OPSAWG
Internet-Draft
Intended status: Informational
Expires: December 17, 2017

H. Song, Ed.
J. Gong
H. Chen
Huawei Technologies Co., Ltd
June 15, 2017

Requirements for Interactive Query with Dynamic Network Probes
draft-song-opsa-dnp4iq-00

Abstract

This document discusses the motivation and requirements for supporting interactive network queries and data collection through a mechanism called Dynamic Network Probes (DNP). Network applications and OAM have various data requirements from the data plane. The unpredictable and interactive nature of the query for network data analytics asks for dynamic and on-demand data collection capabilities. As user programmable data plane is becoming a reality, it can be enhanced to support interactive query through DNPs. DNP supports node, path, and flow-based data preprocessing and collection. For example, in-situ OAM (iOAM) with user-defined flow-based data collection can be programmed and configured through DNP. DNPs serve as a building block of an integrated network data telemetry and analytics platform which involves the network data plane as an active component for user-defined data collection and preparation.

Table of Contents

1. Introduction

Network service provider's pain points are often due to the lack of network visibility. For example, network congestion collapse could be avoided in many cases if it were known exactly when and where congestion is happening or even better, if it could be precisely predicted well before any impact is made; sophisticated network

attacks could be prevented through stateful and distributed network behavior analysis.

In order to provide better application-centric services, user flows and their interaction with networks need to be tracked and understood.

The emerging trend of network automation aims to keep people out of the OAM and control loop to the greatest extent for automated health prediction, fault recovery, demand planning, network optimization, and intrusion prevention, based on big data analytics and machine learning technologies.

These applications need all kinds of network data, either passing through networks or generated by network devices. For such applications to be effective, the data of interest needs to be retrieved in real time and on demand in an interactive and iterative fashion. Continuous streaming data is often required. Therefore, it is valuable to build a unified and general-purpose network telemetry and analytics platform with integrated data plane support to provide the complete network visibility at the minimum data bandwidth. This is in contrast to the piecemeal solutions which only deal with one single problem at a time.

We propose two ideas to enable such a vision. First, we devise the Dynamic Network Probe (DNP) as a flexible and dynamic means for data plane data collection and preprocessing, which can prepare data for data analytics applications (Note that most of the DNPs are so common that it makes perfect sense to predefine the standard data models for them such that the conventional data plane devices can still be designed and configured to support them). Second, we show the possibility to build a universal network telemetry and analytics platform with an Interactive Query (IQ) interface to the data plane which can compile and deploy DNPs at runtime (or configure DNPs dynamically based on standard data models). In such a system, network devices play an integral and active role. We show a layered architecture based on a programmable data plane which supports interactive queries on network data.

In this document We discuss requirements, use cases, working items, and challenges, with the hope to trigger community interests to develop corresponding technologies and standards.

2. Motivation for Interactive Query with DNP

Network applications, such as traffic engineering, network security, network health monitoring, trouble shooting, and fault diagnosis, require different types of data collection. The data are either

normal traffic packets that are filtered, sampled, or digested, or metadata generated by network devices to convey network states and status. Broadly speaking, there are three types of data to be collected from network data plane: path-based, flow-based, and node-based. Path-based data is usually collected through dedicated probing packets (e.g., ping and traceroute); Flow-based data collection designates user flows to carry data of interest (e.g., in-situ OAM [I-D.brockners-inband-oam-requirements]); Node-based data is directly retrieved from selected network devices (e.g., ipfix [RFC7011]).

Some data is considered atomic or primitive. For example, a packet's arrival timestamp at a particular node cannot be further disintegrated. The atomic data can be used to generate synthetic and combinational data. For example, a packet's latency on a path can be calculated through the packet timestamps at the end of the path. Depending on the application, either data may be required. If the application's real intent is the latter, it makes sense to directly provide such data to reduce the data transfer bandwidth, at the cost of a small processing overhead in the data plane and/or control plane. Some synthetic and combinational data can be acquired through multiple data types, but the most efficient way is preferred for a specific network. For the similar purpose of data traffic reduction, applications may not need the "raw" data all the time. Instead, they may want data that is sampled and filtered, or only when some predefined condition is met. Anyway, application's requirements on data are diversified and unpredictable. Applications may need some data which is not readily available at the time of request.

Some applications are interactive or iterative. After analyzing the initial data, these applications may quickly shift interests to new data or need to keep refining the data to be collected based on previous observations (e.g., an elephant flow detector continues to narrow down the flow granularity and gather statistics). The control loop algorithms of these applications continuously interact with the data plane and modify the data source and content in a highly dynamic manner.

Ideally, to support all potential applications, we need full visibility to know any states anytime anywhere in the entire network data plane. In reality, this is extremely difficult if not impossible. A strawman option is to mirror all the raw traffic to servers where data analytics engine is running. This brute-force method requires to double the device port count and the traffic bandwidth, and poses enormous computing and storage cost. As a tradeoff, Test Access Port (TAP) or Switch Port Analyzer (SPAN) is used to selectively mirror only a portion of the overall traffic. Network Packet Broker (NPB) is deployed along with TAP or SPAN to

process and distribute the raw data to various data analytics tools. There are some other solutions (e.g., sflow [RFC3176] and ipfix [RFC7011]) which can provide sampled and digested packet data and some traffic statistics. Meanwhile, network devices also generate various log files to record miscellaneous events in the system.

When aggregating all these solutions together, we can gain a relatively comprehensive view of the network. However, the main problem is the lack of a unified platform to deal with the general network telemetry problem and the highly dynamic and unpredictable data requirements. Moreover, each piecemeal solution inevitably loses information due to data plane resource limitations which makes the data analytical results suboptimal.

Trying to design an omnipotent system to support all possible runtime data requests is also unviable because the resources required are prohibitive (e.g., even a simple counter per flow is impossible in practice). An alternative is to reprogram or reconfigure the data plane device whenever an unsupported data request appears. This is possible thanks to the recently available programmable chips and the trend to open the programmability to service providers. Unfortunately, the static programming approach cannot meet the real time requirements due to the latency incurred by the programming and compiling process. The reprogramming process also risks breaking the normal operation of network devices.

Then a viable solution left to us is: whenever applications request data which is yet unavailable in the data plane, the data plane can be configured in real time to return the requested data. That is, we do not attempt to make the network data plane provide all data all the time. Instead, we only need to ensure that any application can acquire necessary data instantly whenever it actually needs it. This data-on-demand model can support effectively omni network visibility, Note that data collection is meant to be passive and should not change the network forwarding behavior. The active forwarding behavior modification is out of the scope of this draft.

Data can be customized dynamically and polled or pushed based on application's request. Moderate data preprocessing and preparation by data plane devices may be needed. Such "in-network" processing capability can be realized through DNP.

3. Use Cases

3.1. In-Situ OAM with User Defined Data Collection

In-situ OAM [I-D.brockners-inband-oam-requirements] collects data on user traffic's forwarding path. From the control and management plane point of view, each data collection task is a query from the OAM application. In case the data collection function is not hard coded in network devices, DNP can be dynamically deployed to support the in-situ OAM.

While the current in-situ OAM drafts only concern the data plane packet format and use cases, the applications still need a control and management interface to dynamically enable and disable the in-situ OAM functions, which involves the tasks such as choosing the source and destination nodes on the path, the flow to carry the OAM data, and the way to handle the data at the path end. These configuration tasks can be done through DNP.

More importantly, in-situ OAM [I-D.brockners-inband-oam-data] may collect user-defined data which are not available at device configuration time. In this case, the data can be defined by DNP. DNP can further help to preproess the data before sending the data to the subscribing application. This can help to reduce the OAM header size and the application's work load.

3.2. DDoS Detection

In a data center the security application wants to find the servers under possible DDoS attack with a suspiciously large number of connections. It can deploy DNPs on all the portal switches to periodically report the number of unique flows targeting the set of the protected servers. Once the queried data are collected, it is easy to aggregate the data to find the potential DDoS attacks.

3.3. Elephant Flow Identification

An application wants to query the network-wide top-n flows. Various algorithms have been developed at each network device to detect local elephant flows. These algorithms can be defined as DNPs. A set of network devices are chosen to deploy the DNPs so each will periodically report the local elephant flows. The application will aggregate the data to find the global elephant flows. The elephant flow identification can be an interactive process. The application may need to adjust the existing DNPs or deploy new DNPs to refine the detection results.

In some cases, the local resource in a network device is not sufficient to monitor the entire flow space. We can partition the flow space and configure one network device in a group with a DNP to

track only a subset of flows, given the assumption that each device can see all the flows.

3.4. Network Congestion Monitoring

Network congestion is reflected by packet drops at routers or switches. While it is easy to get the packet drop count at each network device, it is difficult to gain insights on the victims, hot spots, and lossy paths. We can deploy DNPs to acquire such information. DNPs are deployed on all network devices to collect the detailed information about the dropped packet such as its signature and the port it is dropped. Based on the collected data, the application can generate the report on the top victims, hot spots, and the most lossy paths.

4. Enabling Technologies for DNP

Network data plane is becoming user programmable. It means the network operators are in control of customizing the network device's function and forwarding behavior. Figure 1 shows the industry trend, which shapes new forms of network devices and inspires innovative ways to use them.

```
                 +-------+         +-------+           +-------+
                 |       |         |       |           |       |
                 |  NOS  |         |  APP  |           |  APP  |
                 |       |         |       |           |       |
                 +-------+         +-------+           +-------+
                     ^                ^                     ^
                     |                |                     | runtime
           decouple  |    ------>     | config time --->    | interactive
                     |                | programming         | programming
                     V                V                     V
                 +----------+      +-------------+  +-------------+
                 | box with |      | box with    |  | box with    |
                 | fixed    |      | programmable|  | interactive |
                 | function |      | chip        |  | programmable|
                 | ASIC     |      |             |  | chip        |
                 +----------+      +-------------+  +-------------+
```

Figure 1: Towards User Programmable Data Plane

The first trend is led by the OCP networking project, which advocates the decoupling of the network operating system and the network device hardware. A common Switch Abstract Interface (SAI) allows applications to run on heterogeneous substrate devices. However,

such devices are built with fixed function ASICs, which provide limited flexibility for application customization.

The second trend is built upon the first one yet makes a big leap. Chip and device vendors are working on opening the programmability of the NPU, CPU, and FPGA-based network devices to network operators. Most recently, programmable ASIC has been proven feasible. High level languages such as P4 [DOI_10.1145_2656877.2656890] have been developed to make the network device programming easy and fast. Now a network device can be programmed into different functioning boxes depending on the program installed.

However, such programming process is considered static. Even a minor modification to the existing application requires to recompile the updated source code and reinstall the application. This incurs long deployment latency and may also temporarily break the normal data plane operation.

User programmable data plane should be stretched further to support runtime interactive programming in order to extend its scope of usability, as proposed in POF [DOI_10.1145_2491185.2491190] Dynamic application requirements cannot be foreseen at design time, and runtime data plane modifications are required to be done in real time (for agile control loop) and on demand (to meet data plane resource constraints). Meanwhile, the data plane devices are capable of doing more complex things such as stateful processing without always resorting to a controller for state tracking. This allows network devices to offload a significant portion of the data processing task and only hand off the preprocessed data to the data-requesting applications.

We can still use static programming with high level languages such as P4 to define the main data plane processing and forwarding function. But at runtime, whenever an application requires to make some modification to the data plane, we deploy the incremental modification directly through the runtime control channel. The key to make this dynamic and interactive programming work is to maintain a unified interface to devices for both configuration and runtime control, because both programming paths share the same data plane abstraction and use the same back-end adapting and mapping method.

NPU-based network devices and virtual network devices running on CPU/GPU can easily support the static and runtime in-service data plane programmability. ASIC and FPGA-based network devices may be difficult to support runtime programming and update natively. However, for telemetry data collection tasks, the device local controller (or even remote servers) can be used in conjunction with the forwarding chip to complete the data preprocessing and

preparation. After all, applications do not care how the data probes are implemented as long as the same API is maintained.

5. Dynamic Network Probes

Network probes are passive monitors which are installed at specific forwarding data path locations to process and collect specific data. DNPs are dynamically deployed and revoked probes by applications at runtime. The customizable DNPs can collect simple statistics or conduct more complex data preprocessing. Since DNPs may require actively modifying the existing data path pipeline beyond simple flow entry manipulation, these operations need to be done through interactive programming process. When a DNP is revoked, the involved shared resources are automatically recycled and returned back to the global resource pool.

DNPs can be deployed at various data path locations including port, queue, buffer, table, and table entry. When the data plane programmability is extended to cover other components (e.g., CPU load, fan speed, GPS coordination, etc.), DNPs can be deployed to collect corresponding data as well. A few data plane objectives can be composed to form probes. These objectives are counter, meter, timer, timestamp, register, and table. Combining these with the packet filter through flow table entry configuration, one can easily monitor and catch arbitrary states on the data plane.

In practice, DNP can be considered a virtual concept. Its deployment can be done through either configuration or programming. For less flexible platforms, probes can be predefined but support on-demand runtime activation. Complex DNP functions can also be achieved through collaboration between data plane and control plane. Most common DNPs can be modeled for easy implementation. The goal is to make DNP implementation transparent to upper layer applications.

The simplest probe is just a counter. The counter can be configured to count bytes or packets and the counting can be conditional. The more complex probes can be considered as Finite State Machines (FSM) which are configured to capture specific events. FSMs essentially preprocess the raw stream data and only report the necessary data to subscribing applications.

Applications can use poll mode or push mode to access probes and collect data. The normal counter probes are often accessed via poll mode. Applications decide what time and how often the counter value is read. On the other hand, the complex FSM probes are usually accessed in push mode. When the target event is triggered, a report is generated and pushed to the application.

Timer is a special global resource. A timer can be configured to link to some action. When the time is up, the corresponding action is executed. For example, to get notification when a port load exceeds some threshold, we can set a timer with a fixed time-out interval, and link the timer to an action which reads the counter and generates the report packet if the condition is triggered. This way, the application avoids the need to keep polling statistics from the data plane.

With the use of global registers and state tables, more complex FSM probes can be implemented. For example, to monitor the half-open TCP connections, for each SYN request, we store the flow signature to a state table. Then for each ACK packet, the state table is checked and the matched entry is removed. The state table can be periodically polled to acquire the list of half-open connections. The application can also choose to only retrieve the counter of half-open connections. When the counter exceeds some threshold, further measures can be taken to examine if a SYN flood attack is going on.

Registers can be considered mini state tables which are good to track a single flow and a few state transitions. For example, to get the duration of a particular flow, when the flow is established, the state and the timestamp are recorded in a register; when the flow is torn down, the flow duration can be calculated with the old timestamp and the new timestamp. In another example, we want to monitor a queue by setting a low water mark and a high water mark for the fill level. Every time when an enqueue or a dequeue event happens, the queue depth is compared with the marks and a report packet is generated when a mark is crossed.

Some probes are essentially packet filters which are used to filter out a portion of the traffic and mirrored the traffic to the application or some other target port for further processing. There are two ways to implement a packet filter: use a flow table that matches on the filtering criteria and specify the associated action; or directly make a decision in the action. An example of the former case is to filter all packets with a particular source IP address. An example of the latter case is to filter all TCP FIN packets at the edge. Although we can always use a flow table to filter traffic, sometimes it is more efficient and convenient to directly work on the action. As being programmed by the application, the filtered traffic can be further processed before being sent. Two most common processes are digest and sample, both aiming to reduce the quantity of raw data. The digest process prunes the unnecessary data from the original packet and only packs the useful information in the digest packet. The sample process picks a subset of filtered traffic to send based on some predefined sampling criteria. The two processes can be used jointly to maximize the data reduction effect.

An application may need to install multiple DNPs in one device or across multiple devices to finish one data analytical task. For example, to measure the latency of any link in a network. We install a DNP on the source node to generate probe packets with timestamp. We install another DNP at the sink node to capture the probe packets and report both the source timestamp and the sink timestamp to the application for link latency calculation. The probe packets are also dropped by the sink DNP. The source DNP can be configured to generate probe packets at any rate. It can also generate just one probe packet per application request.

Using the similar idea, we can deploy DNPs to measure the end-to-end flow latency or trace exact flow paths. In this case, the DNPs can be deployed to enable the corresponding iOAM in-situ data collection service. At the path end, the DNP calculates the desired output based on the collected data.

Applications could have many such custom data requests. Each request lasts various time and consumes various network resources. Dynamic probe configuration or programming is not only efficient but also necessary. In summary, DNP is a versatile tool to prepare and generate just-in-time telemetry data for data analytical applications.

5.1. DNP Types

DNP can be roughly grouped into three types: node-based, path-based, and flow-based. Following is the list of DNPs. Some are atomic and the others can be derived from the atomic ones. Note that the list is by no means comprehensive. The list does not include the device state and status data that is steadily available. Depending on the device capability, more complex DNPs can be implemented. Applications can subscribe data from multiple DNPs to meet their needs. The flow-based data can be directly provided by iOAM data or derived from iOAM data.

5.1.1. Node Based

o Streaming Packets

   * Filter flow by user-defined flow definition.

   * Sample with user-defined sample rate. The sample can be based on interval or probability.

   * Generate packet digest with user defined format.

o Flow Counter

      * Associate poll-mode counter for user-defined flow.

      * Associate push-mode counter for user-defined flow. The counter value is pushed at user-defined threshold or interval.

   o Flow Meter

      * Associate poll-mode meter for user-defined flow.

      * Associate push-mode meter for user-defined flow. The meter value is pushed at user-defined threshold or interval.

   o Queue

      * Queue depth for designated queue is polled or pushed at user-defined threshold or interval.

      * Designated buffer depth is polled or pushed at user-defined threshold or interval.

   o Time

      * Time gap between user-defined flow packets is polled or pushed in streaming data or at user-defined threshold.

      * Arrival/Departure/Sojourn time of user-defined flow packets is polled or pushed streaming data or at user defined threshold.

   o Statistics

      * Number of active flows, elephant flows, and mice flows.

5.1.2. **Path Based**

   o Number of active flows per node on the path.

   o Path latency.

   o Round trip time of the path.

   o Node ID and ingress/egress port of the path.

   o Hop count of the path.

   o Buffer/queue depth of the nodes on the path.

   o Workload of the nodes on the path.

5.1.3. Flow Based

- o Flow Latency: Latency at each hop or cumulative E2E latency for user-defined flow.

- o Flow Jitter: Jitter at each hop or on the entire path for user-defined flow.

- o Flow Bandwidth: Bandwidth at each hop or the bottleneck bandwidth on the entire path for user-defined flow.

- o Flow Path Trace: Port and Node ID, and other data of the path for user-defined flow.

- o Proof of Transit (PoT) for particular set of nodes.

6. Interactive Query Architecture

In the past, network data analytics is considered a separate function from networks. They consume raw data extracted from networks through piecemeal protocols and interfaces. With the advent of user programmable data plane, we expect a paradigm shift that makes the data plane be an active component of the data analytics solution. The programmable in-network data preprocessing is efficient and flexible to offload some light-weight data processing through dynamic data plane programming or configuration. A universal network data analytics platform built on top of this enables a tight and agile network control and OAM feedback loop.

While DNP is a passive data plane data collection mechanism, we need to provide a query interface for applications to use the DNPs for data analytics. A proposed dynamic networking data analytical system architecture is illustrated in Figure 2. An application translates its data requirements into some dynamic transactional queries. The queries are then compiled into a set of DNPs targeting a subset of data plane devices (Note that in a less flexible target with predefined models, DNPs are configured). After the DNPs are deployed, each DNP conducts in-network data preprocessing and feeds the preprocessed data to the collector. The collector finishes the data post-processing and presents the results to the data-requesting application.

```
                  +------------------------------------+
                  |network data analytics applications |
                  +----------- ------------------------+
                              ^
                              V
                  +------------------------------------+
                  |dynamic and interactive query       |
                  +------------------------------------+
                         ^                   |
                         |                   V
                  +---------------+ +------------------+
                  |post process   | |DNP compile/config|
                  +---------------+ +------------------+
                         ^                   |
                         |                   V
                  +---------------+    +---------------+
                  |data collection|    |DNP deployment |
                  +---------------+    +---------------+
                      ^   ^   ^            |   |   |
                      |   |   |            V   V   V
                  +------------------------------------+
                  |network data plane                  |
                  |(in-network data preprocessing)     |
                  +------------------------------------+
```

Figure 2: Architecture of IQ with DNP

A query can be either continuous or one-shot. The continuous query may require the application to refine the existing DNPs or deploy new DNPs. When an application revokes its queries, the idle DNP resource is released. Since one DNP may be subscribed by multiple applications, the runtime system needs to keep track of the active DNPs.

7. Requirements for IQ with DNP

This section lists the requirements for interactive query with DNP:

- Applications should conduct interactive query through a standard interface (i.e., API). The system is responsible to compile the IQ into DNPs and deploy the DNPs to the corresponding network nodes.

- DNPs can be deployed through some standard south bound interface and protocols such as gRPC, NETCONF, etc.

- The interactive query should not modify the forwarding behavior. The API should provide the necessary isolation.

- The deployed DNP should not lower the forwarding performance of the data plane devices. If the DNP would affect the forwarding performance, the query should be denied.

- The system should support multiple parallel queries from multiple applications.

- One application can deploy different DNPs to a set of network nodes and these DNPs work jointly to finish a function.

- DNP may be revoked and preempted by the controller due to resource conflict and application priority.

8. Considerations for IQ with DNP

8.1. Technical Challenges

Some technical issues need to be addressed to realize interactive query with DNP on general network data plane:

- Allowing applications to modify the data plane has security and safety risks (e.g., DoS attack). The counter measure is to supply a standard and safe API to segregate applications from the runtime system and provide applications limited accessibility to the data plane. Each API can be easily compiled and mapped to standard DNPs. An SQL-like query language which adapts to the stream processing system might be feasible for the applications.

- When multiple correlated DNPs are deployed across multiple network devices or function blocks, or when multiple applications request the same DNPs, the deployment consistency needs to be guaranteed for correctness. This requires a robust runtime compiling and management system which keeps track of the subscription to DNPs and controls the DNP execution time and order.

- The performance impact of DNPs must be evaluated before deployment to avoid unintentionally reducing the forwarding throughput. Fortunately, the resource consumption and performance impact of standard DNPs can be accurately profiled in advance. A device is usually over provisioned and is capable of absorbing extra functions up to a limit. Moreover, programmable data plane allows users to tailor their forwarding application to the bare bones so more resources can be reserved for probes. The runtime system needs to evaluate the resulting throughput performance before committing a DNP. If it is unacceptable, either some old DNPs need to be revoked or the new request must be denied.

- o While DNP is relatively easy to be implemented in software-based platform (e.g., NPU and CPU), it is harder in ASIC-based programmable chips. Architectural and algorithmic innovations are needed to support a more flexible pipeline which allows new pipeline stage, new tables, and new custom actions to be inserted at runtime through hitless in-service updates. An architecture with shared memory and flexible processor cores might be viable to meet these requirements. Alternatively, DNPs can be implemented using an "out-of-band" fashion. That is, the slow path processor is engaged in conjunction with the forwarding chip to complete the DNP function.

8.2. Standard Consideration

The query API can be potentially standardized. The actually DNP deployment interface may consider to reuse or extend the IETF standards and drafts such as gRPC [I-D.talwar-rtgwg-grpc-use-cases] and NETCONF [RFC6241]. We may also define standard telemetry YANG [RFC6020] models for common DNPs so these DNPs can be used in a configurable way.

9. Security Considerations

Allowing applications to modify the data plane has security and safety risks (e.g., DoS attack). The counter measure is to supply standard and safe API to segregate applications from the runtime system and provide applications limited accessibility to the data plane. Each API can be easily compiled and mapped to standard DNPs. An SQL-like query language which adapts to the stream processing system might be feasible and secure for the applications.

10. IANA Considerations

This memo includes no request to IANA.

11. Acknowledgments

The authors would like to thank Frank Brockners, Carlos Pignataro, Tom Tofigh, Bert Wijnen, Stewart Bryant, James Guichard, and Tianran Zhou for the valuable comments and advice.

12. Informative References

[DOI_10.1145_2491185.2491190]
Song, H., "Protocol-oblivious forwarding", Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13 , DOI 10.1145/2491185.2491190, 2013.

[DOI_10.1145_2656877.2656890]
Bosshart, P., Varghese, G., Walker, D., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., and A. Vahdat, "P4", ACM SIGCOMM Computer Communication Review Vol. 44, pp. 87-95, DOI 10.1145/2656877.2656890, July 2014.

[I-D.brockners-inband-oam-data]
Brockners, F., Bhandari, S., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mizrahi, T., Mozes, D., Lapukhov, P., and R. <>, "Data Formats for In-situ OAM", draft-brockners-inband-oam-data-02 (work in progress), October 2016.

[I-D.brockners-inband-oam-requirements]
Brockners, F., Bhandari, S., Dara, S., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mozes, D., Mizrahi, T., <>, P., and r. remy@barefootnetworks.com, "Requirements for In-situ OAM", draft-brockners-inband-oam-requirements-02 (work in progress), October 2016.

[I-D.talwar-rtgwg-grpc-use-cases]
Specification, g., Kolhe, J., Shaikh, A., and J. George, "Use cases for gRPC in network management", draft-talwar-rtgwg-grpc-use-cases-01 (work in progress), January 2017.

[RFC3176] Phaal, P., Panchen, S., and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", RFC 3176, DOI 10.17487/RFC3176, September 2001, <http://www.rfc-editor.org/info/rfc3176>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <http://www.rfc-editor.org/info/rfc7011>.

Authors' Addresses

Haoyu Song (editor)
Huawei Technologies Co., Ltd
2330 Central Expressway
Santa Clara, 95050
USA

Email: haoyu.song@huawei.com

Jun Gong
Huawei Technologies Co., Ltd
156 Beiqing Road
Beijing, 100095
P.R. China

Email: gongjun@huawei.com

Hongfei Chen
Huawei Technologies Co., Ltd
156 Beiqing Road
Beijing, 100095
P.R. China

Email: chenhongfei@huawei.com