Network-Programmable Operational Flow Profiling

Alexander Clemm and Uma Chunduri

A detailed understanding of networks and network flows is key for many important operational problems, such as optimizing network performance, identifying causes of degradations or service-level fluctuations, and detecting security threats. However, gaining such insights remains a highly challenging task.

A detailed understanding of networks and network flows is key for many important operational problems, such as optimizing network performance, identifying causes of degradations or service-level fluctuations, and detecting security threats. However, gaining such insights remains a highly challenging task.

Abstract

A detailed understanding of networks and network flows is key for many important operational problems, such as optimizing network performance, identifying causes of degradations or service-level fluctuations, and detecting security threats. However, gaining such insights remains a highly challenging task. This article presents a new approach for obtaining detailed behavioral and performance-related insights at the level of individual packets and flows, based on the concept of network-programmable operational flow profiling. Our approach leverages BPP, a novel networking framework which allows operational flow profiles to be programmed from the network edge without requiring heavy analytics and data collection frameworks.

Introduction

Many future networking applications place high demands on service levels that networks and massively scalable data centers (MSDCs) need to deliver. Examples include networked augmented or virtual reality (AR/VR), the tactile Internet, and Industrial Internet applications. What unites those applications is the fact that they require ultra-reliable low-latency communications (uRLLC) services, as they are highly sensitive to packet loss and do not tolerate delays beyond very few milliseconds [1]. Ensuring such service levels requires a high degree of visibility into and detailed understanding of behavior and performance of underlying networks and flows. This understanding holds the key to a whole spectrum of operational tasks, including optimization of network performance, diagnosis of service-level degradations and fluctuations, and the ability to detect security threats.

Gaining such insights is a highly challenging task and a problem that has been largely unsolved even for today's networking services. This is the case despite impressive advances in big data technology, which still depends on the availability of the "right" data. Today, understanding of network behavior and performance relies largely on the analysis of flow records that provide various statistics about flows, and event and system log messages that indicate unexpected occurrences at devices, end-to-end measurements of service levels, and snapshots of operational data and interface statistics obtained through management protocols. This data is valuable, but it does little to explain what actually happens during a flow. For example, it does not allow relating fluctuations in the dynamic state of a device (e.g., the utilization of different interfaces at a given point in time) with the service level experienced by packets of a particular flow.

More recent advances include telemetry streaming, which allows applications to subscribe to operational data from network devices at finer levels of granularity than was possible before. This is significant progress but still not sufficient to provide an end-to-end and per-flow understanding of what happens during individual packet streams. Another noteworthy advance, in situ operations, administration, and maintenance (IOAM), can be used to collect telemetry for individual packets across a network and debug performance issues in data centers at a very fine-grained level of granularity, for example, to analyze packet queue depths [2]. However, for all its benefits, IOAM is not well suited for understanding the behavior of wholesale flows of production traffic, which requires export and processing of large volumes of data records. Even wholesale packet capture across the network, a non-starter in many scenarios due to scale concerns and legal or regulatory ramifications (e.g., eavesdropping), would not by itself provide sufficient data to provide explanations for questions such as why a temporary service-level fluctuation has occurred or what may have caused some packets to drop at a particular point in time, as context such as operational state during the particular instant of the packet is missing.

What is still lacking today is a mechanism that allows capturing and summarizing both packet and operational data that is observed during a flow, while the flow is occurring (not just after the fact), ideally at line rate, in a way that does not require generating separate data records for each packet and that allows the criteria for what and how to aggregate to be dynamically defined.

In order to address this need, this article introduces the concept of operational flow profiles (OFPs) as a way to gain operational insights into network and flow behavior. An OFP refers to an aggregate of operational data and packet telemetry that is encountered during a flow. For example, an OFP might consist of a histogram of egress interface queue lengths that are encountered at a given node for each packet of a flow. This information might help explain jitter

Digital Object Identifier:
10.1109/MCOM.2019.1800498

The authors are with Futurewei.

caused by varying queue lengths. Such information cannot easily be obtained through existing means: generating, exporting, collecting, and analyzing separate IOAM data records for every packet is prohibitively expensive, while typical time series information of operational data, even when streamed as telemetry, is generally much too coarse grained to correlate it at the packet level.

OFPs can be considered an extension to and generalization of conventional flow profiles that capture properties of the flow itself, such as the distribution of packet lengths, and are typically produced using centralized analytics applications. Of course, rather than just profiling properties that could be analyzed using packet capture (possible but costly), OFPs allow "mashing up" other types of operational data.

While the concept of OFPs is general and could be implemented in different ways, the solution that we propose leverages Big Packet Protocol (BPP), a novel programmable networking framework built around a new network protocol [3]. BPP allows packets to carry additional commands that guide their processing. Stateful extensions to BPP allow packets of a flow to interact with so-called statelets, little pieces of cached memory on networking nodes that are allocated to flows akin to a glorified programmable flow cache. Using BPP, OFPs can be maintained as a part of statelets and programmed dynamically from the network edge. With each packet, the OFP is continuously updated per BPP commands: packets with particular properties can be counted, means and other aggregates can be updated, and telemetry outliers can be recorded.

While some information captured in OFPs can be reminiscent of flow statistics maintained in a regular flow cache, network operators can dynamically introduce new types of OFPs into their network to carry much more versatile semantics. There is no need to export raw data to external systems nor to even apply an external analytics system; instead, all analytics is performed inside network nodes.

The remainder of this article is structured as follows. We present a brief overview of related work and the current state of the art.We provide a brief overview of BPP and its stateful extensions, the technical framework on which OFPs can be subsequently built. We present details of our approach. We introduce OFPs in greater detail and describe how BPP and statelets can be used to generate, maintain, and manage OFPs. We offer concluding remarks and an outlook for future work.

Current State of the Art

There has been a considerable amount of research focusing on profiling of traffic and operational analytics [4], for example, to gain insights about how networks are being used or to detect the spread of malware. One common approach involves learning profiles that are characteristic of traffic known to be of a particular type, then comparing those with the profile of current production traffic and assessing their similarity [5–7]. Those profiles generally involve characteristics of the traffic itself, for example, the number and length of packets in a flow and how that length is distributed among packets. Similarly, the spacing between packets, characteristic of certain streaming applications, could be analyzed. Where profiles go beyond flow statistics that can be obtained as part of Netflow of IPFIX (described below), traffic is captured and forwarded to an external server to perform the analysis, typically involving generating various statistics about the packets in a flow. Of course, if this is to be done at line rate, a very large volume of data needs to be exported and analyzed.

Netflow and IPFIX [8] have been used for many years to aggregate statistics about flows and gain operational insights. Flow records provide flow details such as the number of packets, volume of data transmitted, and number of flow packets that were dropped. It is possible to customize which information elements (IEs) should be collected as part of individual flow records; however, IEs themselves are in effect selected from a catalogue, not freely programmable. While it is possible to extend that catalogue with new enterprise-specific IEs, doing so implies having to make updates to device capabilities which are not easily introduced in a dynamic manner. Instead, lengthy development cycles and vendor support are generally required. For example, it is not possible today to dynamically introduce a new IE that counts the number of packets in a flow that fall within a specific length range, then modify the bounds of its range shortly thereafter as more facts about the network are learned.

Commercial systems for operational analytics frequently focus on time-series analysis of individual parameters to determine anomalies or to detect trends. These systems and related approaches generally rely on the export of large volumes of operational and telemetry data from the network, which is then analyzed in a centralized system. Some refinements to those approaches that decentralize analytics tasks across edge nodes are beginning to emerge [9]. Other approaches focus on end-to-end measurements [10]. All those approaches rarely mash up multiple types of data and focus generally on either network health or service levels, but not both. We are not aware of systems or approaches that combine operational data (e.g., device health status or interface statistics) with data and packet-level observations about flows.

IOAM is a new approach to gain greater operational insight into what happens to individual packets [2]. IOAM is based on the idea of piggybacking trace requests for telemetry data onto packets of a flow, then having nodes add the requested data as they are being traversed. A good handful of data items are available to select from, each associated with a particular bit position set in the packet. Data items include current queue depth, transit delay of the packet within the node, and time the packet was received (at nanosecond resolution). Each node appends its own data to the packet, possibly along with an identification of the node. This enables precise analysis of the performance of individual packets in the network, for example, in a data center setting. A network provider can inject IOAM trace requests at an ingress node of the network, then strip off the IOAM data at the egress node and export it for further analysis to an external

What is still lacking today is a mechanism that allows capturing and summarizing both packet and operational data that is observed during a flow, while the flow is occurring (not just after the fact), ideally at line rate, in a way that does not require generating separate data records for each packet, and does allow the criteria for what and how to aggregate to be dynamically defined.

Statelets can be thought of as a glorified programmable flow cache that is capable of holding arbitrary data that can be dynamically defined, not just flow data. Our approach to OFPs makes extensive use of BPP stateful extensions, as profile data is maintained as a part of statelets and updated for each packet of a flow.

system. Of course, when IOAM is to be applied comprehensively, significant volumes of data will be generated, with a separate record for every packet in a flow, each in turn containing multiple sets of telemetry items for every node on a path. MTU size becomes a limiting factor as packet size grows with the number of hops and requested data items. IOAM delivers interesting data, but any type of further analysis is beyond IOAM's scope and happens outside the network.

Various approaches exist that process and aggregate operational data inside the network, instead of simply exporting raw data and leaving all processing to a centralized system. Classical examples include RMON and the event and expression management information bases (MIBs), which allow users to implement simple monitors on networking devices in the context of Simple Network Management Protocol (SNMP) [11]. A more recent and much more powerful example is Distributed Network Analytics (DNA) [12]. Here, DNA Agents are deployed on networking devices to perform continuous stream queries on operational data at the source, dynamically configured by the DNA Agent itself. Only pre-analyzed data is exported, not raw data streams. However, DNA still requires central orchestration and is not well suited for tasks that involve flow or traffic analysis.

BPP and Statelets

BPP

Big Packet Protocol is a new framework for packet-based networking, centered around a new protocol and the framework to support it. BPP allows packets to provide guidance to network nodes for how to handle the packet and its flow, carrying metadata and commands for this purpose. This provides network operators and edge applications with greater control of packet and flow behavior, as needed, for example, by high-precision networking applications. BPP does not commit to a particular "host protocol" and can be used, for example, at layer 3 with IPv6 or IPv4. This allows BPP to interwork seamlessly with existing network technology. In the following, those aspects that are needed to understand this article are summarized; please refer to [3] for more details.

At BPP's core is the concept of a *BPP Block* that contains metadata and commands for how to handle a packet and its flow. For example, in an Industrial Internet setting, a command could instruct a node to drop a packet if it is late, as determined by a measurement that keeps track of the packet's delay encountered so far. Commands are restricted in their scope to the packet and its associated flow. This allows defining custom behavior for that particular flow without altering network nodes or affecting other packets or flows. BPP Blocks can be injected or stripped off at the edge of a network, which allows packet and flow behavior to be programmed from the edge without needing to modify core network and control infrastructure. BPP's vision is to expand to multi-domain scenarios and end hosts; however, this would introduce significant security ramifications. Therefore, only single network domains will be supported, avoiding the need for additional mechanisms for authentication, authorization, and tamper-proofing of commands and data carried in BPP Blocks.

In many ways, BPP programming is comparable to serverless computing and Lambda functions in the cloud world [13]: Lambda provides a model in which users can simply request the execution of a function on a set of data, without needing to worry about management of virtual compute infrastructure, dimensioning of virtual machines (VMs), or even the life cycle and spinning up or down of containers. This frees users up to truly focus on the needs of their application without introducing a set of second-order problems related to dealing with underlying infrastructure, even when that infrastructure is cloudified and provided as a service. BPP programming introduces similar concepts to the networking world. It allows users to customize behavior and manage their packets and flows without needing to worry about programming software defined networking (SDN) control functionality or orchestrating virtual network functions. At the same time, BPP ensures that any such customization or programming is restricted to a particular flow, isolated from other users and the network provider infrastructure at large.

Statelets

BPP includes a set of stateful extensions, which were mentioned in [3] but not explained in more detail. Since these extensions are crucial to the use case of OFPs, we shall use this context to introduce them in more detail here.

BPP's stateful extensions revolve around the concept of *statelets*. A statelet is a little piece of cached memory on a networking node that is associated with BPP packets of a particular flow using a statelet key. A statelet key is essentially the same as a flow key; it matches packets and their statelets by certain properties, such as source and destination address. BPP commands allow packets and statelets to interact with one another: a BPP command can be used to read, write, and update data in a statelet (e.g., to increment a counter that is part of an OFP). Additional commands allow initialization of statelets and exporting data (e.g., copying data from the statelet into packet metadata or sending it to an external collector). Cleanup of statelets occurs automatically after flow inactivity timeout.

Statelets can be thought of as a glorified programmable flow cache that is capable of holding arbitrary data that can be dynamically defined, not just flow data. Our approach to OFPs makes extensive use of BPP stateful extensions, as profile data is maintained as part of statelets and updated for each packet of a flow.

The structure of a statelet is depicted in Fig. 1. It consists of three parts. A set of housekeeping fields is common to all statelets. Those fields include the statelet key, an expiration field, and certain metadata about the last packet such as its arrival time (e.g., to allow for computation of inter-packet delay) as well as optional sequence metadata (e.g., to allow for detection of packet reordering or drops in a flow). The second part consists of a set of freely programmable data items, used in our case to maintain the OFP. A third part is reserved for future extensions.

Operational Flow Profiling Using BPP

Histogram-Based OFPs: Concept and Examples

As mentioned earlier, an operational flow profile refers to an aggregate of operational data and packet telemetry that is encountered during a flow. It represents an abstraction of properties of the flow as well as conditions and state that are encountered during the flow with the purpose of providing useful operational insights. An OFP is a generalization and extension of a regular flow profile: instead of simply characterizing properties of a flow and its packets, an OFP also characterizes operational conditions encountered during the flow.

There are many data structures that could constitute an OFP. A record with a set of statistical properties (e.g., min, max, mean) of some parameters such as packet length or inter-packet delay is one option. Another type of data structure that is particularly useful for OFPs, and which we shall use for the remainder of our discussion, is that of a histogram. A histogram generally contains multiple buckets, each representing the cardinality of the set of items whose properties match the particular criteria which was established for that bucket. Collectively, histogram buckets span some dimension, such as a parameter's possible value range. For example, a flow profile could be represented by a histogram for the length of packets in the flow, with each bucket representing a different length range from shortest to longest. Each bucket is represented by its own counter, which counts the number of packets that match the bucket criteria. By choosing the number and value ranges of buckets wisely, a histogram can provide a good sense of the overall distribution. Variations are conceivable in which histograms have multiple dimensions or in which buckets have over- or underlapping value ranges.

The concept of histograms and their use as OFPs is depicted in Fig. 2. Figure 2a depicts a histogram representing the distribution of packet lengths in a flow. Its application to determine a classification of the flow by comparing it against a set of known profiles is also conceptually depicted. Figure 2b shows a histogram for the interface utilization of the egress interface that was encountered by each packet during a flow. This one is a true OFP that "mashes up" operational data with traffic data, with packets of a flow guiding the taking of telemetry data snapshots. This particular example depicts a scenario in which most packets of the flow encounter a reasonable interface utilization of below 80 percent (represented by the leftmost bucket) and a small but non-negligible portion encounter very high utilization above 98 percent, which might explain a high number of packet drops observed for that flow. Figure 2c depicts a third histogram showing the distribution of egress queue occupancy encountered by each packet. Here, some packets encounter a very different queue occupancy than others. This could help explain the occurrence of jitter and indicate that resource allocation schemes need to be tuned.

The basic algorithm to maintain a histogram-based OFP for a flow is as follows. When a packet of the flow arrives, check for the target data item that is to be profiled. If this is the first packet, initialize the OFP. Select the proper bucket per the value of the data item (and any other conditions imposed by the profile). Increment that bucket counter. Export OFP on request, when a certain condition is met, or after the flow is deemed to have been terminated.

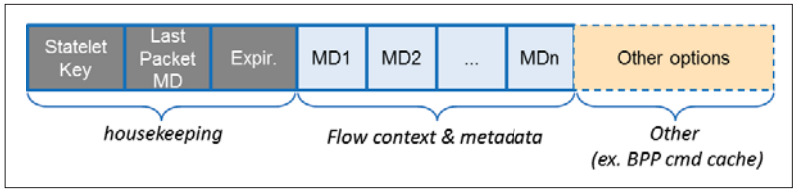

Figure 1. Statelet structure.

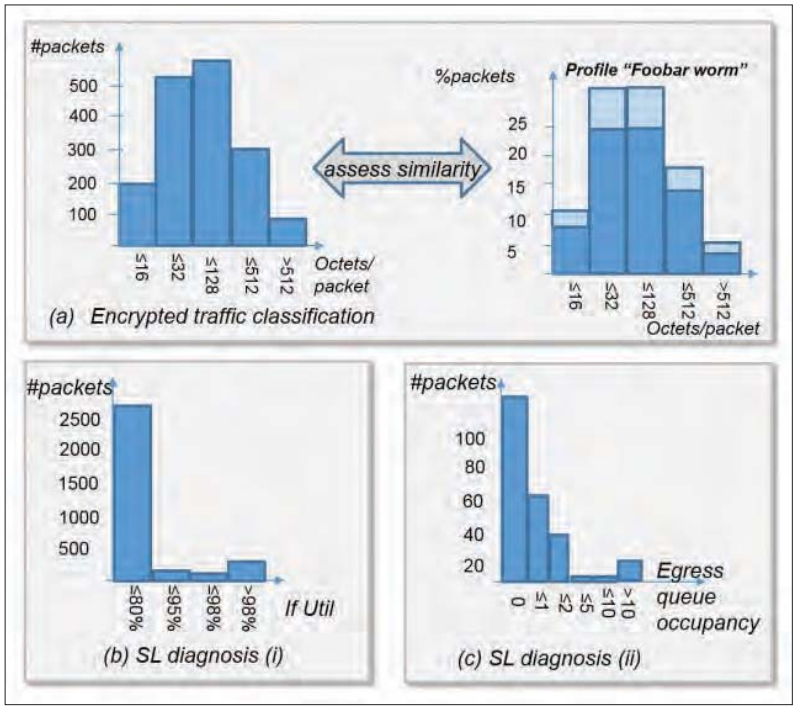

Figure 2. Examples of histogram-based OFPs and their uses.

While the OFP concept looks deceivingly simple, obtaining corresponding operational insights today is exceedingly hard. While data about interface utilization time series and the duration of flows can be obtained using, say, YANG-Push [14] and IPFIX, simply combining this data does not allow accurate correlation of the time series with the particular timing of packets. Likewise, while IOAM can provide data about queue occupancy (but not interface utilization) that is encountered by individual packets, profiling a single flow can require exporting many thousands of records to an external application which subsequently has to crunch large volumes of data.

Note that the data item being profiled may vary; many other data items could be chosen. The number of buckets may also differ, as well as value ranges associated with each bucket. Which particular settings will result in the most characteristic and useful OFP depends on the application and may require dynamic fine tuning. This can be easily accomplished using BPP.

Mapping to BPP

The basic idea behind using BPP to generate OFPs is to have BPP carry a set of commands in each packet that are used to update a statelet that

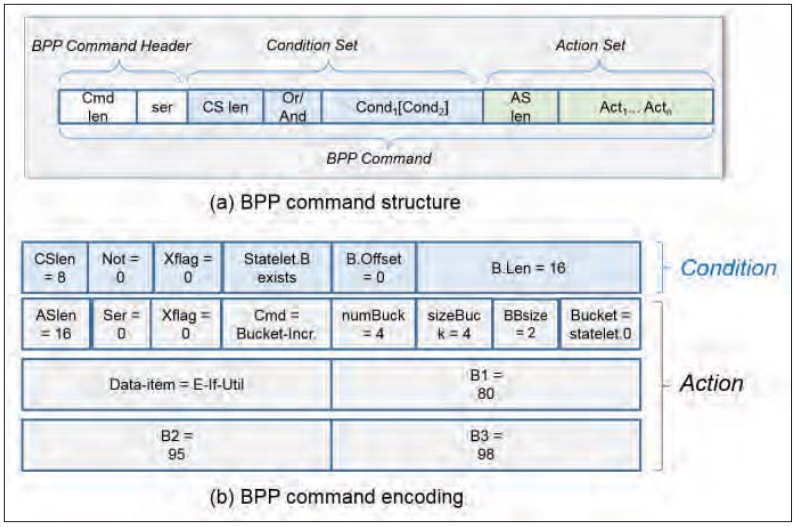

Figure 3. BPP encoding for OFP histogram update.

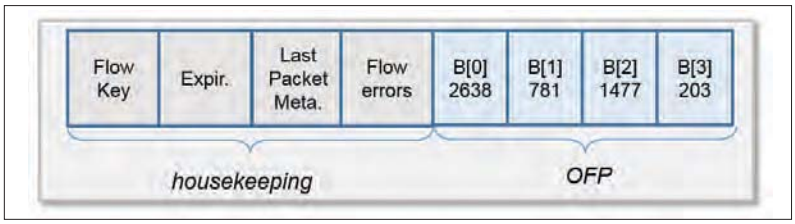

Figure 4. Statelet with OFP.

maintains data for an OFP (e.g., counters associated with histogram buckets). If the condition for the profiled data is met, and it matches the range of a bucket, the corresponding OFP data is updated, in this case the counter incremented. Aggregation functions other than counters and histograms are conceivable, such as means, high water marks, or top-*n* lists. It is also possible to mash multiple conditions and data items, for example, maintain a histogram only for packets of a certain size, or for packets with a short inter-packet arrival delay, or when the egress interface utilization is above a threshold.

Statelet Initialization: When a statelet containing an OFP is not already present on a node, it needs to be initialized with bucket counters set to 0. As an option, OFP metadata that specifies the boundaries between buckets can also be recorded. This avoids the need to carry the same OFP metadata as part of every packet, as only the initial packet of a flow needs to carry the metadata, although special provisions in order to initialize downstream nodes in case of later path changes apply (ranging from periodic resending of metadata to more sophisticated solutions that detect path changes and apply special control logic to redeploy metadata downstream).

Statelet initialization also defines when the statelet is removed and what action to take on removal. By default, an inactivity timer is applied after which the statelet will simply be released. It is possible to specify a different timer value and expiration action, such as exporting the statelet's OFP data to a given destination.

It is not necessary to wait until statelet expiration to export data from it. Export can also be triggered by a command added to the BPP Block of a given packet. For example, an application could trigger OFP export at certain time intervals or after a certain number of packets has been sent.

OFP Update: Updating the OFP involves updating data maintained in the statelet according to the value of the profiled data item that is observed for the current packet. For histograms, this involves selecting the proper bucket by matching the profiled data item's value against bucket ranges, then incrementing the corresponding counter.

BPP Encoding: The encoding of the commands in BPP for the example depicted in Fig. 2b is shown in Fig 3. The BPP Block carries a single command, "bucket-incr," containing the bucket metadata: 4 buckets, 4 octets in length, with bucket boundaries as specified. (The non-optimized version is shown, in which bucket metadata is carried in each BPP packet and not maintained in the statelet.) Executing the command is conditional on the presence of a statelet with a corresponding histogram. A statelet containing the OFP is depicted in Fig. 4.

Other Considerations

Node vs Path Profiling: In many cases, it will be desirable to maintain an OFP at every network device that is traversed. For example, an OFP involving operational data may look different from node to node given their differences in operational state. However, in other cases it will be sufficient to maintain statelets only on nodes meeting a certain criterion, for example, devices that are tagged as edge nodes. For those cases, commands can be marked as conditional to be executed only on those nodes that meet the criterion.

Error Handling: When errors are encountered, such as the inability to access a profiled data item, BPP error handling procedures apply: the error is logged or optionally also flagged in the packet itself, so it can be acted upon by BPP operations infrastructure. Also, an error counter for the statelet can be maintained.

Capabilities: OFPs require support of statelets (i.e., stateful BPP extensions) that may not be supported by every node. Similarly, some OFPs might target a data item that requires special instrumentation that is not supported by every node. When a node encounters an unsupported BPP command or data item, it treats it as an error, setting a corresponding flag in the BPP packet. This can be leveraged to discover any gaps in required capabilities of a BPP network deployment as part of BPP operations.

Blocking-Related Performance Optimization: Interacting with statelets and updating their data may take several processing cycles, leading to the challenge of maintaining OFPs at line rate. In order to avoid introducing unnecessary packet delay, BPP allows certain commands to be flagged, such as the update of data in a statelet, as "non-blocking." While this is not supported by hardware today, this could be leveraged by future hardware to forward a packet even before processing of a non-blocking command is completed.

Packet Length and Fragmentation: The addition of a BPP Block increases the overall size of a packet. This can lead to concerns in conjunction with MTU length. As with other transports, fragmentation and reassembly are controlled by

edge nodes: when a BPP packet is generated at the edge of the network, it is up to the application or gateway to ensure that packet size is not exceeded.

Because OFPs are maintained on network nodes and do not lead to dynamic increases in BPP Block size, fragmentation within the network is not much of an issue. In cases where OFP snapshots or other statelet data are to be added as metadata to BPP Blocks, there is still a chance of increased packet size. Depending on the BPP host protocol, this requires additional encapsulation (e.g., using IPv6 fragment extension headers) [15].

Conclusions and Next Steps

We believe that operational flow profiles constitute a significant advance in network analytics, as they provide valuable operational insights about flows that cannot be easily obtained using existing methods. The fact that OFPs can be dynamically generated using very simple logic which can be applied during packet processing makes them a very attractive application for BPP, a novel networking framework that lets packets carry commands to guide their processing and supports statelets, a form of glorified programmable flow cache.

Important challenges remain. The most important of these concerns the ability to maintain OFPs at line rate without noticeable impact on packet performance of profiled flows. While BPP contains the necessary hooks for performance optimization, supporting this in hardware is an area for further research. As a next step, we plan to build and deploy an experimental implementation of BPP and its stateful extensions, using operational flow profiling as one initial use case. In addition, we plan to develop an SDK to allow for the dynamic introduction and tweaking of OFPs.

Acknowledgment

For inspiring discussions, we would like to thank Richard Li and the members of Futurewei's Future Network team in Santa Clara: Kiran Makhijani, Toerless Eckert, Padma Pillay-Esnault, Lin Han, Yingzhen Qu, Hesham ElBakoury, Lijun Dong, and Alvaro Retana.

References

[1] "Next Generation Protocols (NGP): Scenario Definitions," General Spec. (GS) DGS/NGP-001, ETSI, May 2017.
[2] F. Brockners *et al.*, "Data Fields for In-situ OAM," draft-ietf-ippm-iom-data-05 (work in progress), IETF, Mar. 2019.
[3] R. Li, A. Clemm *et al.*, "A New Framework and Protocol for Future Networking Applications," *ACM SIGCOMM 2018 Wksp. Networking for Emerging Applications and Technologies*, Budapest, Hungary, Aug. 2018.
[4] G. Casale *et al.*, "Special Section on Advances in Big Data Analytics for Management," *IEEE Trans. Network and Service Management*, vol, 15, no. 1, Mar. 2018.
[5] W. Shbair *et al.*, "A Multi-Level Framework to Identify HTTPS Services," *IEEE/IFIP NOMS*, Istanbul, Turkey, Apr. 2016.
[6] M. Zolothukhin *et al.*, "Increasing Web Service Availability by Detecting Application-Layer DDoS Attacks in Encrypted Traffic.," *23rd Int'l. Conf. Telecommun.*, Thessaloniki, Greece, May 2016.
[7] "Encrypted Traffic Analytics," White Paper, Cisco; https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf, 2019, accessed 9 May 2019.
[8] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," IETF RFC 7011, Sept. 2013.
[9] L. Hernandez, H. Cao, and M. Wachowicz, "Implementing an Edge-Fog-Cloud Architecture for Stream Data Management," 2017 *IEEE Fog World Congress*, Nov. 2017.
[10] G. Fioccola *et al.*, "Alternate-Marking Method for Passive and Hybrid Performance Monitoring,," IETF RFC 8321, Jan. 2018.
[11] B. Claise and R. Wolter, "Network Management: Accounting and Performance Strategies," Cisco Press , June 2007. ISBN-13 978-1-58705-198-2.
[12] A. Clemm, M. Chandramouli, and S. Krishnamurthy, "DNA: An SDN Framework for Distributed Network Analytics," *2015 IFIP/IEEE Int'l. Symp. Integrated Network Management*, Ottawa, Canada, May 2015.
[13] T. Lynn *et al.*, "A Preliminary Review of Enterprise Serveless Cloud Computing (Function-as-a-Service) Platforms," *2017 IEEE 9th Int'l. Conf. Cloud Computing Technology and Science*, Hong Kong, China, Dec. 2017.
[14] A. Clemm *et al.*, "Subscription to YANG Datastores," draft-ietf-netconf-yang-push-25 (work in progress), May 2019.
[15] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," IETF RFC 8200, July 2017.

Biographies

Alexander Clemm (aclemm@futurewei.com) is a Distinguished Engineer at Futurewei's Future Networks and Innovation Group in Santa Clara, California. He has been involved in networking software and management technology throughout his career, providing technical leadership for many products from conception to customer delivery, most recently in the areas of high-precision networks and future networking services as well as network analytics, intent, and telemetry. He is also regularly serving on the OC and TPC of management and network softwarization conferences, including IM, NOMS, CNSM, and NetSoft. He has around 50 publications, 50 issued patents, and several books (including *Network Management Fundamentals*) and RFCs. He holds an M.S. degree in computer science from Stanford University and a Ph.D. from the University of Munich, Germany.

Uma Chunduri (uchundur@futurewei.com) is an IP routing expert with extensive background and expertise in service provider networks, enterprise, mobile backhaul and broadband access networks, IP/MPLS, IP FRR, IP Security/IKEv2, and cellular mobility. He is currently a principal engineer with the Future Networks core team at Futurewei, focusing on advanced technology research and standards. His current focus is on programmable network architecture for low-latency and QoS-sensitive industry verticals, on new path-aware routing mechanisms (PPR), and on slice-aware mobility and transport network solutions for 5G cellular networks. He is an active contributor at IETF in Routing, Transport, and Security Areas with 7 published RFCs and 15 Internet Drafts, as well as 40+ patents. He also has several publications and participates in ETSI as Rapporteur of NGP new routing mechanisms.

Important challenges remain. Most important of those concerns the ability to maintain OFPs at line rate without noticeable impact on packet performance of profiled flows. While BPP contains the necessary hooks for performance optimization, supporting this in hardware is an area for further research.