BPP over P4: Exploring Frontiers and Limits in Programmable Packet Processing

Jérôme François, Alexander Clemm, Vivien Maintenant, Sébastien Tabor

BPP over P4: Exploring Frontiers and Limits in Programmable Packet Processing

Jérôme François
Inria, LORIA, University of Lorraine, France
jerome.francois@inria.fr

Alexander Clemm
Futurewei, USA
alex@clemm.org

Vivien Maintenant, Sébastien Tabor
Télécom Nancy, University of Lorraine
firstname.lastname@telecomnancy.eu

***Abstract*—This paper describes experiences gained during the development of a Proof-of-Concept implementation of NewIP/BPP, the protocol at the core of a novel packet-programmable networking framework, using P4, a popular SDN technology for the implementation of networking protocols using protocol-independent packet processors. NewIP/BPP introduces a number of novel requirements whose implementation encountered a number of P4 limitations that proved very challenging to overcome. We hope that the resulting insights will be useful for future implementations of NewIP/BPP as well as for its and P4's evolution.**

***Index Terms*—NewIP/BPP, Big Packet Protocol, BPP, New IP, P4, High-Precision Networking, Network 2030**

I. Introduction

The transition of networks from vendor-defined to operator-defined has been one of the key drivers for networking advances in the past decade. This situation changed with the emergence of network softwarization and Software-Defined Networks (SDN). It allows network operators to customize network behavior and introduce new network and service features by themselves, being empowered to build controller software that would program the network's control plane.

One of the latest technologies to emerge in that context is P4, a technology for the Programming of Protocol-independent Packet Processors [1]. P4 allows an outside controller to program the processing of packets that is performed inside a networking device.

At the same time, new waves of networking applications are beginning to emerge as drivers for new network advances over the next decade. Examples include but are not limited to Industry 4.0 (e.g. industrial controllers and connected robotics), haptic applications and the Tactile Internet, smart infrastructure, and Holographic-Type Communications (HTC). Many of those new applications have in common the need for high-precision communications with guarantees for stringent service-level objectives (such as end-to-end latency).

One technology that has been proposed to meet those challenges is New IP / Big Packet Protocol (NewIP/BPP, or simply BPP), a protocol and framework that allows the behavior of packets and flows to be guided from the edge of the network using information encoded in the packets themselves [2]. That guidance can depend on highly dynamic conditions (such as current queue depth), which allows highly flexible behavior that can be adapted very rapidly. As behavior of packets and flows can be guided simply by adding collateral to a packet, without the need to program controllers or network equipment, BPP goes beyond what traditional network softwarization technologies provide on their own, including P4. In that way, BPP provides a next step in agility. A wide range of applications have been proposed, including Latency-Based Forwarding (for high-precision networking services) [3], Operational Flow Profiling (providing improved visibility and service assurance for operations) [4], and optimized task routing for Mobile Edge Computing.

Before it can be leveraged, BPP needs to be supported on networking devices. The fact that it defines a new protocol makes P4 a possible candidate for its implementation. However, BPP has a number of unique features that we expected would be challenging to support and stretch P4 to its limits. This includes BPP's ability to support parametrized commands that lead to more complex parsing rules than would be required for fixed-length header fields and fields of variable lengths.

This paper describes our experiences with implementing BPP using P4. The challenges that we experienced are leading us to conclude that P4 is in fact ill-suited as an implementation choice for our purposes, as the gap between the capabilities BPP aims to provide and the realities of what P4 can support is proving to be too large to bridge. In addition, our open-source implementation and tests of BPP demonstrate its practical benefits in selected scenarios: real-time applications and in-network telemetry. Our Proof-of-Concept (PoC) is released as open-source software at: https://gitlab.inria.fr/francoij/p4bpp.

The remainder of this paper is structured as follows. Section II provides background about BPP and P4. Section III highlights unique requirements that a NewIP/BPP implementation must address and associated challenges. Section IV discusses our design choices. In section V, some scenarios are introduced that showcase our PoC. Related work is presented in section VI. Section VII concludes the paper.

II. Background

A. NewIP/BPP

BPP is a new protocol and programmable packet framework [2]. The basic idea is to introduce a piece of BPP Collateral into packets between the traditional packet header and user payload. This Collateral contains commands and metadata that provide guidance to network devices for how to process the packet. A command might instruct a network device to increase a counter carried in the metadata when a certain

condition is met, for example when a queue of a certain length is encountered to allow a network operator to better assess flow performance. Contrary to Active Networking proposals in the past [5], commands are directives whose scope and lifecycle are isolated to handling the packet and containing flow; they cannot be used to construct programs from a generalized instruction set and do not allow interactions with the payload.

The structure of a BPP packet is depicted in figure 1. BPP can be used in conjunction with a variety of transports, e.g. at layer 2.5 with MPLS or, for our implementation, at layer 3.5 in conjunction with IP. The Collateral consists of a Collateral header, a command block, and a metadata block. Because the Collateral needs to be distinguished from the payload of the packet and the IP header cannot be extended (IPv6's extension headers are insufficient because not allowed to be processed by intermediate nodes), we are defining BPP as a network protocol with a new protocol type, prefixing the Collateral with the traditional IPv4 (or IPv6) header.

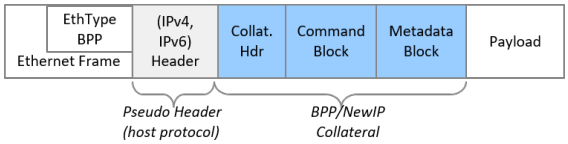


Fig. 1: BPP Packet Structure

Figure 2 depicts the Collateral header, which contains:

- NewIP/BPP version
- Collateral length (16, 32, 128 or multiple of 32)
- Flags for error handling, in case errors are encountered in the processing of commands or metadata along the path
- A reserved flag for future applications

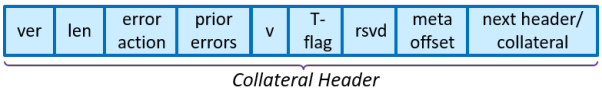


Fig. 2: Collateral Header

A command block (optional) contains a sequence of one or more commands, each structured as follows (figure 3) :

- A command header contains the length of the command and flags indicating whether the command is conditional and whether there is a subsequent command and if it needs to be serialized or can be processed in parallel.
- An (optional) condition set, containing its length and one or two conditions. Each condition has a type (for example, a comparison) and can contain two parameters. Flags indicate whether the condition should be negated, and whether the subsequent condition (if applicable) should be and'ed or or'ed.
- An action set, performed if the condition set evaluates to true, contains the length of the action set and one or more actions. In analogy to conditions, each action contains a type (defining what action to perform) and a set of several parameters. These actions can be used e.g. to modify the metadata of the packet or to reroute the packet.

Parameter values need to be interpreted in different ways depending on the category of the parameter, indicated within the parameter field itself: an actual value to use, a reference to a metadata item by providing an offset in the metadata block, an identifier of a predefined data item such as the egress queue depth, or (in case of support for stateful extensions) a reference to a metadata item retained in a statelet, essentially a programmable flow cache on the device itself.

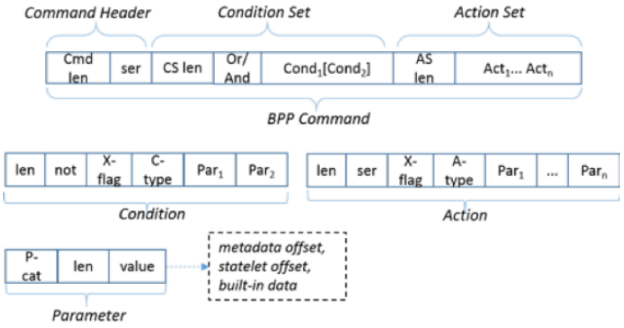


Fig. 3: BPP Command Structure

Finally, the metadata block can contain different data items that are carried with the packet, for instance security data, service level objectives, error details, or variables referred to by parameters in the command block. Data carried in the metadata block can be accessed and modified by intermediate nodes per the command block.

B. P4

We considered the latest version (P4_16) of this language [6]. P4 promotes high flexibility by allowing users to express complex network processing and to offer a switch abstraction that is independent of the actual hardware. Indeed, P4 programs are compiled to a target-independent representation (front-end) to be then transparently compiled again to different specific platform such as NetFPGA [7].

P4 includes a standard architecture *v1model* model to build a L3 switch named BMv2 (Behavioral Model v2). It is composed of the following parts:

- **parser**: P4 packet parser (as a state machine)
- **verify checksum**: Packet checksum verification
- **ingress control**: Actions on incoming (ingress) packets
- **egress control**: Actions on outgoing (egress) packets
- **compute checksum**: Checksum computation
- **deparser**: Deparsing and forwarding

A fundamental item concerns the need to specify the format of packets that the P4 switch must handle. This involves defining the different packet header fields along with their size. This definition is leveraged by the parser and deparser.

P4 is optimized for fixed-length fields. However, it is possible to define one field to be of variable-length of type

varbit. Varbit fields come with notable limitations: it is not possible to modify data in the varbit field. No further support is provided to support substructures within the field, let alone substructures that are in turn of variable length themselves.

Ingress and egress control specify the processing to be performed on packets and are defined using match-action tables (for example, rewriting the MAC address). For this, certain switch metadata (not to be confused with BPP metadata) can be accessed:

- **intrinsic metadata**: provided by the device platform,
- **queueing metadata**: related to the switch queue,
- **user-defined** metadata: provided by the user.

III. Challenges and Solution Approach

There are a number of BPP properties with important ramifications for implementation. There are two challenges in particular, variable substructures and variable number of processing steps required to deal with Collateral.

A. Multiple levels of variable-length substructures

BPP Collateral can contain multiple levels of substructures, some optional, each with variable length and number of components. For example, the command block can contain one or more commands. Each of those can have a varying number of conditions (typically one or two) and actions (typically only one, but more are possible). In turn, each of those may have a varying number of parameters - typically two, but zero, one, or three or more parameters are also possible. Likewise, the metadata block, if present at all, can be of variable length as it can carry a variable number of metadata items.

Unfortunately, P4 simply does not support multiple fields with varying lengths. Mapping the entire Collateral into a single large varbit field might be conceivable in theory, but defeat P4's purpose as the parsing of substructures and packet handling would need to be manually implemented. In addition, deparsing that allows to reconstruct packets that have their Collateral modified would in effect not be possible due to P4 design as highlighted in section II-B.

In order to deal with this limitation, we made the difficult decision to limit the generality of our implementation. Specifically, we decided to mandate a certain fixed Collateral structure that supports a fixed number of commands, each with a fixed number of conditions, actions, and parameters. Given any particular use case, the structure of the Collateral could be determined in advance which the implementation would be able to support. In addition, it is possible to use a "null" condition, action, or parameter for any unneeded fields.

Of course, what gets lost is the generality that allows to reuse the same BPP implementation for any type of Collateral. We felt that was acceptable for our purposes, which was to get a workable implementation of BPP to experiment with. Applying the 80/20 rule, we wanted to be able support what would still be an interesting set of of applications with our limited structure.

At the same time, in case "null" conditions (or actions, or parameters) are applied, the header tax per packet becomes considerably larger than it would have to be otherwise, as the encoding of the Collateral for those cases is less compact. So, our P4-based implementation is also less efficient than would be the case with a more native implementation.

Even with those adaptations, we faced still some additional challenges in parsing the substructures. We will go into the details of that in section IV.

B. Variable number of processing steps

Processing of BPP Collateral can involve a varying number of required processing steps. Some of this is rooted in the variable Collateral structure, as different numbers of commands, actions, and conditions may need to be processed. This aspect is mitigated by limiting our implementation to support only a certain fixed Collateral structure with a given number of commands, actions, conditions, and parameters. This implies an upper bound for processing. Without this limitation, the packet processing pipeline would need to be dimensioned to allow for the maximum number of processing cycles at each step at the pipeline. As BPP does not support iteration or recursion, the number of maximum cycles to process any BPP packet can thus be predetermined.

BPP commands do include an indication whether they need to be serialized or whether they can be executed concurrently. However, we did not make use of this feature.

C. Indirection and variable sources of parameters

Another challenge concerns the possibility that parameters may need to be retrieved from different sources depending on the parameter category:

- **Value**: The parameter field indeed contains the actual value of the parameter itself.
- **Meta**: The parameter field contains a reference to metadata in the packet, defining an offset in the metadata block where the value of the parameter can be found.
- **Statelet**: The parameter field contains a reference to metadata in a statelet where the value to be used can be found. A statelet is a piece of cached memory on the device that is associated with the packet's flow.
- **Data item**: The parameter field contains a reference to data item that contains the value to be used. Examples of data items include the packet's egress queue depth, the packet length, or the current memory utilization.

Therefore, the programmed logic to retrieve the parameter value differs depending on the source, and may require a varying number of processor cycles. Whereas a parameter of type "value" is obtained within one cycle, the other sources each require a level of indirection to be applied.

D. Using P4 externs

P4 allows to invoke additional functionality outside the core of the P4 framework itself, using so-called "extern" functions and data structures which are provided by the underlying host. This provides flexibility by allowing to extend P4 data plane and control plane as needed. While the use of externs would be attractive to support a library of actions to be invoked as part

of BPP commands, we chose to not rely on externs, as use of the P4 framework would have offered us no particular support in their implementation and resulting P4 programs would be no longer platform-independent.

E. Unsupported features

There are a number of other features in BPP that we decided not to support for the initial stage of our implementation in order to showcase an initial set of functionalities. Most noteworthy among these are stateful extensions that revolve around the concept of statelets. Another feature not supported concerns features that allow to place time bounds on the execution of commands after which they are automatically aborted, important in cases where commands map to externs that may not have a predetermined execution time.

IV. Mapping of commands to P4 processing pipeline

A. BPP data structures

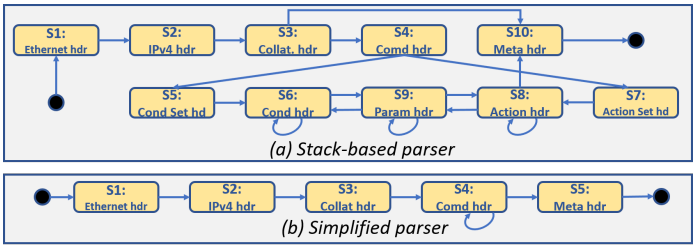


Fig. 4: P4 parsers for BPP packets

Regarding P4's pipeline, the two major operations to support are the parsing and deparsing of packets. As discussed in section III-A, a major problem is the complexity of BPP headers and blocks with many nested and variable-length structures. To circumvent the problem of varbit fields, we slightly adapted the initial specification of BPP by considering different Collateral building blocks (such as Commands and Metadata) as separate "headers" and introducing a *next field* for each to allow a full horizontal chaining of the different fields rather than a nested data structure. This led to the P4 parser shown in figure 4(a). For instance a BPP packet could be: *BPP Header* > *BPP Command* > *BPP Condition Set* > *BPP Condition* > *BPP Parameter* > *BPP Parameter* > *BPP Condition* > *BPP Parameter* > *BPP Parameter* > *BPP Action Set* > *BPP Action* > *BPP Parameter* > *BPP Metadata*. Such structure might be a bit heavy to craft a BPP packet but has the advantage to preserve the flexibility of BPP.

Therefore, despite the fact that loops are not supported by P4, the parsing of BPP packets is now possible and performed using a header stack for each header type previously defined. Indeed, because P4 packet parsing is done using states, an equivalence of a loop is mapped to transitions between parser states. Each state could parse a header type, and push the parsed header into the dedicated stack, then use the *next* field of the header to jump to the next parser state.

However, deparsing is impossible with such a structure. It is not possible to declare the counter variables needed to pull the headers out of their header stack in the correct order, nor to emit packet headers using conditional statements for deparsing. Hence, as explained earlier, we supported only a number of commands fixed in advance, each with a fixed number of conditions and actions, in turn with predefined fixed number and lengths of parameters. This implies a more wasteful encoding of BPP packets as the supported structure in effect provides a template for commands that is padded as needed. It also limits flexibility for BPP applications to an extent. However, when aiming for a very set of specific use cases, those penalties may be entirely acceptable. With this constraint, the parsing does not need to chain different headers anymore to reconstruct initially-proposed BPP variable length fields. The parser is so very condensed in figure 4(b). However a stack is used to store the BPP command headers, allowing to have a variable amount of BPP commands (even if the header stack maximum size is fixed a priori). The issue with the P4 deparser does not exist anymore since deparsing a header stack in P4 leads to the deparsing of every valid header inside it in the same order in which they were pushed in the stack.

BPP metadata could have been implemented with a single varbit containing multiple data items. However, we decided to add more structure to metadata for sake of usability. In our PoC, ten named fields of 64 bits are available to store data.

B. BPP command handling

BPP commands are supported in P4 using a header stack. Since P4 (like BPP) does not allow loops, loop unrolling is applied. In each command, conditions, parameters and actions can be present and are interpreted during the egress processing of the packet. Due to our adaption of BPP to P4 constraints described before, a *length* fields indicates the number of parameters of an action bounded to a maximum value to be defined at the compilation time (two in our case). Our PoC implements a limited set of BPP commands sufficient to demonstrate its viability:

- **SUM (0x01)** : increment a metadata field (2nd parameter) with a value (1st parameter)
- **PUT (0x02)** : puts a value (1st parameter) into a metadata field (2nd parameter).
- **DROP (0xff)** : drops the packet.

Because different types exist for parameters (packet metadata, statelet, built-in), each parameter value is prefixed with its type to address the challenge described in section III-C. In particular, we defined four built-in data items: Time spent by the packet in the switch (from intrinsic P4 metadata); Time spent by the packet in the queue of the switch (from intrinsic P4 metadata); Number of packets in the queue when packet is enqueued (from queuing P4 metadata); Number of packets in the queue when packet is dequeued (from queuing P4 metadata).

Regarding conditions, we also implemented a limited set of operators that can be mixed (less than, greater than, equal to, AND, OR). Parameters are expressed the same way as action parameters. Besides, we include a *negation* field (NOT).

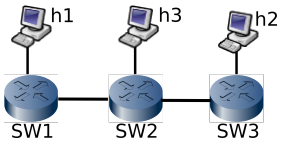
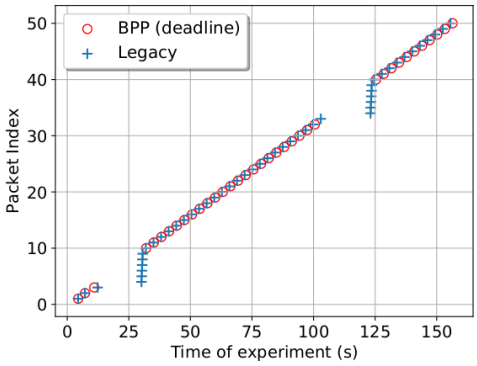
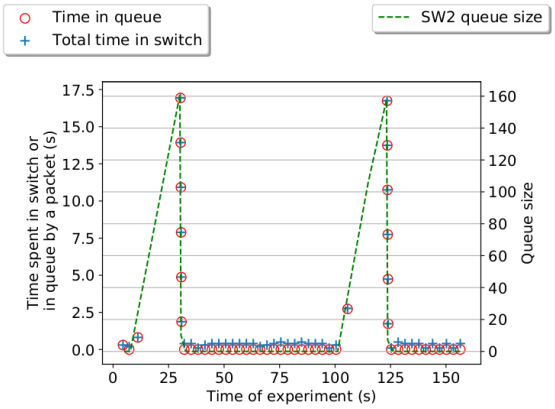


Fig. 5: Experimental topology instantiated with mininet



(a) Obsolete packet dropping



(b) In-band telemetry of switch usage

Fig. 6: BPP application to different use cases

V. Evaluation

Our BPP implementation has been integrated within a L3 switch and tested using p4app [8] that relies on mininet [9] and the *bmv2* software switch. A topology composed of three switches and three hosts is instantiated as shown in Figure 5. This topology is simple but enough to verify the proper capabilities of our BPP implementation because our aim is not evaluate performance. All switches have the same configuration with a queue capacity of 1024 packets and a processing capacity of 10 packets per second. This limitation was arbitrarily set to be able to easily create congestion in order to showcase BPP in the proposed scenarios. In all cases, we generate ICMP traffic (echo request) within BPP packets to have a full packet structure.

A. Scenario 1: real-time application

Considering the envisioned application of BPP in high-precision networks, real-time applications are a good candidate to demonstrate the viability of our proposed implementation.

Regarding the topology in figure 5, host *h2* is constantly receiving data sent by *h1* but data is quickly obsoleted, for example in the case of a streamed real-time video or geo-location of a moving object. The concern with the regular forwarding of IP packet is that obsolescence of data can only be observed by the final receiver, *h2*. Using BPP, we can define into the packets themselves a condition when the packet can be dropped, for example if they have more than a certain amount of time in a queue. It would also be possible make the condition dependent on the sum of time spent in along the different switches of the forwarding path.

h1 sends a packet on a fixed frequency and *h2* receives on-time in normal condition. To illustrate the scenario, artificial congestion is created using *hping3* tool between *h3* and *h2* around time 10s and 100s. Figure 6(a) represents the time when *h2* receives each packet sent and in all cases the observation is clearly observed (lack of packets received). In case of legacy routing, a burst of obsoleted packets arrived at destination after the congestion whereas our prototype was able to drop delayed packets at switch level.

B. Scenario 2: in-band network telemetry

Performing in-band network telemetry allows to transmit monitoring information within application data-packets and so serves two goals. First, resources are saved because no any other connection and so traffic is needed to convey telemetry information. Second, the receiver of packets has direct access to telemetry information. Network telemetry empowers network condition-awareness of applications, usually limited to access to end-to-end metrics. Our PoC provides the ability to know what are the most time-consuming switch before a packet reaches its final destination.

In this scenario, host *h1* sends BPP packets to host *h2* including a BPP command to be executed by every switch along the path that adds up the time the switches took to process the packet and the times the packet spent in each switch queues. Two *metadata* fields are used to store this data.

Figure 6(b) reports telemetry received by *h2*. A congestion is created similarly to the previous scenario. When congestion occurs, the time spent in a queue clearly increases. Due to P4 restrictions highlighted in section IV, the metadata field only contains a maximal number of information to be defined when compiling the p4 switches. In our case, it is limited to 10 fields of 64 bits. Without any compression mechanism, we thus assume a capacity of monitoring 10 switch queue sizes. The queue size of *switch2*, extracted by *h2*, are plotted again in Figure 6(b) (dashed line, right Y-axis scale) in the same experimental setup. Increase of the queue size of switch is observed during the congestion and is consistent with a longer time spent in queue observed in the previous experiment.

VI. Related work

The flexibility brought by P4 allowed the definition of programs of various kinds. Some authors have proposed in-network DDoS protection [10] or load-balancing [11]. Some hypervisors like HyPer4 [12] have been introduced to manage

multiple running P4 programs. DPPx provides a lifecycle management framework to ease the deployment of P4 programs [13]. Although this includes some examples of applications for P4, none of these works aim at enabling the support of a new protocol with similar scope and power as BPP.

Also, there have been many propositions for implementing efficient monitoring support in P4. In addition to usual counters, sketches have been considered [14], even with P4 [15]. Again, none of these works is focused on the mapping and implementation of a new protocol with P4 and so faced different problems. Closer to our work, the authors in [16], [17] aimed at implementing a NDN (Named Data Networking) stack with P4. The authors encountered an equivalent problem as us since names are composed of an arbitrary number of TLVs but they did not face the issue of nested fields.

Research on Network Coding (NC) shares some similarities with our proposition as the packets are decoded and re-coded along their routing. In [18], NC is investigated in the context of critical infrastructure networks and supported by P4. Segment Routing (SR) is a routing scheme where a packet can embed a sequence of segments where it has to go through. A proof-of-concept with P4 exists [19] but it is limited by nature to rewrite the next hop IPv6 address unlike more complex commands promoted by BPP. Potential applications of BPP illustrated in this paper are telemetry operations. In the same space, in-Situ OAM [20] provides a way to collect telemetry data but increases packet size with each node traversed. Another example is In-Network Telemetry (INT), a showcase application for P4 [21] in which the encapsulation of telemetry information is of fixed size and determined in advance.

VII. Conclusion

Our implementation of a BPP PoC with P4 illustrates the gap between protocols that require sophisticated in-network processing capabilities and the ability of advanced data-plane programming frameworks such as P4 to support them.

The highly flexible format of BPP packets combines the ability to express compress packet processing semantics with compact encoding. However, it cannot be readily supported by P4 (P4_16) and we would have faced similar issues with other dataplane abstractions that are inspired by the RMT (Reconfigurable Match Tables) model [22]. With a few compromises, a limited implementation of BPP can still enable powerful and novel use cases. The most severe limitations of P4 revolve around the lack of support for variable length fields and dynamic substructures. In fairness to P4, it was never designed with powerful capabilities such as those envisioned by BPP. However, we hope that our lessons learned will be useful and of interest also beyond the immediate context of BPP, perhaps even to the future evolution of P4 itself.

Future work will focus on optimizations for performance and stateful extensions.

Acknowledgments This work has been partially supported by the NATO Science for Peace and Security Programme under grant G5319 Threat Predict.

References

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

[2] R. Li, A. Clemm, U. Chunduri, L. Dong, and K. Makhijani, "A new framework and protocol for future networking applications," in *SIGCOMM Workshop on Networking for Emerging Applications and Technologies (NEAT)*. ACM, 2018.

[3] A. Clemm and T. Eckert, "High-precision latency forwarding over packet-programmable networks," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Apr. 2020.

[4] A. Clemm and U. Chunduri, "Network-programmable operational flow profiling," *IEEE Communications Magazine*, vol. 57, no. 7, Jul. 2019.

[5] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, D. Rockwell, and C. Partridge, "Smart packets for active networks," in *Open Architectures and Network Programming (OPENARCH)*. IEEE, 1999.

[6] "P4.16 language specification," https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html, accessed: 2020-01-09.

[7] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The p4->netfpga workflow for line-rate packet processing," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19, 2019.

[8] "p4app," https://github.com/p4lang/p4app, accessed: 2020-01-09.

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. ACM CoNEXT '12, 2012.

[10] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

[11] B. Pit-Claudel, Y. Desmouceaux, P. Pfister, M. Townsley, and T. Clausen, "Stateless load-aware load balancing in p4," in *IEEE 26th International Conference on Network Protocols (ICNP)*, 2018.

[12] D. Hancock and J. van der Merwe, "Hyper4: Using p4 to virtualize the programmable data plane," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. ACM CoNEXT '16, 2016.

[13] T. Osiński, H. Tarasiuk, L. Rajewski, and E. Kowalczyk, "DPPx: A p4-based data plane programmability and exposure framework to enhance nfv services," in *IEEE NetSoft*, 2019.

[14] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic Sketch: Adaptive and fast network-wide measurements," ser. ACM SIGCOMM '18, 2018.

[15] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," ser. ACM SIGCOMM '16.

[16] S. Signorello, R. State, J. Francois, and O. Festor, "NDN.p4: Programming Information-Centric Data-Planes," in *Workshop on Open-Source Software Networking (OSSN), IEEE International Conference on Network Softwarization (NetSoft)*, 2016.

[17] R. Miguel, S. Signorello, and F. M. V. Ramos, "Named data networking with programmable switches," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, 2018.

[18] R. Kumar, V. Babu, and D. Nicol, "Network coding for critical infrastructure networks," in *IEEE 26th International Conference on Network Protocols (ICNP)*, 2018.

[19] "Building a PoC of segment routing at 100g using FPGA smart NIC and P4 language," https://tinyurl.com/yb26zakt, accessed: 2020-05-13.

[20] F. Brockners, S. Bhandari, C. Pignataro, H. Gredler, J. Leddy, S. Youell, T. Mizrahi, D. Mozes, P. Lapukhov, R. Chang, D. Bernier, and J. Lemon, "Data Fields for In-situ OAM," Internet Engineering Task Force, Internet-Draft draft-ietf-ippm-ioam-data-10, Jul. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ioam-data-10

[21] J. Liang, J. Bi, Y. Zhou, and C. Zhang, "In-band network function telemetry," in *ACM SIGCOMM Conference on Posters and Demos*, ser. SIGCOMM '18, 2018.

[22] 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, no. 4, 2013.