A New Framework and Protocol for Future Networking Applications

Richard Li, Alexander Clemm, Uma Chunduri, Lijun Dong, Kiran Makhijani
Future Networks, America Research Center
Santa Clara, CA
{renwei.li,alexander.clemm,uma.chunduri,lijun.dong,kiran.makhijani}@huawei.com

ABSTRACT

Future networking applications place demands on networking services that become increasingly difficult to address using existing internetworking technology. This paper presents a new framework and protocol that is designed to meet this challenge, BPP (Big Packet Protocol). BPP is intended as an enabler for a new generation of networking services that depend on the ability to provide precise service level guarantees while facilitating operations. In addition, BPP allows users to define and customize networking behavior from the network edge for their flows in isolation from other users and without needing to rely on lengthy vendor or network operator product cycles.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Network protocols**;

KEYWORDS

Big Packet Protocol, BPP, Future Internet, High-Precision Networking, Programmable Networks, SLAs

ACM Reference Format:
Richard Li, Alexander Clemm, Uma Chunduri, Lijun Dong, Kiran Makhijani. 2018. A New Framework and Protocol for Future Networking Applications. In *Proceedings of Workshop on Networking for Emerging Applications and Technologies (NEAT'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229574.3229576

1 INTRODUCTION

Today's Internet has been a massive success, providing connectivity for a wide variety of applications on a planetary scale. However, in recent years a number of new trends have begun to emerge that indicate fundamental shifts in requirements that future networks will need to be able to address and that will require pushing the boundaries of today's Internet technology [6]. The same trends also point to new possibilities and business opportunities that can be unleashed by technology that meets those new challenges.

These trends include the need for ever more ambitious service level guarantees for applications such as networked Virtual or Augmented Reality (VR/AR), Tactile Internet, or Industrial Internet. Another trend concerns the increasing need for abilities that allow to adapt and customize adapt network behavior to particular applications and deployments, not just by vendors or operators but by applications and end users. Other trends include the emergence of private networks that move contents and data ever-closer to users, as well as the need for greatly simplified network operations.

Fundamentally, many of these trends are putting the best-effort nature of today's Internet to the test, as they point towards the need for high-precision networks: high precision in terms of service level guarantees that can be given, in terms of packet loss that is avoided, in the way resources are being allocated and accounted for, and in terms of behavior that is custom-guided by the needs of individual services and applications as opposed to being determined by best-effort principles. While incremental and fragmented technologies are emerging that cater to different aspects and use cases for each of these trends, what has been lacking so far is a comprehensive, holistic approach that rethinks how networking services are provided. This is what BPP delivers.

BPP is a novel approach to packet-based networking, intended to address the limitations current internetworking technology is faced with. It introduces a new protocol, BPP, as well as the framework to support it. BPP introduces the concept of a BPP Block, a block of data that is carried as part of data packets in addition to header information and user payload. This data includes metadata as well as commands that collectively provide guidance to nodes in the network for how to handle packets and flows. BPP Blocks can be injected or stripped at the edge of a network to interwork seamlessly with existing network technology. BPP does not commit to a particular "host protocol"; it can be used at Layer 3 with IPv6 or IPv4 or even at Layer 2.

BPP allows networking services to be programmed from the edge of the network by information carried in packets and flows without requiring access to a network provider's control plane, without need for administrative access to network devices to reprogram them, and without dependency on lengthy product development or standardization cycles. This programming occurs at the level of the individual packet and flow. BPP packets can affect their own behavior and that of their flow but not the behavior of other packets or flows, in similar ways as tenants in a cloud can program only their own virtual resources and not those of other tenants.

BPP consists of four cornerstones: (1) The BPP protocol itself defines how packets carry information that guides their processing; (2) BPP stateful extensions involving so-called statelets allow packets to retain and interact with state in the network; (3) BPP infrastructure provides the on-device execution environment for packet commands and statelets; (4) BPP network operations provide the functionality required to safely deploy, monitor, and run BPP applications.

This paper presents an overview of BPP, its architecture and design considerations. Due to space limitations, it focuses on the protocol itself, not on other cornerstones of the framework. We introduce BPP in section 2. Section 3 describes a set of use cases and explains by way of example how BPP can address them. Section 4 discusses related work. A brief outlook and conclusions are given in section 5.

2 BPP

2.1 BPP Overview

BPP is based on the idea of injecting meta-information into packets in order to provide guidance to intermediate nodes for how to process those packets. This is done by attaching BPP Blocks with directives that provide guidance for how the packet should be processed or what resources must be allocated for a flow, as well as metadata about the packet and the flow that the packet is a part of. Rather than relying on in-built logic of networking devices that may result in best-effort treatment of the packet, a BPP networking device will act on those commands and metadata to handle the packet, overriding any “regular” packet processing logic that is deployed on the device. Commands can be used, for example, to determine conditions when to drop a packet, which queue to use, when to swap a label, to allocate a resource, or to measure a service level and compare it against a service level objective (SLO).

This concept allows network services and behavior of packets and flows to be programmed by injecting BPP Blocks into packets at the edge of the network. There is no need to program networking devices or network controllers directly. At the same time, the programmed behavior is isolated from other flows and restricted to the packet and its flow.

A BPP packet is structured as depicted in Fig. 1. It consists of a pseudo-header of its host protocol (pointing to BPP Block as next protocol), as well as one (or more) BPP Blocks. Other alternatives could be used to carry BPP Blocks in the future, such as IPv6, NSH [13] or Geneve [8].

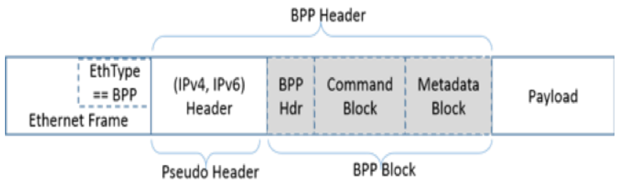

Figure 1: BPP Packet Structure

2.1.1 *BPP Block.*

A BPP Block contains a Command Block with commands and their conditions and parameters, as well as a Metadata Block carrying additional metadata. Both Metadata Block and Command Block are optional, as there will be applications that may have use for only one and not the other. In addition, a BPP Block Header (depicted in Fig. 2) describes the overall structure of the BPP Block and includes several fields, such as BPP version and block length indicating the length of the BPP Block (one of 16, 32, 128, or x, with x being a multiple of 32 provided in a conditional field if x is selected). In addition, there are flags related to error handling: whether prior errors in processing the BPP Block were encountered by other nodes along the path, whether verbose error reporting is requested to record additional error info in metadata, and exception handling behavior (e.g. carry on or drop packet). A timing constraint flag indicates the presence of SLO-related metadata, which can be used (for example) to constrain the number of cycles that can be spent to process a BPP Block. Also included are reserved flags for future use, metadata offset, and optional checksum. Finally, a next header field points to a subsequent BPP Block, if needed. The BPP Block Header fits into 4 to 6 octets, depending on optional fields.

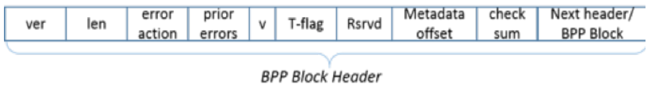

Figure 2: BPP Block Header Structure

2.1.2 *Command Block.*

A BPP Command Block can carry one or more BPP Commands. Each BPP Command consists of three parts: a Command Header, a set of conditions, and a set of actions to be applied when the set of conditions evaluates to true. The structure of BPP commands and selected command components is depicted in Fig. 3.

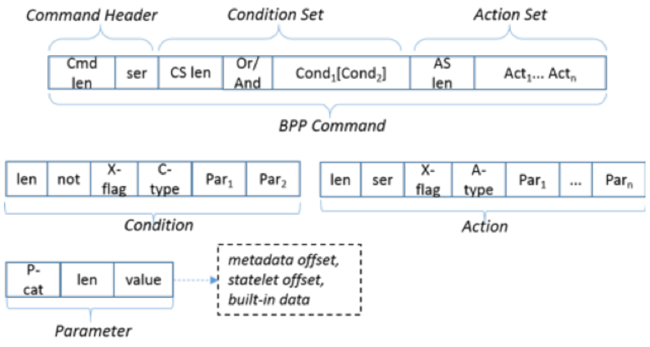

Figure 3: BPP Command Structure

The Command Header includes the length of the command (selectable from several fixed-size choices) and a flag to indicate if the command needs to be serialized or can be executed in parallel with its preceding command, if any.

Conditions are carried as part of a Condition Set. To facilitate fast assessment at line rate and keep command size small, only up to two conditions are allowed with a flag indicating whether to apply “or” or “and”. The type of condition is indicated by a combination of two fields, C-type (Condition Type) and X-flag (eXtension flag, which allows to extend the set of available types of conditions to choose from with new types that may be vendor or deployment specific, as indicatted when the flag is set). Examples of condition types include comparisons (e.g. to check egress link utilization with a threshold) or the “True” condition for unconditional commands. A flag with each condition can be used to negate the condition (logical “not”).

Actions in BPP consist of a set of one or more BPP action primitives (section 2.2) along with any parameters. As with conditions, to facilitate extensions, the type of action is indicated by a combination of two fields, A-type (Action Type) and extension (X-) flag. Each action comes with a flag that indicates whether it needs to be serialized within the command or whether it can be executed in parallel with the prior action.

Parameters of conditions and actions fall into one of several categories: (1) a reference to a built-in data item on a node, for example, queue depth of an egress interface; (2) a reference to a metadata item that is carried in the BPP Block, e.g. for a command to replace a packet header address with a label carried in metadata; (3) a data value, for example a threshold for a condition involving a comparison (e.g. "if current delay > 110% of guaranteed delay, drop this packet"); (4) in conjunction with stateful extensions (section 2.6), a reference to a data item maintained as part of a statelet.

Parameters are structured as follows: one field indicates the category of parameter, a second field its length. A third field provides the actual parameter: an offset to the Metadata Block (in case of metadata or data values that are carried along with metadata), a reference to a built-in data item, or an offset to a statelet (in case of stateful extensions per section 2.6).

While BPP supports conditionals and parallelization constructs, it does not include general programming language constructs such as loops or recursion. Such constructs would increase program complexity and could result in Command Blocks that take longer to execute, impacting delay and buffering needs. However, BPP does provide extension mechanisms that allow network operators to extend the supported set of primitives which can subsequently be referred to by commands and conditions with the extension flag set. This allows to introduce more complex behavior through a backdoor when required.

General computations and the storing of intermediate results are currently not supported by BPP but might be in the future. When supported, a new parameter category will be that of an ephemeral variable, or register, that can be used to store intermediate results during the processing of a packet.

2.1.3 Metadata Block. As is indicated by its name, the Metadata Block can be used to carry additional metadata as part of BPP Blocks. This includes context that can be useful in processing the packet. Examples of such metadata include information about the packet or the flow (e.g. application tags or classifiers), security material, identity metadata, SLOs, and information used for accounting purposes. Metadata can be referenced by conditions and commands themselves, or they can be acted on in other ways by nodes along the path. In addition, the Metadata Block can carry data referred to by parameters in the BPP Command Block. Another use for the Metadata Block is to use it to return results from BPP Commands, for example for applications that perform measurements as part of a service level monitoring scheme or that collect telemetry for operational purposes. Metadata that is attached in this way can, for example, be collected by an egress node. It could also be acted on by other nodes downstream. For example, to provide custom path telemetry, a piece of metadata could be used to count the number of hops for which a packet encountered a specific condition, such as a queue depth above a certain threshold, or to compute its mean.

The structure of the Metadata Block is reasonably straightforward. It includes a set of metadata items, each associated with tag or type, length and value and referable by tag or by offset. In addition, each metadata item includes a permission flag that indicates whether or not the metadata may be altered by intermediate nodes. Extensions to allow for additional signing and securing of metadata, for example in multi-domain scenarios, are for further study.

2.2 BPP Action Primitives

Action primitives that form the basis of BPP Commands fall into several categories, depending on the target of the command – the packet, the device, or the flow. Primitives that act on a packet include primitives to drop, mark, buffer a packet, to select a queue, or to direct the packet to a particular interface. In addition, this includes primitives that interact with metadata: to consume metadata, e.g. to obtain a field's value (by offset, by name i.e. metadata tag, or by name and index), or to produce or update metadata, e.g. to modify a field's value, add a field, or push or pop a value. Some primitives are geared towards measuring service levels such as inter-packet arrival delays. Other primitives allow commands to interact with fields in the packet header, for example to read the TTL field, to modify the DSCP field, or to swap an address in the header with a metadata field.

Primitives that act on the device include actions to allocate and reserve resources, enabling applications to implement their own reservation schemes. Allocation and reservation actions do involve state. Parameters to those primitives include an expiration policy (by default: release of reserved resources after an inactivity period) and a parameter with accounting information (which can be used to indicate how to charge). Other actions include OAM actions, for example actions used to retrieve specific telemetry data.

Primitives that act on a flow, for example to deduplicate or reorder packets, are conceivable but for further study. BPP does not include primitives to interact with payload data. Packet payload is completely opaque to BPP in order to avoid interference with application level protocols and in order to ensure security, privacy, and integrity of user data.

2.3 Built-In Data Items

Parameters can refer to certain data items support for which is already built into BPP and that does not need to be specifically programmed. This includes metadata about the packet, for example the packet-received-time, the packet's ingress and egress interface, the packet modulo sequence number (if applicable), the queue depth of the egress interface, or the packet's transit delay within the node. Stateful extensions also provide access to the time that has elapsed since previous packet and the number of missed packets since the previous packet that was received (indicative of packet drops or path changes). Furthermore, when the capability is supported, BPP can be "flow-aware", allowing parameters to refer to entries in the flow cache and IP Flow Information eXport (IPFIX) Information Elements.

In addition, BPP supports selected metadata about the network device, such as the router ID and selected egress interface information and statistics. Some of the network device data is tied to the enabling of special capabilities, as it may allow a BPP application to discover facts about a network that may be security sensitive. It is conceivable to extend BPP in the future to support a broader range of parameters, such as MIB objects or operational YANG data.

2.4 Error Handling

Whenever execution of a command is attempted, there is a possibility of errors. For example, a device may not have a capability that a command calls for, a resource may not be available, a command may exceed a given time budget or a timing constraint indicated by the T-flag, a piece of metadata may not be available. For those and other scenarios, BPP error behavior needs to be defined. Each command includes a field that indicates what to do when an error is encountered, e.g. whether to carry on, to ignore the command in subsequent hops, or to drop the packet. Once an error is encountered, an error flag is set in the BPP Block. This indicates to nodes further downstream that treatment of the packet has been compromised, including the egresss node that strips the BPP Block from the packet. Any further error handling is then up to the application.

Of course, this provides only very basic information. To support troubleshooting operations, a flag in the BPP Block can be set to request verbose error information. When this flag is set, any node that encounters an error will add a piece of metadata that contains a log entry for the error, such as the router ID at which the error occurred (if enabled by the network provider), the type of error that was encountered, and the offset of the condition, command, or parameter that caused the error. In case stateful extensions are supported (see below), error information can also be recorded in the statelet. This is particularly useful in cases where an error affects the ability to even forward the packet.

2.5 Host Protocol and Size Considerations

BPP Blocks are carried as part of a host protocol, inserted between that protocol's header and payload. To avoid issues with extending IPv4 or IPv6, new EtherTypes are introduced for BPP. The new EtherType combines host protocol header (still processed according to the rules of the host protocol) and BPP Block. Because BPP is intended to be used for applications with stringent service level requirements, and in order to allow BPP to be more easily supported by hardware to achieve processing at line rate, BPP Block size should be kept fixed and to a minimum. At the same time, the number of commands as well as the number and size of parameters and metadata items can vary greatly for different applications, calling for flexibility in block size even if encoding for commands and comparison functions is kept very compact. To bridge both concerns, BPP allows a limited set of different BPP Block sizes that applications can select from. Block sizes include 16, 32, 128, and X octets (X being a multiple of 32). Likewise, command sizes include 4, 8, 32, or X octets (X being a multiple of 8).

In cases where applications want to increase BPP Block size as a packet traverses the network, for example in order to append additional metadata, MTU considerations apply. In cases where a BPP command would lead to exceeded MTU and fragmentation of the packet is not an option, an error will be indicated.

2.6 Stateful Extensions

BPP supports stateful extensions that introduce so-called statelets which retain context about flows on intermediate networking devices. A statelet can be thought of as a special type of programmable flow cache which can be used by BPP commands to deposit state on a device that can then be referred to by subsequent packets. Statelets can be used for purposes as varied as programming custom flow statistics and measurement algorithms to enable new schemes for accounting and service level assurance, to cache context and retain signaled information about a flow, to cache BPP commands for a flow, and to maintain reservations and accounting information for allocated resources used to accelerate services.

Maintaining state on a network device and building networking applications that depend on state being properly maintained also introduces a number of challenges. For example, infrastructure needs to reclaim state that is no longer used. New error conditions arise, such as unability to allocate memory to initialize a statelet. Additional considerations concern the need to deal with scenarios in which path changes occur, which may lead to the need to reinitialize statelets or replicate state along nodes on the path. Some applications, notably measurement applications that compare time stamps and telemetry of the current and of previous packets, may be sensitive to and hence need to be aware of conditions such as packet drops. BPP stateful extensions do include constructs and mechanisms that allow to easily detect, deal with, and act on such scenarios in an orderly fashion which cannot be described here due to space limitations. We plan to explain stateful extensions in greater detail in a future paper.

2.7 BPP Node Infrastructure and Capabilities

BPP requires infrastructure on network nodes in order to work. In addition to being able to parse BPP Blocks and supporting BPP itself, nodes need to provide support for BPP's action primitives. This requires an interpreter or virtualization layer that maps BPP action primitives to device primitives. At some point, special hardware support might be introduced. Node infrastructure also needs to support built-in data items. Stateful extensions further require a statelet cache and a facility to reclaim cache memory from inactive flows.

Not every packet needs to include commands to guide every aspect of packet processing. For example, some applications may attach metadata in order to provide a context-specific classification to a packet, but leave the forwarding decision to the forwarding devices. Likewise, some applications may make use of BPP commands and statelets to program advanced measurement techniques, but not otherwise act on a flow. For this reason, BPP implements a packet-processing hierarchy. The basic idea is that any packet processing of a packet is still applied by the forwarding device as usual, but guidance carried inside a BPP packet takes precedence. In case of stateful extensions, guidance cached inside a statelet is applied second. If neither packet nor statelet give guidance, standard packet processing behavior applies. The packet processing hierarchy is able to differentiate between different behavior aspects; for example, a command to modify a piece of metadata does not interfere with forwarding behavior, but a command to drop a packet would.

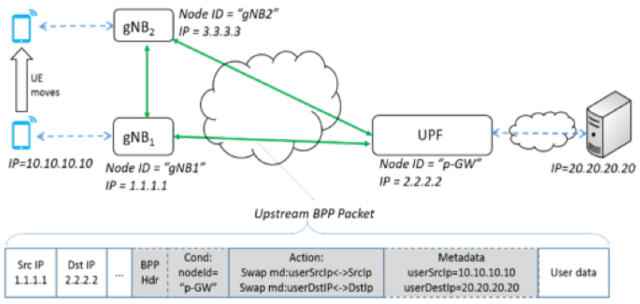

Figure 4: GTP tunnel removal in 5G using BPP

3 EXAMPLES AND USE CASES

3.1 5G Flow Mobility Handover

One of the continuous challenges in mobile networks involves handover of sessions between base stations (here referred to as gNB) during mobility events. 5G handover as currently planned builds on existing 4G/LTE solutions [1]. It involves a tightly coordinated sequence of GPRS Tunneling Protocol (GTP) control messages involving three GTP tunnels: the original GTP tunnel that is used to transport user data prior to the mobility event, the new GTP tunnel that must established before the original can be torn down, and a third tunnel to transmit buffered user data between gNBs. In a GTP tunnel, user packets are in effect "wrapped" for transport within the 4G network, incurring an overhead of 40 header bytes for additional GTP, UDP, and IP headers. The complexity becomes an even greater issue in 5G than in 4G because of a larger number of handovers due to smaller cells and larger number of connections [2].

Using BPP, this scheme can be greatly simplified (see also Fig. 4). Instead of using GTP tunnels, user traffic is directly carried in BPP packets. The basic idea involves swapping header addressing information supplied by the user with addressing information to forward the packet within the core 5G network. Original user addressing information is carried in BPP metadata. A second BPP command carried within the packet is used to perform an operation on the other end, swapping back the metadata into the header to restore user addressing, conditional on the node's address matching the destination address, indicating that the edge of the network has been reached. In addition, commands are used to direct downstream packets to be buffered at the new gNB, and to play out the buffer (directed by a subsequent packet) when the handover is complete. In addition, BPP conditional commands allow to check and take certain mitigating action when buffer overflow is imminent.

3.2 Flow Profiling

There is currently a considerable amount of research involving analyzing characteristics of flows to infer certain information about the nature of the user payload even when encrypted [11] [14]. Solutions typically involve copying flows to an external application that generates a profile of the flow and subsequently matches it against known patterns. A simple example of a profile would be a histogram that captures a flow's packet length distribution. Drawbacks of those solutions include the amount of additional traffic that is generated and possibly legal concerns regarding export of user traffic to external systems for analysis.

Using BPP with stateful extensions, an application can automatically generate a flow's profile without needing to copy traffic. To do so, BPP Blocks contain commands to increment a set of counters conditional on the value of the target data item, e.g. the packet's length. Each counter corresponds to a histogram bucket and is maintained in a statelet. Data from the statelet is subsequently exported to a collector, either directed by a particular packet or upon expiration of the statelet. Further extensions are conceivable in which even the flow classification occurs via BPP commands.

3.3 Mobile Edge Computing

For edge servers with relatively limited computational resources, the server execution/computation time is non-negligible and accounts for the overall latency. Proper server selection depends on expected latency at the server as well as latency incurred in the network. BPP can facilitate awareness of user application's latency requirements (deterministic or stochastic) and task properties (burst arrivals, synchronized arrivals) by carrying those in metadata. In addition, BPP commands can be used to measure service levels as the packet traverses the network. For example, inter-packet delay variation is computed using a corresponding command which subtracts the arrival of the previous packet from the current time, conditional on there being no gaps in the flow from the previous packet. This data is collectively used at the edge to facilitate various decisions including task split and assignment, server selection, and single or multiple queue allocation by the on-path nodes in the edge cloud. Optionally, flows can be "dejittered" by including commands to buffer a packet when ahead or to apply priority queuing when behind.

4 RELATED WORK

Active Networks (AN) were an area of major research in the late 1990'ies and early 2000's [5][3][10]. AN was centered around the idea to let users to inject programs into the network using a controller ("discrete approach") or carried as part of packets ("capsules") and executed by traversed nodes ("integrated approach"). BPP is aimed at enabling high precision for service level guarantees and mitigating effects of the "Best Effort" nature of internetworking technology, whereas AN targets applications that involve payload, such as transcoding or content caching. BPP's programming model is different, geared towards the processing and management of a packet or flow, not the assembly of programs using a higher-level programming language as with AN. Contrary to AN, BPP does not allow payload sent as part of the payload of the packet to be inspected. For example, applications to transcode data streams or apply content caching schemes are outside BPP's scope to avoid threats to privacy or interference with upper-layer applications. Importantly, BPP has an isolation concept that restricts the scope of commands to the packet or flow. For example, you could not use BPP to install a general packet filter on a device. AN provides no such isolation property and can be used for more general device reprogramming. Finally, BPP includes various concepts not inherently supported in AN, such as stateful extensions, metadata, operations support, and various functions needed by flow and packet-oriented applications such as measurements and the detection of conditions such as packet drops or path changes.

Software-Defined Networks (SDN) [15] are a set of technologies that aim to make networks more programmable by virtue of separating data and control plane, typically by moving control logic to separate controllers. BPP moves intelligent processing back inside the network and re-converges, even collapses some aspects of the control plane into the data plane. It facilitates extremely fast control loops for intelligent treatment of packets, enabling many services, applications, and functions that SDN by itself simply can't.

P4 [12] is a language used to program packet processing pipelines in packet forwarding ASICs, allowing to define custom parsing rules and new protocol logic. Several vendors such as Cavium and Broadcom are introducing similar languages. P4's control model follows the SDN architecture and involves a separate control plane to deploy commands on networking devices. BPP, on the other hand, is a new protocol with a control model that involves commands and metadata carried in the packets themselves, processed as packets traverse the network. P4 and related approaches do not define a new protocol. To the contrary, they facilitate the implementation of new (or existing) protocols, making BPP and P4 (and its brethren) in fact complementary.

In-Situ OAM [7] (Operations, Administration Maintenance) can be used to record operational and telemetry data in a packet while the packet traverses a path between two points in the network. In effect, only a single type of command geared at collecting telemetry data is provided.

Segment Routing (SR) [4] is a routing architecture based on source routing, with packets carrying a sequence of segment identifiers that defines the path across the network. Some bits can be used to for commands that program VPN services. BPP's scope is much broader and enables a larger class of applications. As a result, it features a very different and more powerful programming model.

Service Function Chains (SFC)[9] allow to forward data traffic along a sequence of service functions. It is possible to have an ingress node encode SFCs within packets themselves, using a Network Service Header (NSH). BPP allows to program specific functions within the packet itself, supporting stateless as well as stateful programming models. Using commands to assemble a chain of service functions might constitute one specific type of application, but the set of applications BPP enables is much broader. While it is conceivable to leverage some of NSH's metadata encoding for BPP, BPP does not introduce tunneling (unlike NSH).

5 CONCLUSIONS

BPP is a novel approach to packet-based networking that is intended to address many of the limitations that current internetworking technology is faced with in order to enable new waves of networking applications with stringent service level and network precision requirements. BPP lets users and applications customize network behavior in a secure manner that does not affect other flows. Its ability to carry commands and metadata provides the facilities to make packets service-level aware as they traverse the network, which in turn is key to being able to provide service-level guarantees and facilitate high precision. Designed to be extensible and flexible, BPP addresses future networking requirements in an integrated manner, accelerating innovation and facilitating the introduction of new services.

Our next steps include building proofs-of-concepts for selected use cases in order to fine tune various BPP aspects, such as the set of action primitives. We also plan to flesh out in greater detail required infrastructure support and SDKs, including both device platform and execution environment as well as the operations side. We are also investigating whether to support general computations for intermediate results as well as extensions for improved accounting, validation of service level guarantees, and support of end hosts.

ACKNOWLEDGMENTS

For valuable discussions and feedback, we would like to thank Toerless Eckert, Lin Han, Padma Pillay-Esnault, and Yingzhen Qu.

REFERENCES

[1] 3GPP. 2018. *System Architecture for the 5G System. Specification.* Technical Specification (TS) 23.501. 3rd Generation Partnership Project (3GPP). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144 Version 15.1.0.

[2] A. Mohammadkhan, K.K. Ramakrishnan, A. Sunder Rajan, and C. Maciocco. 2016. A Clean-Slate EPC Architecture and Control Plane Protocol for Next-Generation Cellular Networks. *CAN '16 Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking* (Dec. 2016), 31–36. https://doi.org/10.1145/3010079.3010084

[3] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, D. Rockwell, C. Partridge. 1999. Smart Packets for Active Networks. *IEEE Second Conference on Open Architectures and Network Programming (OPENARCH 99), New York, NY/USA* (1999). https://doi.org/10.1109/OPNARC.1999.758557

[4] L. Ginsberg B. Decraene S. Litkowski R. Shakir C. Filsfils, S. Previdi. 2018. *Routing Architecture. Draft-ietf-spring-segment-routing-15.* I-d. 1–30 pages. https://datatracker.ietf.org/doc/draft-ietf-spring-segment-routing

[5] D. Tennenhouse, D. Wetherall. 2002. Towards an Active Network Architecture. *DARPA Active Networks Conference and Exposition (DANCE'02), IEEE, Washington DC/USA* (May 2002). https://doi.org/10.1145/1290168.1290180

[6] ETSI. 2016. *Next Generation Protocols (NGP): Scenario Definitions.* General Specification (GS) DGS/NGP-001. European Telecommunications Standards Institute (ETSI). http://www.etsi.org/deliver/etsi_gs/NGP/001_099/001/01.01.01_60/gs_NGP001v010101p.pdf Version 1.1.1.

[7] F. Brockners, S. Bhandari, C. Pignataro, H. Gredler, J. Leddy, S. Youell, T. Mizrahi, D. Mozes, P. Lapukhov, R. Chang, D. Bernier. 2017. Data Fields for In-situ OAM, Draft-ietf-ippm-iom-data-01. *IETF* (2017).

[8] J. Gross, I. Ganga, T. Sridhar. 2018. Geneve: Generic Network Virtualization Encapsulation. Draft-ietf-nvo3-geneve-06. *IETF* (March 2018). https://doi.org/10.1145/161468.161469

[9] C. Pignataro J. Halpern. 2015. *Service Function Chaining (SFC) Architecture.* RFC 7665. RFC Editor. 1–32 pages. https://www.rfc-editor.org/rfc/rfc7665.txt

[10] J. Lee, R. Fancescangeli, J. Janak, S. Srinivasan, S. Baset, H. Schulzrinne, Z. Despotovic, W. Kellerer. 2011. Netserv: Active Networking 2.0. *IEEE International Conference on Communications (ICC), Kyoto, Japan* (2011). https://doi.org/10.1109/iccw.2011.5963554

[11] M. Zolothukhin, T. Hämäläinen, T. Kokkonen, J. Siltanen. 2016. Increasing Web Service Availability by Detecting Application-Layer DDoS Attacks in Encrypted Traffic. *23rd International Conference on Telecommunications (ICT), Thessaloniki, Greece* (May 2016). https://doi.org/10.1109/ICT.2016.7500408

[12] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review* 4 (2014), 87–95. Issue 3. https://doi.org/10.1145/2656877.2656890

[13] C. Pignataro P. Quinn, U. Elzur. 2018. *Network Service Header (NSH).* RFC 8300. RFC Editor. 1–40 pages. https://www.rfc-editor.org/rfc/rfc8300.txt

[14] W. Shbair, T. Cholez, J. Francoi. 2016. A Multi-Level Framework to Identify HTTPS Services. *IEEE/IFIP Network Operations and Management Symposium (NOMS), Istanbul, Turkey* (2016). https://doi.org/10.1109/ICT.2016.7500408

[15] W. Xia, Y. Wen, C. Foh, D. Niyato, H. Xie. [n. d.]. A Survey on Software-Defined Networking. *IEEE Communication Surveys Tutorials* 17 ([n. d.]).