hui Liang, Min Wang, Tingyu Li, and Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
†Email: {zqzhu}@ieee.org

***Abstract*—In this work, we try to combine software-defined networking (SDN), in-band network telemetry (INT), and data analytics to realize a novel closed-loop network automation system. To architect the data plane, we jointly consider P4-based and protocol-oblivious forwarding (POF) based programmable data plane switches (PDP-SWs) to build a generic programmable data plane (G-PDP), such that the two types of PDP-SWs can benefit each other mutually to overcome their own drawbacks. In the control plane, we expand ONOS to ensure that it can effectively manage the PDP-SWs in the G-PDP. We also design and implement data analyzers (DAs) in its data analytics subsystem, and deploy them distributedly in the G-PDP to alleviate the burden of data processing in the control plane. Our proposal is demonstrated experimentally with a network system prototype that consists of six PDP-SWs, and the results confirm that it can realize closed-loop network automation effectively and balance the tradeoff between flexibility and performance properly.**

***Index Terms*—Network automation, Programmable data plane (PDP), P4, Protocol-oblivious forwarding (POF), In-band network telemetry (INT), Software-defined networking (SDN).**

I. Introduction

Nowadays, the booming of network services has reshaped our daily lives toward “conducting everything on the Internet", and this trend will persist for the foreseeable future [1]. Hence, network systems are undergoing dramatic changes to diversify network elements and protocols in the Internet [2, 3]. For instance, mobile networks are being transitioned to 5G [4, 5], elastic optical networking (EON) has been developed to make the optical layer more flexible [6–8], and programmable data plane (PDP) are unleashing the packet layer from being restricted by network protocols [9, 10]. To better utilize these new technologies, people have proposed software-defined networking (SDN) [11, 12], virtual network slicing [13–15], and network function virtualization (NFV) [16–18], and widely deployed them in production networks. Although the aforementioned advances have made the Internet more adaptive and programmable, they also complicate the network control and management (NC&M) mechanisms [19]. This dilemma motivates researchers to consider how to leverage data analytics to mimic the “observe-analyze-act" loop in human brains for realizing realtime, self-adaptive and cost-efficient network automation [20, 21].

However, it is never a easy job to implement closed-loop network automation because it needs support from a few perspectives. First of all, compared with distributed and autonomous NC&M, a centralized NC&M mechanism can assist network automation better since it can obtain a global view on the network and orchestrate more network elements (NEs). Therefore, SDN [11] is preferred because it decouples the control and data planes of a network to facilitate centralized NC&M. Secondly, network automation requires the NEs to be not only flexible and programmable (*i.e.*, not being restricted by network protocols) but also high-performance (*i.e.*, having high throughput and short packet processing latency). This demand can be supported by PDP [9, 10], which enables the customization of data plane logic (*i.e.*, defining packet fields and programming packet processing pipelines at will) without sacrificing packet processing performance. The protocol independent forwarding (PIF) project of open network foundation (ONF) [22] suggests that PDP can leverage the programming protocol-independent packet processers (P4) [9] or protocol-oblivious forwarding (POF) [10]. As P4 and POF take different technical approaches, each of them has pros and cons.

The third support needed by network automation is the capability of visualizing a network in various granularities. In SDN, the centralized control plane can still use the traditional polling-based approaches (*e.g.*, the simple network management protocol (SNMP) [23]) to monitor the data plane in the out-of-band and coarse-grained manner. However, this is not good enough, because in order to let network automation make accurate and timely decisions, a fine-grained (*i.e.*, in flow-/packet-level) and realtime network monitoring technique is necessary. To this end, in-band network telemetry (INT) [24] has attracted intensive interests recently [25–28]. Specifically, INT is a PDP-enabled network monitoring technique, with which the realtime status of each PDP switch (PDP-SW) on a packet’s forwarding path is recorded and inserted as specific INT fields in the packet’s header. Hence, INT can collect per-packet/per-hop information of a flow, and monitor it comprehensively in the end-to-end manner. This overcomes the delay and consistency issues of polling-based monitoring. Finally, network automation needs to be equipped with an effective data analytics technique, for collecting, parsing, indexing and processing the enormous amount of network status data and inferring accurate NC&M decisions from it in realtime.

In this work, we design, prototype and experimentally demonstrate a closed-loop network automation system. The

innovations in our proposed system can be understood in the following three parts. In the data plane, we jointly consider P4-based and POF-based PDP-SWs to build a generic PDP (G-PDP), such that the two types of PDP-SWs can benefit each other mutually to overcome their own intrinsic drawbacks. For instance, the runtime programmability of POF-based PDP-SWs makes the G-PDP more flexible, while the packet processing performance of P4-based ones ensures the throughput of the G-PDP. In the control plane, we extend the well-known ONOS platform [29] to make sure that it can effectively manage the switches in the G-PDP. Meanwhile, we also implement deep learning (DL) based modules in the control plane to enable knowledge-defined NC&M. Finally, in the data analytics subsystem, we design and implement data analyzers (DAs), each of which can collect, parse and index the packets with INT fields at a speed of 2 million packets per second (Mpps), and deploy them distributedly in the data plane to offload the tasks of data analytics and alleviate the burden of data processing in the control plane.

To demonstrate the effectiveness of our proposal, we consider service function chaining (SFC) [18] as the background, and design and conduct experiments with a network system prototype that consists of six PDP-SWs. Our experimental results indicate that the closed-loop network automation system can make accurate and timely NC&M decisions to manage the G-PDP in a self-adaptive manner. Specifically, the service policies in the G-PDP can be updated adaptively according to realtime network status, without causing any service interruption, while we also maintain the packet processing capacity of critical nodes in the G-PDP to meet the stringent QoS requirements of network applications.

The rest of the paper is organized as follows. Section II explains the background of the key techniques considered in this work. We describe the design of our closed-loop network automation system in Section III. The experimental demonstrations are presented and discussed in Section IV. Finally, we summarize the paper in Section V.

II. Background and Related Work

In this section, we describe the background and related work about PDP and INT briefly, which are the key techniques considered in this work to build closed-loop network automation.

A. Programmable Data Plane (PDP)

OpenFlow [30] is one of the most famous specifications for SDN. However, the data plane specified by OpenFlow is protocol-dependent, which means that it defines match fields and actions based on existing network protocols. Therefore, the specification of OpenFlow has to be updated frequently to accommodate new protocols, while this not only causes compatibility issues but also limits the programmability of data plane. To address these issues, people turned to develop the PDP techniques that can customize data plane logic without being restricted by network protocols. According to the PIF project of ONF [22], both P4 and POF can facilitate PDP.

P4 specifies the guidelines for writing and compiling packet processing programs, and with it, we customize how a PDP-SW handles packets in configuration and runtime phases [22]. Specifically, in the configuration phase, we define packet processing pipelines with the P4 language, compile them to target-specific binaries, and program the programmable packet processors in PDP-SWs with the target-specific binaries, while in the runtime phase, each pipeline is activated by installing flow tables in it. Therefore, it would be difficult to readjust the packet processing pipelines in P4-based PDP-SWs in runtime (*i.e.*, they cannot fully ensure runtime programmability). Besides, with the support from hardware, P4-based PDP-SWs have superior packet processing performance [31]. For the control plane, P4Runtime [32] was developed and integrated in the ONOS platform [29], which enables runtime control of P4-based PDP-SWs via gRPC [33].

On the other hand, POF inherits the operation principle of OpenFlow, *i.e.*, the control plane can program the data plane in runtime by installing flow tables in switches and composing packet processing pipelines with them. However, POF refers to packet fields in a more generic way, *i.e.*, each packet field is denoted as a tuple <*offset, length*>, where *offset* specifies the bit-offset of the field in a packet to tell its location, and *length* describes the length of the field in bits [34]. Then, each entry in a flow table defines its match field(s) in <*offset, length*>, while the corresponding match action(s) are specified with the instructions defined in the underlying primitive instruction set of POF [35] (*i.e.*, the instructions also operate based on <*offset, length*>). Hence, the packet processing in POF is also protocol-agnostic, and POF-based PDP-SWs can fully ensure runtime programmability (*i.e.*, packet processing pipelines in POF-based PDP-SWs can be changed in runtime with *TableMod* messages). Nevertheless, as all the existing POF-based PDP-SWs are based on software [27, 36, 37], their packet processing throughput (*i.e.*, 10 Gbps for packets with sizes at 128 bytes or longer) cannot match to that of P4-based PDP-SWs. As for the control plane, we have expanded ONOS to add in the support of POF [38].

To this end, we can see that P4 and POF achieve different tradeoffs between flexibility and performance, which suggests that there might not be a universal winner between them to adapt to all the scenarios of network automation. This motivates us to jointly consider P4-based and POF-based PDP-SWs to build a G-PDP, and study how to make the two types of PDP-SWs benefit each other mutually.

B. In-band Network Telemetry (INT)

Since its inception [24], INT has been considered as a powerful network monitoring technique to visualize the dynamic operations in networks in a realtime and flow-oriented way. With momentum gained from P4-based PDP-SWs, people have leveraged INT to collect per-packet information at a line-rate of 100 Gbps [39]. However, other than the packet processing throughput, the implementations of INT should also pay attention to the overheads caused by inserting INT fields in packets. Specifically, the insertion of INT fields can not only

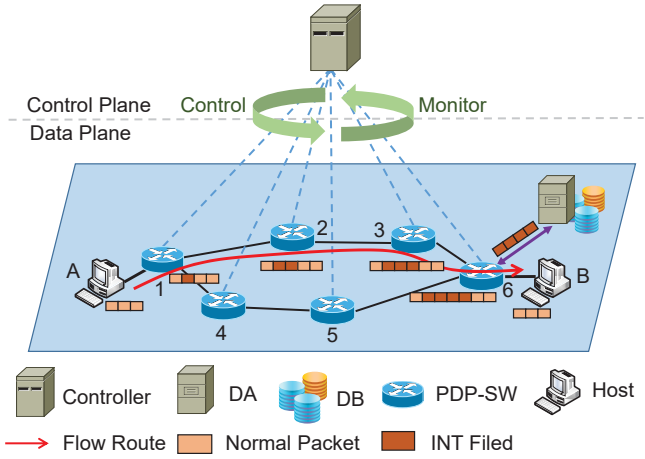


Fig. 1. Self-adaptive INT in a network automation system.

consume noticeable bandwidth resources but also generate excessively long packets [27]. To reduce the overheads of INT, a few selective/probabilistic INT schemes have been proposed in [25–28], which were all based on the idea of sampling packets for inserting INT fields.

Note that, due to its unique benefits on network monitoring, INT has already become an important part in network automation. Therefore, how accurate and timely the NC&M decisions from a network automation system can be actually depends significantly on the INT scheme used in the network automation system. Meanwhile, considering the dynamic network environment of today's Internet, we would expect the network automation system to readjust its INT scheme on-the-fly, such that the tradeoff among overheads, accuracy and timeliness of the INT-based network monitoring is always optimized according to the network status and the requirements of network automation [40–42].

Specifically, Fig. 1 shows the principle of self-adaptive INT in a network automation system. Here, we assume that selective INT is used. The network controller can configure certain PDP-SWs in the data plane to update the INT schemes on them in runtime. For instance, the controller may change the sampling frequency and locations of telemetry data collection, and modify the types of telemetry data to collect at the PDP-SWs. The distributed data analyzers (DAs) in the data plane collect, parse and index the packets with INT fields, and store the extracted telemetry data in the databases (DBs) associated with them. Then, the DAs analyze the telemetry data, and provide realtime feedbacks to the controller if necessary. For example, when a DA detects a sign of network exception, it may suggest the controller to increase the sampling frequency of INT and the types of telemetry data to collect at certain PDP-SWs, in order to zoom in the monitoring on suspicious network region(s) for expediting the trouble-shooting. Nevertheless, as not all the PDP-SWs are runtime programmable, we need to carefully design the operation procedure of our closed-loop network automation to make P4-based and POF-based PDP-SWs work cooperatively for self-adaptive INT.

III. System Design and Implementation

In this section, we first explain our considerations on system design, then describe our design of the control plane, data plane and data analytics subsystem in details, and finally present the implementation of the network automation system.

A. Considerations on Operation Principles

Note that, how to collect and process network status data is a key problem to address in the design of network automation system. The following two operation principles explain our considerations on this problem for the system design.

- *Integration of in-band/out-of-band monitoring*: With the centralized control plane, SDN enables polling-based schemes to monitor the G-PDP in the global but coarse-grained manner, and the status data is collected periodically through the northbound interface. Hence, this can be understood as out-of-band monitoring. On the other hand, INT inserts realtime telemetry data in packet headers to realize fine-grained and flow-oriented network visualization, *i.e.*, the in-band monitoring. To ensure the effectiveness of network automation, the control plane needs to observe and analyze the status of the G-PDP in different granularities, *i.e.*, both realtime/non-realtime and global/flow-specific telemetry data is required. Hence, we will integrate in-band and out-of-band monitoring schemes in our network automation system.
- *Collaboration of centralized/distributed data analysis*: As INT collects per-packet/per-hop information of each flow, it will generate large volumes of telemetry data. Therefore, if we implement DL-based data analytics modules in the distributed DAs and process the telemetry data with them, a fairly large amount of data processing burden can be offloaded from the controller. Then, the centralized and distributed data analysis can collaborate as follows. The DAs realize flow-level data analysis in the distributed way, and only send reports to the controller when necessary (*e.g.*, an exception is detected). The controller collects the reports to obtain the digested information about active flows, and combines the information with the global status data stored locally to reach intelligent NC&M decisions for network automation.

B. Control Plane to Manage G-PDP

Since the G-PDP consists of both P4-based and POF-based PDP-SWs, we design the control plane as shown in Fig. 2 to make sure that the PDP-SWs can be managed efficiently and network applications can be operated with minimized dependency on P4 and POF. Note that, as the control plane is developed based on ONOS, it can leverage the cluster-based configuration to improve scalability. The detailed explanations on the modules in the control plane are as follows.

- *G-PDP Manager*: On the northbound side, this module provides a universal interface to network applications, such that they can care less about the difference between P4 and POF. On the southbound side, according to an

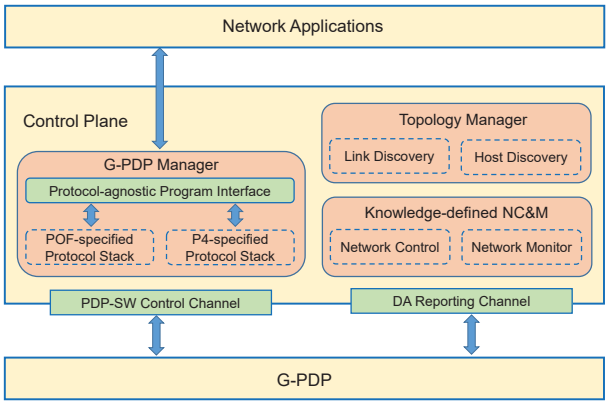


Fig. 2. Design of the control plane for G-PDP.

application's intent, the protocol-agnostic program interface translates its flow rules into P4-based flow entries and POF-based flow tables, and installs them into the corresponding PDP-SWs by using the P4-/POF-specified protocol stacks and the PDP-SW control channel.

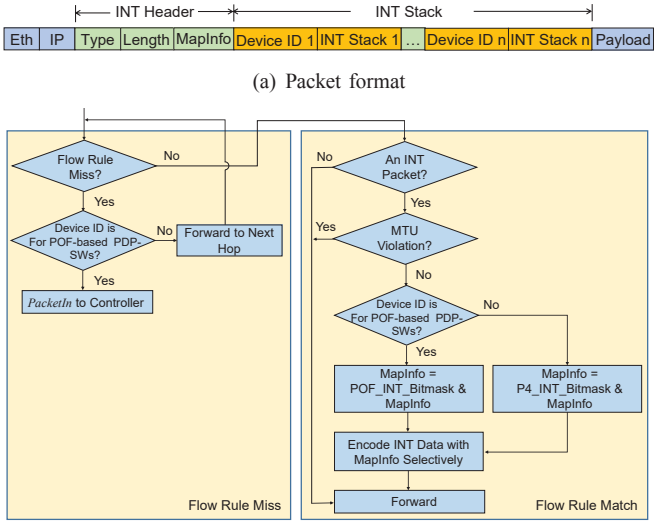
- *Topology Manager*: This module helps the controller to maintain the topology of the G-PDP correctly and timely. Here, the link discovery and host discovery are based on the link layer discovery protocol (LLDP) and the address resolution protocol (ARP), respectively. Meanwhile, we distinguish P4-based and POF-based PDP-SWs by assigning non-overlapped *Device IDs* to them. Specifically, each *Device ID* contains 4 bytes, and the highest 1-byte in it are set as *0x00* and *0xff* to indicate that a PDP-SW is P4-based and POF-based, respectively.
- *Knowledge-defined NC&M:* The network monitor submodule in this module collects telemetry data through both the in-band and out-of-band monitoring schemes, while one or more DL modules are implemented in the network control submodule to facilitate closed-loop network automation based on the telemetry data, *i.e.*, realizing the control loop of 'observe-analyze-act".

C. Design of G-PDP

1) Configuration of G-PDP: In our network automation system, we assume that the G-PDP consists of both P4-based and POF-based PDP-SWs. Here, the P4-based PDP-SWs are the commercially-available ones based on the Tofino ASIC [31], while the POF-based PDP-SWs are software-based ones that we developed in [27] by expanding the OpenvSwitch (OVS) platform, namely, OVS-POF. As the software/hardware architectures of the PDP-SWs are different, some of the telemetry data types that can be collected by them are also different, as listed in Table I. For instance, a POF-based PDP-SW can measure the data traffic on a port in terms of accumulated bytes (*Bytes*), number of accumulated packets (*Packets*), and instant bandwidth usage (*Bandwidth*), while due to its limitations on arithmetic calculations, a P4-based PDP-SW usually only provides the queue length of the buffer on an output port (*Queue Length*). Meanwhile, both types of PDP-SWs can record the per-hop information of an individual packet, such as the latency to process it in a PDP-SW (*Hop Latency*), its ingress time to a PDP-SW (*Ingress Time*), and its input and output ports at a PDP-SW (*Input Port*/*Output Port*).

TABLE I
AVAILABLE TELEMETRY DATA TYPES IN PDP-SWS

Telemetry Data Type	POF-based PDP-SWs	P4-based PDP-SWs
Forwarding Behavior	✓	✓
Queue Length		✓
Bytes	✓	
Packets	✓	
Bandwidth	✓	
Hop Latency	✓	✓
Ingress Time	✓	✓
Output Port	✓	✓
Input Port	✓	✓
Device ID	✓	✓



(a) Packet format

(b) Packet processing procedure

Fig. 3. Designs in G-PDP to enable selective INT.

2) Packet Format and Packet Processing Procedure: Since INT is necessary to our network automation system, we implement selective INT (Sel-INT) [27] in the G-PDP. Here, because the G-PDP consists of P4-based and POF-based PDP-SWs, we design a unified packet format, as shown in Fig. 3(a). The packet fields, which are related to Sel-INT (*i.e.*, the INT header and INT stack), are inserted after the IP header of each INT packet, where an INT packet refers to a packet that contains INT-related fields. Specifically, the length of the INT header is 5 bytes, while the length of the INT stack is variable.

In the INT header, the 2-byte *Type* field contains *0x0908* and we use it as a flag to indicate an INT packet, the 1-byte *Length* field records the number of hops that the packet has experienced, which helps a DA to extract telemetry data in the end, and the 2-byte *MapInfo* field is a bitmap, which instructs

the PDP-SWs along the packet's forwarding path to collect the required types of telemetry data and insert the collected data as fields in the INT stack. A *Device ID* field in the INT stack is 4-byte and it refers to a unique PDP-SW in the G-PDP. The *INT Stack* field following a *Device ID* includes all the required types of telemetry data about the hop.

The Sel-INT in the G-PDP works as follows. When a packet enters its ingress PDP-SW, the PDP-SW first determines whether it should be selected as an INT packet according to the policy provided by the control plane. If yes, the PDP-SW inserts an INT header in the packet, collects telemetry data as required, and encodes the collected data in the INT stack following the INT header. Otherwise, the packet will be forwarded to the next hop directly. Next, when the packet enters each intermediate PDP-SW on its forwarding path, the same operations are applied on it. Finally, the egress PDP-SW duplicates the INT packet to send to a DA, removes all the INT-related fields to convert it back to a normal packet, and then sends the packet to its destination host. Hence, the Sel-INT is made transparent to the users of network applications.

Note that, with Sel-INT, the required types of telemetry data to be collected at each hop are determined by the *MapInfo* field, whose content is provided by the control plane. Specifically, *MapInfo* uses each of its lowest 10 bits to represent a telemetry data type listed in Table I, *i.e.*, a bit 1 means that the corresponding type of telemetry data should be collected, *vice versa*. As P4-based and POF-based PDP-SWs support different types of telemetry data, we introduce two INT bitmasks, *i.e.*, *P4_INT_Bitmask* and *POF_INT_Bitmask*, which equal *0x02ff* and *0x031f*, respectively, according to Table I. Then, as shown in the packet processing procedure in Fig. 3(b), each PDP-SW uses its own INT bitmask to turn off unsupported telemetry data types in the *MapInfo* field of an INT packet, before conducting the required INT operations.

As the length of an INT packet increases after each hop, we also program the PDP-SWs to check its length against the maximum transmission unit (MTU) of the network. Specifically, as illustrated in Fig. 3(b), if a PDP-SW finds that an MTU violation will occur on a packet due to the insertion of INT-related fields at this hop, it will skip the INT operations on the packet. Note that, this generally will not affect the accuracy and timeliness of Sel-INT, because the skipped telemetry data can be collected with other shorter packets.

3) Self-adaptive Network Monitoring with Sel-INT: The basic principle of Sel-INT is based on the observation that most network applications would not require per-packet telemetry data collection, because the network status will not change that fast. Therefore, the overheads of INT can be significantly reduced with packet sampling, which involves the selections of INT packets (*i.e.*, packet selection) and telemetry data types (*i.e.*, type selection). Here, in our network automation system, both the packet selection and the type selection are determined at the ingress PDP-SW of an application flow. Specifically, the frequency of inserting INT headers in the flow's packets decides the sampling frequency of packet selection, while the content of the inserted *MapInfo* determines the type selection.

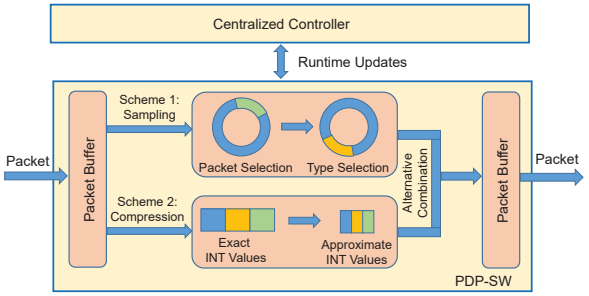


Fig. 4. Self-adaptive Network Monitoring with Sel-INT.

The overheads of INT can be further reduced by leveraging data compression, which maps the exact values of telemetry data to approximate ones that can be represented with less bits. For instance, according to the number of PDP-SWs in a normal network, the *Device ID* field can be compressed to one byte, and as for a type of telemetry data whose value is time-varying (*e.g.*, *Packets* and *Queue Lengths*), we can use division/logarithm operations to compress it with acceptable precision. As shown in Fig. 4, packet sampling and data compression can be combined and applied to INT packets alternatively according to the instructions from the controller.

Note that, for realizing self-adaptive network monitoring, the schemes of Sel-INT on packet flows need to be updated in runtime. For example, since the additional bandwidth usage caused by Sel-INT can induce unnecessary congestion when the original traffic load of a flow is already high, the controller needs to adjust the schemes of Sel-INT dynamically. Meanwhile, as we have explained above, the Sel-INT scheme on a packet flow is enforced at its ingress PDP-SW. Therefore, to fully explore the runtime programmability of POF-based PDP-SWs, we can put them at the edge of the G-PDP to facilitate self-adaptive network monitoring (*i.e.*, letting the controller update their packet processing pipelines for Sel-INT in runtime), while P4-based PDP-SWs should be put on the critical nodes in the G-PDP (*e.g.*, those whose node betweenness centralities or/and connectivity degrees are relatively high) to better utilize their superior packet processing performance.

D. Data Analytics Subsystem

To efficiently process the enormous telemetry data collected with Sel-INT in realtime, we design DAs and deploy them in the G-PDP distributedly. We leverage the data plane development kit (DPDK) [43] to accelerate the collecting and parsing of INT packets in the DAs. We also implement a filtering scheme to save the storage space in each DA, *i.e.*, the consistent or slow-varying values of telemetry data are ignored. To process the collected telemetry data and reach timely and intelligent NC&M decisions, our network automation system leverages both the centralized controller and distributed DAs and implements DL-assisted data analytics in them.

E. System Implementation

Fig. 5 shows the overall architecture for the implementation of our network automation system. To make the system user-

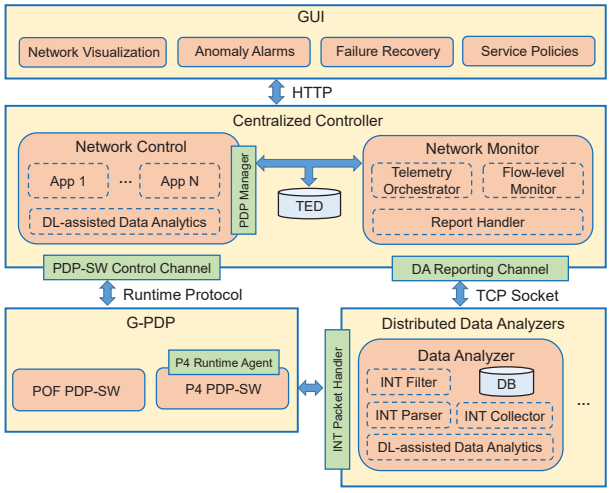


Fig. 5. Architecture of our system implementation.

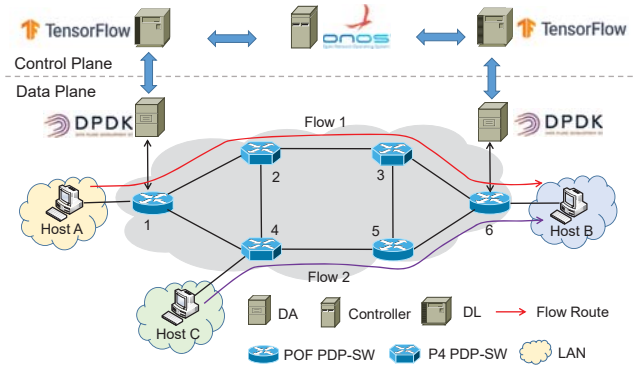


Fig. 6. Experimental Setup.

friendly, we implement a graphical user interface (GUI) based on the open-source Grafana framework, and put it on top of the controller to visualize the status of the G-PDP in realtime. Besides, the GUI also displays the anomaly alarms from the controller and DAs, and provides the interfaces for the network operator to invoke failure recovery and deploy service policies.

For the centralized controller, Fig. 5 specifically explains the implementation of the knowledge-defined NC&M in it, which includes the submodules for network control and network monitor. Moreover, we also implement a traffic engineering database (TED) there to record the provisioning schemes of the active flows in the G-PDP. In the network monitor, the flow-level monitor communicates with the distributed DAs through the DA reporting channel to collect rich telemetry data regarding the applications running in the G-PDP, and based on the telemetry data, the telemetry orchestrator implements self-adaptive adjustments on Sel-INT schemes to optimizing them for network automation. Then, the network control implements the suggested adjustments in the G-PDP through the PDP-SW control channel. Meanwhile, it also leverages DL-assisted data analytics to ensure that the NC&M decisions are tailored properly according to the QoS demands of each application.

In the G-PDP, the packet processing pipelines in POF-based PDP-SWs can be updated in runtime by the controller, with *TableMod* and *FlowMod* messages, while for the P4-based PDP-SWs, we implement a P4 runtime agent based on Stratum on each of them, and make it communicate with the controller via gRPC to realize runtime control (*i.e.*, managing the flow entries in P4-based packet processing pipelines). The configuration of each DA is also laid out in Fig. 5. Here, the database (DB) is used for storing the filtered and digested telemetry data, and it is implemented based on influxDB, which is an open-source time-series database platform.

IV. Experimental Demonstration

In this section, we prototype the proposed network automation system in a small but real network testbed, and conduct experiments to demonstrate its effectiveness.

A. Experimental Setup

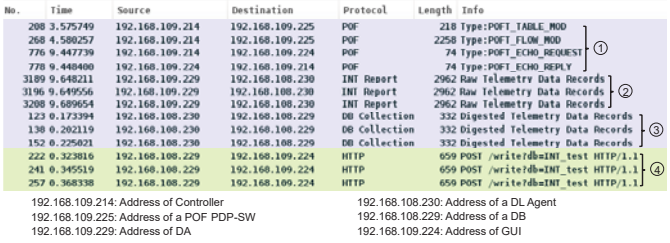
Fig. 6 shows the configuration of the network testbed for experimental demonstrations, which is built with real-world NEs. The centralized controller is developed based on the ONOS platform, and it runs on a high-performance Linux server. All the DL-assisted data analytics modules (*i.e.*, those in the controller and DAs) are implemented by leveraging TensorFlow, and they also run on Linux servers.

In the network testbed, the G-PDP consists of three P4-based PDP-SWs and three POF-based PDP-SWs, and the PDP-SWs are interconnected according to the six-node topology in Fig. 6. Here, each P4-based PDP-SW is a hardware-based commercial product based on the Tofino ASIC [31], while a POF-based PDP-SW is our OVS-POF [27] and it runs on an independent high-performance Linux server. The DAs are home-made, and they leverage DPDK to achieve the packet processing throughput of 2 million packet per second (Mpps) [27]. The end hosts (*i.e.*, *Hosts A-C* in Fig. 6) are emulated with commercial traffic generators/analyzers, and we configure them to be located in different local area networks (LANs). We set the bandwidth capacity of each link in the G-PDP as 10 Gbps. As shown in Fig. 6, we consider two application flows in the network testbed, *i.e.*, *Flows* 1 and 2. Here, *Flow* 1 has time-varying data-rate that changes within $[2, 6]$ Gbps according to a realistic trace in [44], while the data-rate of *Flow* 2 is fixed at 4 Gbps.

B. System Function Verification

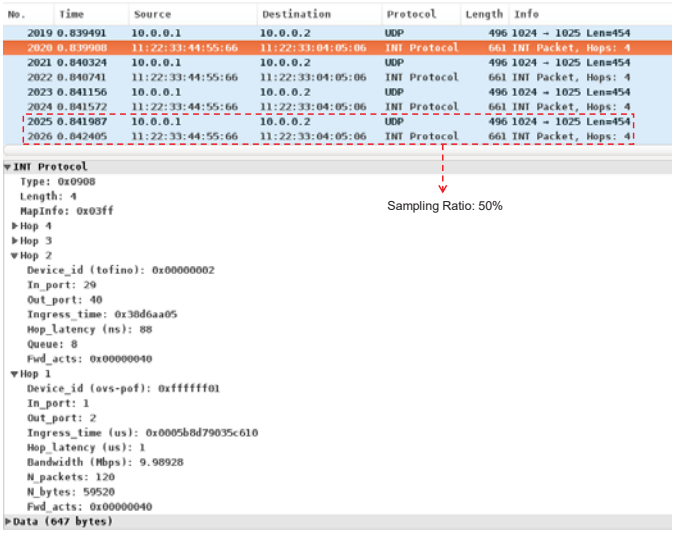
We first show the experimental results regarding the closed-loop operation of the network automation system in Fig. 7. Specifically, the results are about the *Flow* 1 in Fig. 6. First of all, we collect the control messages that the network automation system generates for the flow with Wireshark, and list the results in Fig. 7(a). When the flow first enters the G-PDP, the control plane configures the related PDP-SWs to set up its forwarding path and assign an initial Sel-INT scheme to it (**Step 1**). When INT packets reach their last hop in the G-PDP (*i.e.*, *PDP-SW* 6), the PDP-SW will duplicate and send them to a DA (**Step 2**). Next, after the INT packets have

been collected, parsed, indexed and analyzed by the DA, it will send the telemetry data digested from them to the DB for being queried by the controller in the future, and it will also forward the digested data to the GUI via HTTP messages (**Steps 3** and **4**). Then, the GUI can visualize the statistics of the flow as shown in Fig. 7(b). Meanwhile, the controller can further analyze the digested data and invoke self-adaptive readjustments on the flow's Sel-INT scheme. For instance, the wireshark capture of the packets in *Flow* 1 suggests that the packet sampling frequency has been adjusted to 50%.



(a) Flow of control messages

(b) Snapshot of GUI



(c) Details of an INT packet

Fig. 7. Closed-loop operation of our network automation system.

C. Necessity of G-PDP

We then leverage SFC as the background to demonstrate the necessity of G-PDP. Specifically, we consider two scenarios, each of which needs to deploy virtual network functions (vNFs) on *PDP-SW* 6 in the testbed. There are two types of vNFs, which are for firewall (vNF-FW) and NAT (vNF-NAT). As vNF-FW and vNF-NAT can both be realized by using the "match-and-action" principle, they can be instantiated on PDP-SWs directly by deploying correct packet processing pipelines there. The two active flows are still *Flows* 1 and 2 in Fig. 6.

In *Scenario 1*, we only activate *Flow* 1, but the vNF(s) deployed on *PDP-SW* 6 needs to be updated dynamically over time. Then, we consider two configurations of the G-PDP, *i.e.*, *PDP-SW* 6 is a P4-based and POF-based one, respectively. The experiment runs as follows. Initially, we only deploy vNF-NAT on *PDP-SW* 6, and at $t = 65$ seconds, we need to deploy vNF-FW on it too. Hence, the packet processing pipelines on PDP-SWs need to be reprogrammed in runtime. As for the P4-based PDP-SW, this cannot be done without service interruption, because it has to update the P4 program, and recompile and reload it to its programmable ASIC. As shown in Fig. 8(a), the recompilation of the P4-based PDP-SW causes a service interruption on *Flow* 1, which lasts for $\sim$16 seconds.

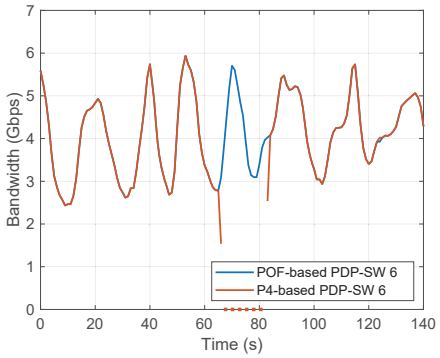
We then change *PDP-SW* 6 to be a POF-based one, and redo the experiment. With the runtime programmability of POF-based PDP-SWs, vNF-FW can be deployed on *PDP-SW* 6 instantly by letting the controller send *TableMod* messages to update its packet processing pipelines. Therefore, as shown in Fig. 8(a), there will be no service interruption. The experimental results in *Scenario 1* suggest that the full flexibility of the data plane cannot be ensured without POF-based PDP-SWs.

In *Scenario 2*, we do not change the vNF deployments dynamically, but consider a dynamic traffic condition. Specifically, *Flow* 1 is first activated at $t = 0$ second, and at $t = 65$ seconds, we activate *Flow* 2 to send traffic from *Host C* to *Host B*. As *Flows* 1 and 2 both go through *PDP-SW* 6, the pressure of packet processing on it can be increased dynamically. Here, we still consider two cases in which *PDP-SW* 6 is a P4-based and POF-based one, respectively. This time, the superior packet processing performance of P4-based *PDP-SW* 6 guarantees that it can process the packets from the two flows and implement the vNFs' operations on them without any difficulty. Specifically, as shown in Fig. 8(b), the packet throughput of P4-based *PDP-SW* 6 is just the sum of the two flows after $t = 65$ seconds. However, due to the limited packet throughput of POF-based PDP-SWs, the POF-based *PDP-SW* 6 can cause packet drops after $t = 65$ seconds.

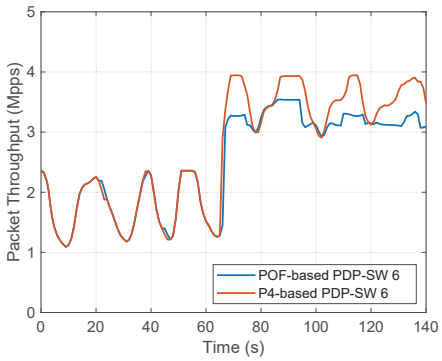
To this end, we can see that by jointly consider P4-based and POF-based PDP-SWs, G-PDP can make them benefit each other mutually. More specifically, in the G-PDP, we should deploy POF-based PDP-SWs on the nodes that might need to readjust their packet processing pipelines dynamically, to fully explore their runtime programmability, and as for the P4-based PDP-SWs, we should put them on the critical nodes that might need to process packets in heavy loads, to better utilize their superior packet processing performance.

D. Performance Benchmark of Control Plane

Note that, the closed-loop operation of our network automation system cannot be realized time-efficiently without an effective controller. Meanwhile, compared with the conventional SDN controllers, the controller in our system needs to take

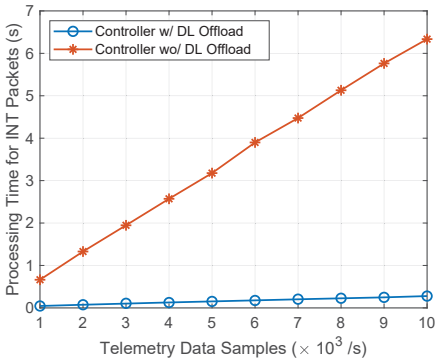


(a) *Scenario 1*: receiving bandwidth on *Host B*

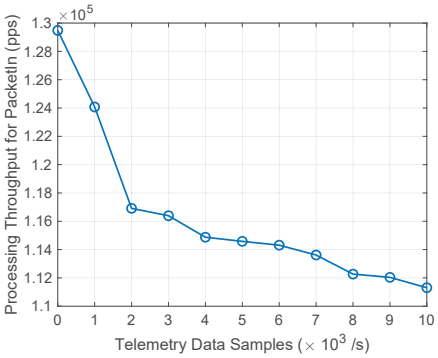


(b) *Scenario 2*: receiving packet rate on *Host B*

Fig. 8. Experimental results to demonstrate the necessity of G-PDP.



(a) Average processing time for INT packets



(b) Average throughput for *PacketIn* messages

Fig. 9. Results of performance evaluations on the controller.

care of additional tasks on DL-assisted data analytics. In our design, we actually introduce distributed DAs to offload some of the additional tasks from the controller. Hence, we design the experiments to verify and quantify the benefits brought by the distributed DAs, *i.e.*, benchmarking our controller with the one that does not leverage distributed DAs to offload the tasks. Fig. 9(a) plots the results on the total processing time for a burst of INT packets that includes different number of telemetry data samples. It can be seen that the DAs effectively offload the DL tasks, and thus the total processing time from the controller that uses the task offloading is much shorter.

Next, we measure our controller's throughput for processing *PacketIn* messages, when it needs to handle the digested telemetry data from the DAs simultaneously. The measurement is done by leveraging the POF Cbench tool [45], which emulates 30 PDP-SWs to generate *PacketIn* messages. In Fig. 9(b), we can see that the controller's throughput for processing *PacketIn* messages decreases by 15.4%, when the speed of telemetry data samples from the DAs increases from 0 to 10,000 samples/second.

V. Conclusion

In this work, we designed, prototyped and experimentally demonstrated a closed-loop network automation system. In its data plane, we jointly considered P4-based and POF-based PDP-SWs to build a G-PDP, such that the two types of PDP-SWs can benefit each other mutually to overcome their own intrinsic drawbacks. In the control plane, we expanded the ONOS platform to make sure that it can effectively manage the PDP-SWs in the G-PDP. Meanwhile, we also implemented DL-based modules in the control plane to enable knowledge-defined NC&M. We designed and implemented DAs in its data analytics subsystem, and deployed them distributedly in the G-PDP to offload the tasks of data analytics and alleviate the burden of data processing in the control plane. The experiments with a network system prototype that consists of six PDP-SWs confirmed the effectiveness of our proposal.

Acknowledgments

This work was supported in part by the NSFC projects 61871357, Zhejiang Lab Research Fund 2019LE0AB01, and SPR Program of CAS (XDC02070300).

References

[1] 2021 Global Networking Trends Report. [Online]. Available: https://www.cisco.com/c/en/us/solutions/enterprise-networks/networking-technology-trends.html.

[2] P. Lu *et al.*, "Highly efficient data migration and backup for Big Data applications in elastic optical inter-data-center networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[3] H. Lu, M. Zhang, Y. Gui, and J. Liu, "QoE-driven multi-user video transmission over SM-NOMA integrated systems," *IEEE J. Sel. Areas Commun.*, vol. 37, pp. 2102–2116, Sept. 2019.

[4] M. Shafi *et al.*, "5G: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE J. Sel. Areas Commun.*, vol. 35, pp. 1201–1221, Jun. 2017.

[5] Y. Gui, H. Lu, F. Wu, and C. Chen, "Robust video broadcast for users with heterogeneous resolution in mobile networks," *IEEE Trans. Mobile Comput., in Press*, 2020.

[6] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[7] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[8] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[9] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[10] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[11] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–98, Apr. 2014.

[12] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[13] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[14] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding (A-SVNE) in optical datacenter networks," *J. Opt. Commun. Netw.*, vol. 7, pp. 1160–1171, Dec. 2015.

[15] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.

[16] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

[17] W. Fang *et al.*, "Joint spectrum and IT resource allocation for efficient vNF service chaining in inter-datacenter elastic optical networks," *IEEE Commun. Lett.*, vol. 20, pp. 1539–1542, Aug. 2016.

[18] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[19] R. Govindan *et al.*, "Evolve or die: High-availability design principles drawn from Google's network infrastructure," in *Proc. of ACM SIGCOMM 2016*, pp. 58–72, Aug. 2016.

[20] A. Mestres *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, pp. 2–10, Jul. 2017.

[21] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.

[22] Protocol Indepedent Forwarding. [Online]. Available: https://opennetworking.org/news-and-events/protocol-independent-forwarding/

[23] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," *RFC 1098*, May 1990. [Online]. Available: https://tools.ietf.org/html/rfc1157.

[24] C. Kim *et al.*, "In-band network telemetry (INT)," *Tech. Spec.*, Jun. 2016. [Online]. Available: https://p4.org/assets/INT-current-spec.pdf.

[25] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," in *Proc. of CloudNet 2018*, pp. 1–3, Oct. 2018.

[26] B. Niu *et al.*, "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. 82 413–82 423, Jun. 2019.

[27] S. Tang *et al.*, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, pp. 708–721, Jun. 2020.

[28] B. Basat *et al.*, "PINT: Probabilistic in-band network telemetry," in *Proc. of ACM SIGCOMM 2020*, pp. 662–680, Aug. 2020.

[29] ONOS. [Online]. Available: https://onosproject.org/.

[30] OpenFlow Switch Specification. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[31] Intel Programmable Ethernet Switch Products. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html.

[32] ONOS+P4 Tutorial for Beginners. [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=16122675.

[33] gRPC: A high performance, open source universal RPC framework. [Online]. Available: https://grpc.io/.

[34] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. of ACM HotSDN 2013*, pp. 127–132, Aug. 2013.

[35] J. Yu *et al.*, "Forwarding programming in protocol-oblivious instruction set," in *Proc. of ICNP 2014*, pp. 577–582, Oct. 2014.

[36] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, pp. 275–288, Mar. 2018.

[37] Q. Sun, Y. Xue, S. Li, and Z. Zhu, "Design and demonstration of high-throughput protocol oblivious packet forwarding to support software-defined vehicular networks," *IEEE Access*, vol. 5, pp. 24 004–24 011, 2017.

[38] H. Huang *et al.*, "Realizing highly-available, scalable and protocol-independent vSDN slicing with a distributed network hypervisor system," *IEEE Access*, vol. 6, pp. 13 513–13 522, 2018.

[39] 100G in-band network telemetry with Netcope P4. [Online]. Available: https://www.netcope.com/Netcope/media/content/100G-In-band-Network-Telemetry-With-Netcope-P4.pdf

[40] D. Bhamare *et al.*, "IntOpt: In-band network telemetry optimization for NFV service chain monitoring," in *Proc. of ICC 2019*, pp. 1–7, May 2019.

[41] R. Hohemberger *et al.*, "Orchestrating in-band data plane telemetry with machine learning," *IEEE Commun. Lett.*, vol. 23, pp. 2247–2251, Dec. 2019.

[42] S. Tang, J. Kong, B. Niu, and Z. Zhu, "Programmable multilayer INT: An enabler for AI-assisted network automation," *IEEE Commun. Mag.*, vol. 58, pp. 26–32, Jan. 2020.

[43] DPDK: Data plane development kit. [Online]. Available: https://www.dpdk.org/.

[44] S. Liu and Z. Zhu, "Generating data sets to emulate dynamic traffic in a backbone IP over optical network," *Tech. Rep.*, 2019. [Online]. Available: https://github.com/lsq93325/Traffic-creation/blob/master/README.md?tdsourcetag=s_pctim_aiomsg

[45] POF Cbench Tool. [Online]. Available: https://github.com/USTC-INFINITELAB/pof-cbench.