Abstract

This document standardizes both 1) a means of requesting a stream of packet samples from any device generating, routing, or forwarding traffic, and 2) receiving metadata information from the network element about these packet samples, and the structure of said stream metadata. A main design requirement is to provide network elements with widely varying capabilities (e.g., ASICs, NPUs, NICs, vSwitches, CPUs) a mechanism to sample and export packets at high rates, by allowing communication of the specific bit formats of internal data headers applied to the packet flow, in a way that enhances interoperability between traffic sources and sinks. Historically, Netflow and similar mechanisms have been used for these use cases; however, the increasing packet rates of very high-speed devices and increasing variance in the information available to data planes lends itself to both a less-prescriptive set of packet formats as well as a decoupling of the sampling action from the collection and analysis mechanisms.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 May 2020.
1. Introduction
1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

The following terms are used within this document:

Client: The device configuring the Replicator.

Receiver: The device receiving the packet stream.

Replicator: The device performing the actual packet replication, as requested by a Client, and sending the resulting replicated packet stream to a Receiver.

Point: The location inside the Replicator (e.g., a forwarding ASIC) that performs the actual packet replication. There may be multiple physical interfaces serviced by one Point, or one interface may be serviced by multiple Points, that may have different capabilities.

1.3. Motivation for Disaggregation of Telemetry

A key concept for this proposal is to enable very high rate sample generation for network elements, while at the same time separating the sampling mechanism itself from specific analysis or transport protocols. If we separate the component functions of how these problems have been traditionally solved, these functions lend themselves to being viewed as a layered stack such as the one in the figure:

Figure: Packet sampling and analysis viewed as a layered stack
The primary advantage of the stack model is the ability to disaggregate functions from each other. For example, providing a self-describing, flexible format for the metadata abstracts the data plane -- in other words the upper layers do not need to know how many bits wide a metadata field is, they only need to know that it is present and the semantics. Separating the transport function allows for multiple use cases: a router wishing to sample packets for internal consumption within the same system might use a locally defined (perhaps even proprietary) transport header, while putting the sampled metadata and packet into a UDP packet allows for it to be transported to any IP-reachable collector, regardless of the geographic or topological distance from the Replicator itself.

This document standardizes the "Sampling / Metadata" and "Export / Transport" components of the above stack.

2. Use Cases

This document is designed around the following current and foreseeable use cases that operators have today.
2.1. Use Case 1: Traffic Analytics

Operators typically use a mix of NetFlow, IPFIX, and inline traffic samplers spread throughout the network to gather data for analytics. With the next generation of hardware, 400Gb/s interfaces are becoming available, with higher data rates under development in their respective standards bodies. This will require at least an augmentation of any inline traffic samplers, which are quite expensive. Additionally, the pace of growth in the data plane is outgrowing the pace of growth of the control plane. This is especially visible with relatively control plane or CPU-heavy protocols such as NetFlow, where current sampling rates are simply not going to be sustainable long-term, primarily due to on-box control plane hardware limitations. Being able to capture a filtered, sampled collection of actual packets throughout the network is very valuable for understanding how the network is being used, to provide hard data to justify network topology augments and/or technology changes.

This proposal addresses this use case by: 1) making the data replication mechanism as simple as possible, reducing the need for high levels of complexity in the data plane; 2) decoupling the sampling/collection of packets from the analysis, which in turn allows for the analysis to be performed on distributed, horizontally-scalable platforms rather than being constrained to the compute and storage capabilities of a local network element.

2.2. Use Case 2: Network Behavior Verification

This use case focuses on the potential ability to have the ASICs stream discarded packets, along with an indication as to the reason for the drop. With fields denoting the reason for dropped packets such as QoS policies, buffer contention, ACLs, etc., such discarded traffic could be streamed (potentially at a sampling rate of 1:1, i.e. every packet) off-box for analysis to determine if the observed behavior was expected, or trigger alerts that QoS policies may be having adverse effects on the network. The ability to include the packet payload provides additional context, allowing examination of the platform behavior and affected policies.

This proposal addresses this use case by allowing samplers which have such capabilities to communicate to the receiver: 1) drop codes(reasons) that are known, 2) the semantics of those codes, and 3) the specific bit formats for the receiver to use when decoding.
2.3. Use Case 3: Standardization

Standardizing the way these data streams are formed and communicated between the Replicators, Clients, and Receivers in a fashion that allows vendors flexibility in what work the ASIC has to do to support sampled streaming (by allowing communicating of an extremely dynamic header in a manner than control planes can manage) allows systems to be used between all platforms in an interoperable fashion. The alternative is to build independent systems for each packet replication solution that may end up being developed, resulting in much higher costs for an overall solution.

This proposal addresses this use case by allowing the sampled packet header to provide varying metadata fields, without mandating specific positions or widths. This arrangement of fields and their format is a function of the Replicator, and information about how to handle this data is exchanged between the Replicator, Client, and Receiver at the initialization of the session. The motivation for such latitude in encoding and sizing is quite intentional, as it permits widely varying capabilities within the Replicators.

2.4. Use Case 4: Security Automation

An automated security platform can utilize this proposal to set up a "normal security analysis" stream at a very low sampling rate (for example, 1 in 20,000) for constant monitoring at various points throughout the network. Upon seeing something it deems ‘interesting’, or by manual input, it can add in an additional, targeted, stream, at a very high sampling rate (potentially 1:1) for detailed analysis and mitigation efforts.

Examples of past incidents where this may have been useful are the NTP MONLIST attacks, DNS attacks, or DDoS attacks (although 1:1 would most likely not be used in a DDoS case, unless performing the initial data collection).

The security platform could potentially then use the collected packets to generate an auto-mitigation plan based on heuristics (i.e., 99% of this sudden burst of traffic has something in common, deploy mitigation targeting that.)
3. Stream Setup

The configuration and setup between the Client and the Replicator utilizes the YANG model as listed in Appendix B and any supported configuration method (NETCONF, RESTCONF, gRPC, etc.). The tree output of this model, as provided in Appendix A is provided as an aid to understanding the interactions and tree structure as described in this document.

3.1. Client queries Replicator for Points

A Client MUST first request from the Replicator the available configurations via the ‘points’ branch, which provides the following information:

* ‘name’ - The name of the Point. This serves as a key, and SHOULD NOT be interpreted by software as anything other than a possibly-human-readable uniquely identifying value. A Replicator MAY choose to use an internal path, an encoded address, or any other value of its choosing.

* ‘interfaces’ - The physical interfaces this Point is servicing. A Replicator MAY offer the same interfaces under different Points, with a different set of options. A Replicator MAY not offer a Point for every interface available on the system.

* ‘filters’ - What filters can be applied (for example, against certain IP fields, against parts of the frame, etc.). A Replicator MAY not be able to honor every combination of filters submitted in a request, or MAY not offer any filtering capability at all. A Replicator MAY only be able to support a limited number of filters, which MAY be returned in in the ‘max-filters’ branch.

* ‘min-ratio’ and ‘max-ratio’ - Minimum and maximum sampling rates possible at this point. These are provided as a number N, denoting one sample will be returned for every N. A Replicator MAY not be able to offer a ‘min-ratio’ of 1 (i.e. every packet).

* ‘samplers’ - A list of any current samplers already active on this Point as requested by this Client, and the branch manipulated in the next section. A Replicator SHOULD NOT inform a Client about the sampling sessions from other Clients.

* Optionally, the maximum frame length the Point can replicate into the sample in ‘max-frame-length-copy’.

* Optionally, the maximum offset into a frame the Point can inspect in ‘max-frame-depth-inspect’.
* Optionally, the maximum number of samplers that this Point can accommodate in 'max-samplers'. A Client MUST still check for success, as highly complex filters may reduce the amount of replication the Point can do from this stated maximum.

3.2. Client submits a request to the Replicator

The Client then can request one or more streams to be set up on the Replicator, taking into consideration the provided information. This is performed by sending a request via adding an entry to the 'samplers' list in the 'points' branch and filling in the parameters listed below:

* 'name' MUST be unique in the list, and MAY be any valid string value up to 255 characters. The Replicator MUST isolate namespaces between Clients (as one Client SHOULD NOT be able to see other Clients' entries).

* 'destination' sets the transport mechanism and Receiver address. It should be noted that the Client and Receiver MAY be separate devices. The mechanism of exchanging information between the Client and Receiver about this setup process is outside the scope of this document. At present, the only supported transport mechanism is a UDP tunnel, as detailed below in Section 4.

* 'client-heartbeat' MUST be set to 0.

* The desired sampling rate ('ratio'), along with what degree of variance the Client can accept ('min-ratio' and 'max-ratio'). For example, the client may request a 1 in 2000 rate, but specify a range in the variance of 1900-2100. A proposal may come back with the sampling rate offered of 1 in 2048, due to restrictions on the Replicator.

* Optionally, one, or more filters in the 'filters' container, as seen in the 'filter-type' typedef in the Yang model. Generally, a Client would filter at least on a specific interface and direction, but many other filter options are possible.

When the client is done with its configuration, it MUST set 'status' to the 'client-request-complete' value, and the 'request' branch MUST be read-only from this point forward.
3.3. Replicator offers Proposals

Upon receiving the 'status' change to 'client-request-complete', the Replicator updates the 'proposals' branch. This branch details zero, one, or more ways the Replicator can fulfill the sampling request. While generally there will only be zero or one proposals, a Replicator MAY offer more. For example, matching a sampling rate exactly would result in performance loss but a 'close enough' option can be offered that does not, or offers of what headers can be captured in the resulting stream. Each proposal includes a unique ID number, allowing the Client to select one, as detailed below.

If the Replicator is unable to provide any Proposals, the 'proposals' list MUST be empty, a human-readable error message MAY be returned in the 'proposal-error' field, then the 'status' field MUST be set to 'replicator-proposal-error'.

If the Replicator was able to provide Proposals, it MUST set the 'status' field to 'replicator-proposals-available' when it is finished, and the 'proposals' branch MUST be read-only until the Client finishes the Proposal selection step below.

Part of each Proposal is a 'stream-format' branch, which informs the Client of the packet format the Receiver will be receiving. This format completely defines the entirety of the resulting data flow format besides the outer UDP wrapper - there is no normative format. A couple non-normative examples of what may result are provided in Section 4.

To adequately addresses the use cases stated above, a Replicator SHOULD support as a minimum set of capabilities:

* An action field that denotes a pass or drop (ideally with drop reason)

* Capturing at least 128 octets of payload

* The original frame length

* Sampling rates up to 1:1 (i.e. every packet is replicated), and down to 1:20000 or smaller.

* Having different sampling sessions having different sampling rates (to allow a "general" session to be watching a broad selection of traffic, and more specific sessions targeting exact flows or situations)

* At least two sessions per physical interface
* Filtering on ingress port
* Filtering on action
* Filtering on direction of traffic

3.4. Client selects a Proposal

Upon either a notification or detection that the 'status' field has been updated, the Client then may then set the 'proposal-selected' entry to the value of the desired ID offered in 'proposals', and then set 'status' to 'client-proposal selected'. At this point, the Replicator:

* MAY remove unnecessary branches in the 'proposals' list, but MUST retain the selected one.
* MUST either install the requested sampling stream if possible, then MUST set 'status' to 'replicator-install-success'. If it cannot, it MAY set 'install-error' to a human-readable error message and MUST set 'status' to 'replicator-install-error'.

3.5. Ending sampling and cleanup

When a Client is finished with a sampling session, it deletes its entry in the 'samplers' tree to terminate a sampling session. Otherwise, a Client MUST refresh its entry by setting 'client-heartbeat' to 0 at least every 3600 seconds. The 'client-heartbeat' is then incremented by the Replicator. If 'client-heartbeat' exceeds 3600, the Replicator SHOULD consider the sampling configuration and any associated sampling session no longer necessary, terminate the sampling, and delete the entry. A Replicator MAY allow configuration to increase this timeout.

4. Data Stream Format

After the stream setup has been completed, the Receiver MUST use the stream-format data that the Replicator has calculated in its proposal. The Client and Receiver MUST NOT assume that the stream-format data is consistent between one stream setup and any other (there may be different versions of ASICs, different capabilities, different versions of operating systems, or different filters may yield a different format), or that the payload is always at the end (it could appear at the beginning or in the middle, and sufficient data is provided by the other fields to extract the data correctly). The stream-format data provides the Client with what information is provided at what location in the resulting packet. The Replicator MUST follow the expectation that is provided in these fields.
There is one captured packet per encapsulated packet, and thus the outer encapsulation length can be used to deduce the length of one variable-length field (designated by a field length of 0) contained within. If there is more than one variable-length field, a matching "-size" field type MUST be provided for all but one of the variable-length fields (as a single variable length can be deduced from the wrapper length).

This means there is no normative packet format or data layout - a large point of this specification is to allow that packet format to be negotiated and decided between the Client and Replicator, with the information passed back via the stream-format data.

One example of what the resulting packet may look like (but not a normative listing of what it is - the actual format can be any combination of fields, of any size, in any order), the data inside the resulting data stream after the UDP tunnel header may look like the following:

Example 1: Packet layout

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----------------------------------------------
| Incoming Port | Timestamp                                       |
+---------------+-----------------------------------------------+
| Act           | Frame Length | Internal Data 1                           |
+---------------+---------------+--------------------------------------------+
|                | Payload       |
+---------------+-----------------------------------------------+

Figure 2
```

This non-normative example may be associated with a stream-format as per the following table:

```
+-----------+-------+------------------------+----------------------+
| Field     | Field | Field Type             | Field Type-Data      |
| Name      | Size  |                        |                      |
+===========+=======+========================+======================+
| Incoming  | 8     | port-ingress           | A listing of         |
| port      |       |                         | values that may be   |
|           |       |                         | seen in this         |
|           |       |                         | field, mapped to     |
|           |       |                         | interface-refs       |
```

Gray & Wobker
Expires 7 May 2020
[Page 11]
<table>
<thead>
<tr>
<th>Timestamp</th>
<th>24</th>
<th>timestamp-nsec-ingress</th>
<th>Two 32-bit numbers giving when the &quot;0&quot; of this field is based off of, using the PTP Truncated Timestamp format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act</td>
<td>2</td>
<td>action</td>
<td>A listing of values that may be seen in this field, mapped to action types (accepted, dropped, etc.)</td>
</tr>
<tr>
<td>Frame Length</td>
<td>17</td>
<td>frame-length-ingress</td>
<td>Note that this denotes the original frame length - the payload field MAY not include the entire payload.</td>
</tr>
<tr>
<td>Internal Data 1</td>
<td>13</td>
<td>padding</td>
<td>Note that this may be ASIC-internal-only data, or some other information that would be expensive to prune out. ‘padding’ fields MUST have all content ignored.</td>
</tr>
<tr>
<td>Payload</td>
<td>0</td>
<td>frame-payload-ingress</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Example 1: Stream-format data

Another non-normative example, which is similar to the [I-D.tuexen-opsawg-pcapng] enhanced packet block (EPB) format (and thus, this Replicator may in fact be a server offering a tcpdump-based backend using this frontend):
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Size</th>
<th>Field Type</th>
<th>Field Type-Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface ID</td>
<td>32</td>
<td>port</td>
<td>A listing of values that may be seen in this field, mapped to interface.refs from [RFC8343].</td>
</tr>
<tr>
<td>Timestamp</td>
<td>64</td>
<td>timestamp-msec</td>
<td>Two 32-bit numbers giving when the &quot;0&quot; of this field is based off of, using the PTP Truncated Timestamp format.</td>
</tr>
<tr>
<td>Captured Packet Length</td>
<td>32</td>
<td>frame-payload-size</td>
<td>Note: This allows us to have the Options field as our real variable length field.</td>
</tr>
<tr>
<td>Original Packet Length</td>
<td>32</td>
<td>frame-length</td>
<td></td>
</tr>
<tr>
<td>Packet Data</td>
<td>0</td>
<td>frame-payload</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>0</td>
<td>padding</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Packet-format response example 2

To restate the prior note, the above is purely an example of what the format could be - the actual format used is negotiated between the Client and Replicator, and can have practically any layout, with any additional fields.

A Client SHOULD take efforts to be notified when a change has occurred on the Replicator (e.g., port or line card changes, device reboot, etc.), and re-verify and re-apply as needed its sampled streaming configurations when such a change is detected.
5. IANA Considerations

This document defines a new UDP port number, entitled "Sampled Streaming", and assigns a value of TBD1 from the Service Name and Transport Protocol Port Number Registry

https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml:

+------|-------------------+
| Tag  | Description       |
+======|===================+
| TBD1 | Sampled Streaming |
+------|-------------------+

Table 3

This document requests registration of a URI in the "IETF XML Registry" RFC 3688 [RFC3688]. Following the format in RFC 3688, the following registration is suggested:

Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the "YANG Module Names" registry RFC 6020 [RFC6020]:

name: ietf-sampled-streaming
prefix: ss
reference: This document

6. Security Considerations

Vendors and deployments must take into consideration that this functionality allows a mirroring of traffic, with configurable destinations and filters. Similar functionality already exists in various remote packet mirroring systems, and similar considerations should be taken. Filters utilizing the source port of TBD1 SHOULD be applied at the edges of a provider’s network to provide an additional layer of security.

A Replicator SHOULD ensure that Clients can only see their own entries in the ‘samplers’, and MUST ensure that once a Client has created an entry in the samplers list, only that same Client may re-query or make changes to it.
7. Acknowledgments

The authors would like to thank Joe Clarke, Marek Hajduczenia, Brian Harber, Paolo Lucente, Jim Rampley, and Dmytro Shytyi for their reviews and providing helpful suggestions and feedback of this draft.

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Yang Model Tree Reference

```
module: ietf-sampled-streaming
  +--rw points* [name]
  |    +--rw name string
  |    +--ro interfaces* []
  |    |    +--ro if if:interface-ref
  |    +--ro filters* []
  |    |    +--ro filter filter-type
  +--ro min-ratio uint32
  +--ro max-ratio uint32
```

Gray & Wobker  Expires 7 May 2020
Appendix B. Yang Model

module ietf-sampled-streaming {
    prefix ss;

    import ietf-interfaces {
        prefix if;
    }

    import ietf-inet-types {
        prefix inet;
    }

    organization "IETF Working Group";
    contact "Editor: Andrew Gray
    <mailto:Andrew.Gray@charter.com>";
    description "This module contains a collection of YANG definitions for
managing sampled streaming subscriptions.

Copyright (c) 2019 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC XXXX (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself for full legal notices.

The key words ‘MUST’, ‘MUST NOT’, ‘REQUIRED’, ‘SHALL’, ‘SHALL NOT’, ‘SHOULD’, ‘SHOULD NOT’, ‘RECOMMENDED’, ‘NOT RECOMMENDED’, ‘MAY’, and ‘OPTIONAL’ in this document are to be interpreted as described in BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here."

revision 2019-10-22 {
  description
    "Updates based on feedback for -02 draft: Adding more forwarded action-types. frame-payload changed to be explicit about direction. Added -size types explicitly for frame-payload and padding to allow using more than one zero-length field.";
  reference
    "draft-gray-sampled-streaming-02";
}
revision 2019-08-06 {
  description
    "Updates based on feedback for -01 draft.";
  reference
    "draft-gray-sampled-streaming-01";
}
revision 2019-06-25 {
  description
    "Initial version.";
  reference
    "draft-gray-sampled-streaming-00";
}
typedef filter-type {
  type enumeration {
    enum interfaces {
      description
    }
  }
}
"List of interfaces to filter against."

enum action {  
    description  
    "Filter against a list of actions that the Point took (i.e. 
        only consider packets that were actually forwarded).";
}

enum direction {  
    description  
    "Direction to sample traffic in.";
}

enum ip-version {  
    description  
    "The version number in the IP header.";
}

enum ip-v4-srcip {  
    description  
    "The IPv4 header’s source IPv4 address.";
}

enum ip-v4-dstip {  
    description  
    "The IPv4 header’s destination IPv4 address.";
}

enum ip-v4-ttl {  
    description  
    "The IPv4 header’s Time to Live.";
}

enum ip-v4-prot {  
    description  
    "The IPv4 header’s protocol number.";
}

enum ip-v6-srcip {  
    description  
    "The IPv6 header’s source IPv4 address.";
}

enum ip-v6-dstip {  
    description  
    "The IPv6 header’s destination IPv4 address.";
}

enum frame-size {  
    description  
    "The total size of the frame.";
}

enum frame-payload {  
    description  
    "Specific payload octets.";
}

enum frame-length {
typedef field-type {
    type enumeration {
        enum padding {
            description
            "Padding bits that MUST be ignored.";
        }
        enum padding-size {
            description
            "This packet’s length of a variable-length padding field.";
        }
        enum port {
            description
            "An indication of the port the traffic was sampled from.";
        }
        enum direction {
            description
            "Which direction the traffic went.";
        }
        enum port-ingress {
            description
            "What port the traffic was received from (may be different
            than ’port’)";
        }
        enum port-egress {
            description
            "What port the traffic is leaving on (may be different than
            ’port’)";
        }
        enum timestamp-msec-ingress {
            description
            "The timestamp the packet was received at, in integer
            milliseconds. The epoch of this number is provided in the
            timestamp container of the returned field information.";
        }
        enum timestamp-usec-ingress {
            description
            "The timestamp the packet was received at, in integer
            microseconds. The epoch of this number is provided in the
            timestamp container of the returned field information.";
        }
    }
}
enum timestamp-nsec-ingress {
    description
    "The timestamp the packet was received at, in integer
    nanoseconds. The epoch of this number is provided in the
timestamp container of the returned field information.";
}

enum timestamp-msec-egress {
    description
    "The timestamp the packet left the point at, in integer
    milliseconds. The epoch of this number is provided in the
timestamp container of the returned field information.";
}

enum timestamp-usec-egress {
    description
    "The timestamp the packet left the point at, in integer
    microseconds. The epoch of this number is provided in the
timestamp container of the returned field information.";
}

enum timestamp-nsec-egress {
    description
    "The timestamp the packet left the point at, in integer
    nanoseconds. The epoch of this number is provided in the
timestamp container of the returned field information.";
}

enum frame-length {
    description
    "The generic frame length. Note that due to chipset
    capabilities, this MAY not be the same as the captured
    packet length.";
}

enum frame-length-ingress {
    description
    "The frame length as received by the point. Note that due
to chipset capabilities, this MAY not be the same as the
    captured packet length.";
}

enum frame-length-egress {
    description
    "The frame length after local processing, as it leaves the
    point. Note that due to chipset capabilities, this MAY
    not be the same as the captured packet length.";
}

enum frame-payload-size {
    description
    "The length of the payload that has actually been copied
    into this stream.";
}

enum frame-payload-ingress {
typedef action-type {
  type enumeration {
    enum forwarded {
      description
      "Generically forwarded normally through the system. A more specific action type code SHOULD be used.";
    }
    enum forwarded-label-change {
      description
      "Forwarded, with a generic MPLS label change having occurred.";
    }
    enum forwarded-label-swap {
      description
      "Forwarded, with a MPLS label swap.";
    }
    enum forwarded-label-pop {
      description
      "Forwarded, with a MPLS label pop.";
    }
    enum forwarded-label-push {
      description
      "Forwarded, with a MPLS label push.";
    }
    enum forwarded-cpu-punt {
      description
      "Forwarded after a CPU punt.";
    }
    enum forwarded-tunnel {
      description
      "Forwarded after a tunnel.";
    }
  }
}
"Forwarded with additional outer wrapper for tunneling."

enum forwarded-tunnel-frr {
    description
    "Forwarded with additional outer wrapper due to fast reroute.";
}
enum dropped {
    description
    "Generically dropped. A more specific action type code SHOULD be used.";
}
enum dropped-rate-limit {
    description
    "Dropped due to a rate limiter applied.";
}
enum dropped-buffer {
    description
    "Dropped due to no buffer space.";
}
enum dropped-security {
    description
    "Dropped due to a security policy.";
}
enum dropped-error {
    description
    "Dropped due to the frame being in error.";
}
enum dropped-cpu-punt {
    description
    "Dropped after a CPU punt.";
}
enum passed-to-cpu {
    description
    "Passed on to the CPU, but what the CPU did with it is unknown.";
}

description
    "Possible actions taken on a packet.";

typedef destination-type {
type enumeration {
    enum udp {
        description
        "Sent with a UDP header.";
    }
}}
typedef direction-type {
    type enumeration {
        enum ingress {
            description
            "Traffic flowing into the Replicator.";
        }
        enum egress {
            description
            "Traffic flowing out of the Replicator.";
        }
        enum both {
            description
            "Capture both ingress and egress traffic.";
        }
    }
    description
    "Different possible direction types.";
}

typedef status-type {
    type enumeration {
        enum client-request-complete {
            description
            "The Client has completed its request setup.";
        }
        enum replicator-proposals-available {
            description
            "The Replicator has finished processing the request, and
            has proposals available in the ‘proposals’ branch.";
        }
        enum replicator-proposal-error {
            description
            "The Replicator encountered an error attempting to come up
            with a proposal. ‘proposal-error’ MAY contain an
            explanation.";
        }
        enum client-proposal-selected {
            description
            "The Client has updated ‘proposal-selected’ and is ready
            for the Replicator to install the requested sampling.";
        }
        enum replicator-install-success {
            description
            "The Replicator has installed the requested sampling.";
        }
    }
    description
    "Different possible status types.";
}
"The Replicator has successfully activated the sampling, and it is operating."
}
enum replicator-install-error {
    description
        "The Replicator encountered an error installing the sampling. ‘install-error’ MAY contain an explanation.";
    }
}
description
    "The status of a sampler entry."
}

typedef frame-headers {
        type bits {
            bit eth-l1-preamble {
                position 0;
                description
                    "Will include the Ethernet preamble.";
            }
            bit eth-l1-sof {
                position 1;
                description
                    "Will include the Ethernet start of frame delimiter";
            }
            bit eth-l2-dmac {
                position 2;
                description
                    "Will include the outer Ethernet destination MAC.";
            }
            bit eth-l2-smac {
                position 3;
                description
                    "Will include the outer Ethernet source MAC.";
            }
            bit eth-l2-vlan {
                position 4;
                description
                    "Will include any 802.1Q-2018 VLAN tags.";
            }
            bit eth-l2-type {
                position 5;
                description
                    "Will include the Ethertype or size.";
            }
            bit eth-l2-fcs {
                position 6;
description
"Will include the Frame Check Sequence after the payload.";
}

bit eth-l1-ipg {
  position 7;
  description
  "Will include the inter-packet gap. Be aware that different Ethernet speeds may have different lengths.";
}

bit mpls-tags {
  position 8;
  description
  "Will include MPLS tags."
}
}

description
"Listing of fields to be provided in a frame capture."
}

grouping filters {
  description
  "Filter definition. Multiple filters are ANDed."
  leaf name {
    type string {
      length "1..255";
    }
    description
    "A name for this filter."
  }
  list interfaces {
    when "./.type = 'interfaces'";
    key "int";
    description
    "Filter down to only this list of interfaces."
    leaf int {
      type if:interface-ref;
      description
      "A specific interface to filter against."
    }
  }
  list actions {
    when "./.type = 'action'";
    key "action";
    description
    "Filter down to only this list of actions."
    leaf action {
      type action-type;
    }
  }
}
description
"One specific action code."
}
)
list directions {
when "./type = 'direction'"
key "direction"
description
"Which direction(s) to sample traffic in."
leaf direction {
  type direction-type;
description
  "Specific direction code."
}
)
leaf type {
type filter-type;
mandatory true;
description
  "The type of filter associated."
}
leaf ipv4-address {
when "./type = 'ip-v4-srcip' | ./type = 'ip-v4-dstip'"
type inet:ipv4-address-no-zone;
description
  "The IPv4 address to filter on."
}
leaf ipv6-address {
when "./type = 'ip-v6-srcip' | ./type = 'ip-v6-dstip'"
type inet:ipv6-address-no-zone;
description
  "The IPv6 address to filter on."
}
leaf version {
when "./type = 'ip-version'"
type inet:ip-version;
description
  "The value of the IP version number to match on."
}
container frame-payload {
when "./type = 'frame-payload'"
description
  "Frame payload fragment to match on."
leaf offset {
  type uint16;
description
  "offset in octets from the start of the frame to begin the match on."
}
leaf match {
  type binary;
  description
    "The bytes to match on.";
}

leaf frame-length {
  when "../type = 'frame-length'";
  type uint16;
  description
    "Frame length to match on.";
}

grouping stream-format {
  description
    "This contains the packet format data that this sampling stream
     is sending. This is only valid after configuration. The
     length fields are given in bits, and are consecutive. Needed
     gaps should use a ‘padding’ element.";
  list fields {
    key "name";
    description
      "The listing of the fields that will be encapsulated and sent
       to the receiver.";
    leaf name {
      type string {
        length "1..255";
      }
      description
        "Human readable name of what this field contains.";
    }
    leaf size {
      type uint32 {
        range "0..524280";
      }
      description
        "The size of this field, in bits. The value of ‘0’ denotes
         a variable-sized field.";
    }
    leaf type {
      type field-type;
      description
        "The type of this data.";
    }
  list action-mappings {
    when "../type='action'";
key "value";
  description
  "The mapping of values to action-type codes, valid for
type=action.";
leaf value {
  type binary;
  description
  "The value that will appear in the header.";
}
leaf meaning {
  type action-type;
  description
  "What this value indicates.";
}
}
list port-mappings {
  when "../type=ingress-port" | ../type=egress-port";
  key "value";
  description
  "The mapping of values to interfaces, valid for
type=ingress-port or type=egress-port";
leaf value {
  type binary;
  description
  "The value that will appear in the header.";
}
leaf port {
  type if:interface-ref;
  description
  "The port the value maps to.";
}
}
list direction-mappings {
  when "../type=direction";
  key "value";
  description
  "The mapping of values to direction codes, valid for
type=direction.";
leaf value {
  type binary;
  description
  "The value that will appear in the header.";
}
leaf direction {
  type direction-type;
  description
  "The direction the traffic in respect to the port.";
}
container timestamp {
    when "../type='timestamp-nsec' | ../type='timestamp-usec' | ../type='timestamp-msec'";
    description "Supplemental data for type=timestamp*, in PTP Truncated Timestamp Format. Provides the time used as the epoch for the number in the data stream."
    leaf seconds {
        type uint32;
        description "Specifies the integer portion of the number of seconds since the epoch."
    }
    leaf nanoseconds {
        type uint32;
        description "Specifies the fractional portion of the number of seconds since the epoch, in integer number of nanoseconds."
    }
}
leaf payload-contents {
    when "../type='frame-payload-ingress' | ../type='frame-payload-egress'";
    type frame-headers;
    description "Details about what parts of the frame this payload field SHOULD contain. Note carefully the 'SHOULD' - for a variety of reasons (different forwarding paths, exception handling, etc.), the actual headers of any one frame MAY be different than this."
}
list points {
    key "name";
    description "A listing of the capture points available on this device, what ports they provide for, and what filtering is available at those points."
    leaf name {
        type string {
            length "1..255";
        }
        description "The name of this capture point."
    }
}
list interfaces {
    config false;
    description "List of interfaces that are available at this point."
    leaf if {
        type if:interface-ref;
        mandatory true;
        description "An interface tied to this capture point."
    }
}

list filters {
    config false;
    description "List of filtering options available at this point."
    leaf filter {
        type filter-type;
        mandatory true;
        description "One specific filter available at this point."
    }
}

leaf min-ratio {
    type uint32 {
        range "1..max"
    }
    config false;
    mandatory true;
    description "The minimum sampling ratio (1:N, with N being this value) this point can provide."
}

leaf max-ratio {
    type uint32 {
        range "1..max"
    }
    config false;
    mandatory true;
    description "The maximum sampling ratio (1:N, with N being this value) this point can provide."
}

leaf max-samplers {
    type uint32;
    config false;
    description "The maximum number of additional samplers that can be
installed at this point.
}

leaf max-filters {
    type uint32;
    config false;
    description
        "The maximum number of filtering rules permitted at this
         location. Note this is an absolute maximum, and fewer rules
         that are complex may still be rejected by the device."
}

leaf max-frame-length-copy {
    type uint16;
    config false;
    description
        "The maximum size that the point can replicate and copy into
         the header."
}

leaf max-frame-depth-inspect {
    type uint16;
    config false;
    description
        "The offset of the last octet in a frame the point can
         perform filtering against."
}

list samplers {
    key "name";
    description
        "A list of all the samplers attached to this point."
    leaf name {
        type string;
        mandatory true;
        description
            "A unique name given to this sampler."
    }
    leaf status {
        type status-type;
        mandatory true;
        description
            "The current status of this sampler."
    }
    leaf client-heartbeat {
        type uint32;
        mandatory true;
        description
            "The number of seconds since the Client has refreshed this
             request. The Client MUST only be able to set this value
             to 0, the Replicator MUST keep track of it, and SHOULD
             delete this entry when it reaches 3600."
container destination {
    description "The destination of where to send the UDP stream to.";
    leaf type {
        type destination-type;
        mandatory true;
        description "The type of encoding for the destination.";
    }
}

container udp-parameters {
    when "../type='udp'";
    description "Parameters for destination-type=udp. Source port is always the port number assigned by IANA."
    leaf destination-ip {
        type inet:ip-address-no-zone;
        mandatory true;
        description "The destination IP to send the stream to.";
    }
    leaf destination-port {
        type inet:port-number;
        mandatory true;
        description "The destination UDP port number to send the stream to.";
    }
}

container request {
    description "The request as sent in by a Client.";
    container filters {
        description "Requested filters to apply to the stream.";
        uses filters;
    }
    leaf ratio {
        type uint32 {
            range "1..max";
        }
        mandatory true;
        description "The requested sampling ratio (1:N, with N being this value).";
    }
    leaf min-ratio {
type uint32 {
    range "1..max";
} description
"The minimum value of N the client will accept."
}
leaf max-ratio {
    type uint32 {
        range "1..max";
    } description
    "The maximum value of N the client will accept."
}
}
list proposals {
    key "id";
    config false;
    description
    "The proposals as offered by the Replicator.";
    leaf id {
        type uint32 {
            range "1..max";
        } description
        "An id-number representing this proposal for selection."
    }
    leaf ratio {
        type uint32 {
            range "1..max";
        } mandatory true;
        description
        "The offered ratio available."
    }
    leaf performance-penalty {
        type boolean;
        description
        "Selecting this offer will result in a performance penalty on the device (usually due to ASIC recirculation)"
    }
    leaf performance-penalty-amount {
        type uint16 {
            range "0..10000";
        } description
        "The performance penalty amount, in hundredths of a percent. This value is not required even if performance-penalty is true. If present, it MUST be
treated as an estimate.
}
}
}
}
}

treated as an estimate.
}
}
}
}
}
}

Authors’ Addresses

Andrew Gray
Charter Communications
8560 Upland Drive, Suite B