

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 21, 2018

B. Claise
R. Barnes
J. Clarke
Cisco
January 17, 2018

Semantic Versioning and Structure for IETF Specifications
draft-claise-semver-02

Abstract

In the Internet engineering ecosystem, there is increasingly a need for specifications that evolve over time, and which are encoded directly in structured formats (e.g., YANG models). Internet-Drafts are a poor fit for working groups that want to produce structured specifications, and publishing versions of an evolving specification as RFC makes it difficult to track the specification over time. This document outlines recommendations for how working groups can provide consistent, meaningful versions for specifications over time and work directly on structured documents while still fitting within established IETF processes.

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 <http://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 July 21, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Managing Semantic Versions	3
2.1. Versioning for Work in Progress	4
3. IETF Consensus for Structured Specifications	5
4. Example history	6
5. Creating Internet-Drafts from a Repository	8
6.1. URIs	9
Authors' Addresses	9

1. Introduction

The pace at which the software that drives the Internet is developed and deployed today is much faster than it was early in the Internet's history. In this environment, maintaining interoperability can be more challenging.

Two of the major mechanisms that have been developed for driving interoperability in a more dynamic ecosystem are structured specifications and semantic versioning. Structured specifications allow much of the work of implementing a specification to be automated, so that developers can focus on the parts of a specification that really need human involvement. Semantic versioning helps operators know what versions can be deployed without breaking running systems, so that they can safely deploy updated versions of a specification more quickly.

The traditional practices of the IETF interact poorly with these mechanisms. Each document presented to the IETF for last call and IESG approval must be formatted as an Internet-Draft, i.e., as unstructured text. All RFCs across the IETF share a single, common numbering space, so that RFC numbers have no useful semantic with respect to versioning. Nonetheless, there is still a need to be able to capture the consensus of the IETF at critical points in the life-cycle of a specification.

This document describes recommendations for how a working group (WG) that wishes to produce structured specifications with semantic version numbers can interact best with IETF processes.

2. Managing Semantic Versions

While some of this process might apply to completely new work (such as a YANG module), the process in this document applies to WG adopted work items or to modification of an existing IETF-approved work item (typically a bis document).

We start from the premise that a working group controls a version-controlled repository (e.g., Git, SVN, etc.) for a structured specification (not formatted as an Working Group Internet-Draft), and can "tag" commits in the repository as having certain version numbers. We assume that there is one repository per specification, so that version tags don't need to state the specification to which they refer.

The recommended structure for semantic versions follows the widely-used three-part convention, with an additional field for use in working group processes:

MAJOR.MINOR.PATCH

The fields in such a structured version have the following semantics (cf. semver.org):

- o MAJOR is incremented when the new version of the specification is incompatible with previous versions.
- o MINOR is incremented when new functionality is added in a manner that is backward-compatible with previous versions.
- o PATCH is incremented when bug fixes are made in a backward-compatible manner.

In IETF terms, this versioning scheme provides functionality analogous to several parts of the traditional IETF process.

- o MAJOR is analogous to an "obsoletes" relation between RFCs
- o MINOR is analogous to an "updates" relation between RFCs
- o PATCH is analogous to use of the RFC errata process

The more major a change to the specification, the more consensus is required. When the WG wants to make a MAJOR change to a structured specification, the specification MUST be converted into Internet-Draft format and run through the typical IETF consensus process. Every commit that is tagged with a MAJOR version change MUST also

have a tag of the form "RFC-XXXX" indicating the number of the RFC describing the change.

MINOR changes follow the same rules, except that the Area Director for the WG MAY approve the issuance of a minor version with only WG consensus, not full IETF consensus. Any change to the structured specification that significantly changes the security considerations for the protocol or requires additional IANA actions MUST be converted into Internet-Draft format and submitted for IETF consensus. With the approval of the AD for the WG, changes without such impacts MAY be approved by consensus of the working group.

PATCH-level changes MAY be made by the editors, with the consent of the WG chairs.

When a working group starts up work on a new version of the specification, regardless of whether it's a minor update or a complete rewrite, they SHOULD create a dedicated branch of the repository for the new version, where changes related to the new version will be committed. The semantic version, i.e. MAJOR and MINOR is assigned when this branch is merged back to the main specification. Once there is consensus to update the main specification to that version, the branch should be merged, and the merge commit tagged with the new version number.

When working on modification to an existing work item (typically a bis document), the process is to fork a git repo, branch, make a proposal, then push the branch back over to the Working Group when/if the Working Group adopts it.

2.1. Versioning for Work in Progress

It can be useful to mark certain versions of a work in progress as checkpoints, e.g., for reference at a hackathon. These checkpoints should follow their own version sequence, much like Internet drafts:

LABEL-VERSION

- o LABEL is a string that identifies the feature branch
- o VERSION is incremented whenever a new revision is tagged

These tags are analogous to Internet-Draft names. Much like an Internet-Draft name, the choice of LABEL values is up to the editors and WG chairs. For cases expected to result in a given version, they may choose to use that as a label value. For example, if the WG has agreed to embark on a major revision to the protocol, then they might

use the label "v2.0.0-beta", so that the working revisions would be "v2.0.0-beta-0", "v2.0.0-beta-1", etc.

It's important to note that not every commit needs a version. Much like working groups using Github to manage Internet-Drafts today only periodically submit them to the IETF, a WG can do work in a repository and only tag versions when they are useful to the WG. Beta versions should be tagged at key points in the development process:

- o Before an IETF meeting or WG interim meeting
- o Before a hackathon or interop event
- o Before a working group or IETF last call

Work on a new version SHOULD be conducted on a dedicated branch. Once there is consensus to update the main specification to that version, the branch should be merged, and the merge commit tagged with the new version number.

3. IETF Consensus for Structured Specifications

While working groups may use any format for specifications under development, the Internet Standards process requires that a document proposed as an RFC must be submitted in the RFC format, i.e., as unstructured text. Proposed RFCs are also required to contain explanatory text not typically contained in structured specifications, most notably Security Considerations and IANA Considerations. Thus, a working group that is focused mainly on a structured specification will have to convert the structured specification to the RFC format and add the additional explanatory text.

In order to keep the repository as the primary home for the specification, the working group should keep any required explanatory text in the repository alongside the structured specification, and use automated tools to generate an RFC-formatted document from the artifacts in the repository. As the outputs are reviewed in the IETF last call and IESG processes, the editors should reflect their responses in the repository, generating updated versions of the RFC-formatted document as necessary.

Whenever an Internet-Draft is generated from the repository, the corresponding commit in the repository should be tagged with the full name and version of the Internet-Draft. Additionally, a reference to the repository (e.g., a URL) should exist in the text of the

resulting Internet-Draft. These steps enable the evolution of the draft to easily be tied back to the evolution of the repository.

4. Example history

The below sequence of commits and tags shows the progress of a structured specification through several stages of its life-cycle. (Time flows up from the bottom, as is common in version control logs; most recent is on top.)

An initial version of a protocol is proposed for a Birds of a Feather (BoF) and a WG is formed. The WG develops version 1.0.0 of the specification. Along the way, they tag betas when they need an easy way to refer to a version, e.g., before working group last call (WGLC).

Once the WG has reached consensus, an Internet-Draft is created from the repository (draft-ietf-wg-proto-00) and submitted for the IETF consensus process, resulting in an RFC (RFC XXX1) that describes the first version of the protocol. In this example, there is never a need to publish an individual (author-named) internet-draft, because the WG worked directly on the structured specification and obtained consensus on it.

Comments from the IETF last call (LC) and the IESG are incorporated in the repository, and new versions of the Internet-Draft are generated for IESG review and submission to the RFC editor.

```
...
|
* e3091df (tag:v1.0.0, tag:draft-ietf-wg-proto-02, tag:RFCXXX1)
|   Responses to IESG comments
|
* 7494725 (tag:draft-ietf-wg-proto-01) Responses to IETF LC comments
|
* 8e2be54 (tag:proto-2, tag:draft-ietf-wg-proto-00)
|   Responses to WGLC comments
|
* 9703a60 (tag:proto-1) Responses to comments at IETF meeting
|
* 2b83977 Responses to J. Smith comments
|
* 8b75ele (tag:proto-0) Responses to BoF comments
|
* 1991498 Initial submission
```

The WG adds two features to the specification. The first feature is major enough that the chairs decide it needs IETF consensus,

resulting in a second Internet-Draft going through the IETF consensus process (draft-ietf-wg-proto-feature-00) to become an RFC (RFC XXX2). The second feature is minor enough that it can be approved by WG consensus.

```

...
|
| * a5f3214 (tag:v1.2.0) Merge branch 'v1.2'
| \
| * 8fb9cb6 Responses to WGLC comments on feature Y
| |
| * 39322e9 (tag:featureY-1) Responses to WG comments; ready for WGLC
| |
| * 39322e9 Add feature Y
| /
| * d1d201d (tag:v1.1.1) Fix validation errors
|
| * 6571483 (tag:v1.1.0, tag:draft-ietf-wg-proto-feature-04,
|   tag:RFCXXX2) Merge branch 'v1.1'
| \
| * abc3f5e (tag:draft-ietf-wg-proto-feature-03) Resolution of DISCUSSES
|   from Security AD
|
| * 3ab54f3 (tag:draft-ietf-wg-proto-feature-02) Resolution of DISCUSSES
|   from Internet and Transport ADs
|
| * cabblf6 (tag:draft-ietf-wg-proto-feature-01) Responses to WGLC
|   and IETF LC comments on feature X
|
| * fbfaa6b (tag:featureX-0, tag:draft-ietf-wg-proto-feature-00)
|   Responses to comments on feature X
|
| * 0630638 Add feature X
| /
| * e3091df (tag:v1.0.0, tag:draft-ietf-wg-proto-02, tag:RFCXXX1)
|   Responses to IESG comments
|
...

```

The WG develops a major revision of the protocol, resulting in a third Internet-Draft (draft-ietf-wg-protobis-00) going through the IETF consensus process, resulting in RFC XXX3.

```
...
|
| * a9d7d29 (tag:v2.0.0, tag:draft-ietf-wg-protobis-01 tag:RFCXXX3)
|   Merge branch 'v2'
| \
|  * df5d437 (tag:draft-ietf-wg-protobis-00) Responses to
|    WGLC and IETF LC comments
|
|  * 986ebb6 (tag:v2.0.0-beta-1) Checkpoint for hackathon
|
|  * d86986e (tag:v2.0.0-beta-0) Some more v2 features
|
|  * ca02154 Restructure for v2
| /
| * a5f3214 (tag:v1.2.0) Merge branch 'v1.2'
|
| ...
```

This example history is greatly simplified. In a real WG, there will be far more commits without versions, as the WG incorporates proposals, edits, explanatory text, etc. But this example highlights the key moments in the life-cycle of a specification.

5. Creating Internet-Drafts from a Repository

As noted above, WGs should have one repository per specification. Over the lifetime of the specification, it will be necessary for automated tools to build Internet-Drafts from this repository. A standardized repository layout can help automate this process.

The root level of the repository should have a file named "index.xml" or "index.md", depending on whether XML [1] or Markdown [2] is being used. This file should itself be a valid source for an Internet-Draft, including information about the title to be used, authors' / editors' names, security considerations, etc. It will act as a template into which the structured specification will be included when an Internet-Draft is created.

The template file should include files that comprise the structured specification as figures at appropriate points in the draft. The Markdown syntax provided by "mmark" provides a syntax for including code fragments [3] from external files, including the ability to selectively include parts of a file.

6. References

6.1. URIs

- [1] <https://tools.ietf.org/html/rfc7991>
- [2] <https://github.com/miekg/mmark/wiki/Syntax>
- [3] <https://github.com/miekg/mmark/wiki/Syntax#including-code-fragments>

Authors' Addresses

Benoit Claise
Cisco

Email: bclaise@cisco.com

Richard Barnes
Cisco

Email: rlb@ipv.sx

Joe Clarke
Cisco

Email: jclarke@cisco.com

opsawg
Internet-Draft
Intended status: Standards Track
Expires: June 19, 2019

Z. Li
R. Gu
China Mobile
J. Dong
Huawei Technologies
December 16, 2018

Export BGP community information in IP Flow Information Export (IPFIX)
draft-ietf-opsawg-ipfix-bgp-community-12

Abstract

By introducing new Information Elements (IEs), this draft extends the existing BGP-related IEs to enable IP Flow Information Export (IPFIX) to export BGP community information, including BGP standard communities defined in RFC1997, BGP extended communities defined in RFC4360, and BGP large communities defined in RFC8092. Network traffic information can then be accumulated and analyzed at the BGP community granularity, which represents the traffic of different kinds of customers, services, or geographical regions according to the network operator's BGP community planning. Network traffic information at the BGP community granularity is useful for network traffic analysis and engineering.

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 June 19, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	5
3. BGP Community-based Traffic Collection	5
4. IEs for BGP Standard Community	6
5. IEs for BGP Extended Community	7
6. IEs for BGP Large Community	7
7. Operational Considerations	8
8. Security Considerations	9
9. IANA Considerations	9
10. Acknowledgements	11
11. References	12
11.1. Normative References	12
11.2. Informative References	12
Appendix A. Encoding Example	14
A.1. Template Record	14
A.2. Data Set	15
Authors' Addresses	16

1. Introduction

IP Flow Information Export (IPFIX) [RFC7011] provides network administrators with traffic flow information using the Information Elements (IEs) defined in [IANA-IPFIX] registries. Based on the traffic flow information, network administrators know the amount and direction of the traffic in their network, and can then optimize their network when needed. For example, the collected information could be used for traffic monitoring, and could optionally be used for traffic optimization according to operator's policy.

[IANA-IPFIX] has already defined the following IEs for traffic flow information exporting in different granularities: sourceIPv4Address, sourceIPv4Prefix, destinationIPv4Address, destinationIPv4Prefix, bgpSourceAsNumber, bgpDestinationAsNumber, bgpNextHopIPv4Address, etc. In some circumstances, however, especially when traffic engineering and optimization are executed in Tier 1 or Tier 2 operators' backbone networks, traffic flow information based on these

IEs may not be completely suitable or sufficient. For example, flow information based on IP address or IP prefix may provide much too fine granularity for a large network. On the contrary, flow information based on AS number may be too coarse.

BGP community is a BGP path attribute that includes standard communities [RFC1997], extended communities [RFC4360], and large communities [RFC8092]. The BGP community attribute has a variety of use cases, one of which is to use BGP community with planned specific values to represent groups of customers, services, and geographical or topological regions, as used by operators in their networks. Detailed examples can be found in [RFC4384], [RFC8195] and Section 3 of this document. To understand the traffic generated by different kinds of customers, from different geographical or topological regions, by different kinds of customers in different regions, we need the corresponding community information related to the traffic flow information exported by IPFIX. Network traffic statistics at the BGP community granularity are useful not only for the traffic analyzing, but also can then be used by other applications, such as traffic optimization applications located in an IPFIX Collector, SDN controller or PCE. [Community-TE] also states that analyzing network traffic information at the BGP community granularity is preferred for inbound traffic engineering. However, [IANA-IPFIX] lacks IEs defined for the BGP community attribute.

Flow information based on BGP community may be collected by an IPFIX Mediator defined in [RFC6183]. IPFIX Mediator is responsible for the correlation between flow information and BGP community. However, no IEs are defined in [RFC6183] for exporting BGP community information in IPFIX. Furthermore, to correlate the BGP community with the flow information, the IPFIX Mediator needs to learn BGP routes and perform lookups in the BGP routing table to get the matching entry for a specific flow. Neither BGP route learning nor routing table lookup are trivial for an IPFIX Mediator. The IPFIX Mediator is mainly introduced to reduce the performance requirement for the Exporter [RFC5982]. In fact, to obtain the information for the already defined BGP related IEs, such as `bgpSourceAsNumber`, `bgpDestinationAsNumber`, and `bgpNextHopIPv4Address`, etc, the Exporter has to hold the up-to-date BGP routing table and perform lookups in the table. The Exporter can obtain the BGP community information in the same procedure, thus the additional load added by exporting BGP community information is minimal if the Exporter is already exporting the existing BGP-related IEs. It is RECOMMENDED that the BGP community information be exported by the Exporter directly using IPFIX.

Through running BGP [RFC4271] or BMP [RFC7854] and performing lookups in the BGP routing table to correlate the matching entry for a

specific flow, IPFIX Collectors and other applications, such as SDN controller or PCE, can determine the network traffic at the BGP community granularity. However, neither running BGP or BMP protocol nor routing table lookup are trivial for the IPFIX Collectors and other applications. Moreover, correlation between IPFIX flow information and the BGP RIB on the Exporter (such as a router) is more accurate, compared to the correlation on a Collector, since the BGP routing table may be updated when the IPFIX Collectors and other applications receive the IPFIX flow information. And as stated above, the Exporter can obtain the BGP community information during the same procedure when it obtains other BGP related information. So exporting the BGP community information directly by the Exporter to the Collector is both efficient and accurate. If the IPFIX Collectors and other applications only want to determine the network traffic at the BGP community granularity, they do not need to run the full BGP or BMP protocols when the BGP community information can be obtained by IPFIX. However, the BMP protocol has its own application scenario, and the mechanism introduced in this document is not meant to replace it.

By introducing new IEs, this draft extends the existing BGP-related IEs to enable IPFIX [RFC7011] to export BGP community information, including the BGP standard communities [RFC1997], BGP extended communities [RFC4360], and BGP large communities [RFC8092]. Flow information, including packetDeltaCount, octetDeltaCount [RFC7012], etc., can then be accumulated and analyzed by the Collector or other applications, such as an SDN controller or PCE [RFC4655], at the BGP community granularity, which is useful for measuring the traffic generated by different kinds of customers, from different geographical or topological regions according to the operator's BGP community plan, and can then be used by the traffic engineering or traffic optimization applications, especially in the backbone network.

The IEs introduced in this document are applicable for both IPv4 and IPv6 traffic. Both the Exporter and the IPFIX Mediator can use these IEs to export BGP community information in IPFIX. When needed, the IPFIX Mediator or Collector can use these IEs to report BGP community related traffic flow information it gets either from Exporters or through local correlation to other IPFIX devices.

As stated above, the method introduced in this document is not the definitive and the only one to obtain BGP community information related to a specific traffic flow, but a possible, efficient and accurate one.

No new BGP community attributes are defined in this document.

Note that this document does not update the IPFIX specification [RFC7011] and the Information Model [RFC7012]. Rather, IANA's IPFIX registry [IANA-IPFIX] contains the current complete Information Element reference, per Section 1 of [RFC7012].

Please refer to [IANA-IPFIX] for the complete list of BGP-related IEs.

Please refer to Appendix A of this document for the encoding example and Section 3 for a detailed use case.

2. Terminology

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

IPFIX-specific terminology used in this document is defined in Section 2 of [RFC7011] and Section 2 of [RFC6183].

BGP standard community: The BGP Communities attribute defined in [RFC1997]. In order to distinguish it from BGP extended communities [RFC4360], and large communities [RFC8092], BGP Communities attribute is called BGP standard community in this document.

3. BGP Community-based Traffic Collection

[RFC4384] introduces the mechanism of using BGP standard community and extended community to collect the geographical and topological related information in the BGP routing system. [RFC8195] gives some examples of the application of BGP large communities to represent the geographical regions. Since the network traffic at the BGP community granularity represents the traffic generated by different kinds of customers, from different geographical regions according to the network operator's BGP community plan, it is useful for network operators to analyze and optimize the network traffic among different customers and regions. This section gives a use case in which the network operator uses the BGP community-based traffic information to adjust the network paths for different traffic flows.

Consider the following scenario, AS C provides a transit connection between ASes A and B. By tagging with different BGP communities, the routes of AS A and B are categorized into several groups respectively in the operator's plan. For example, communities A:X and A:Y are used for the routes originated from different geographical regions in AS A, and communities B:M and B:N are used for the routes

representing the different kinds of customers in AS B, such as B:M is for the mobile customers and B:N is for the fixed line customers. By default, all traffic originating from AS A and destined to AS B (we call it traffic A-B) goes through path C1-C2-C3 (call it Path-1) in AS C. When the link between C1 and C2 is congested, we cannot simply steer all the traffic A-B from Path-1 to Path C1-C4-C3 (call it Path-2), because it will cause congestion in Path-2.

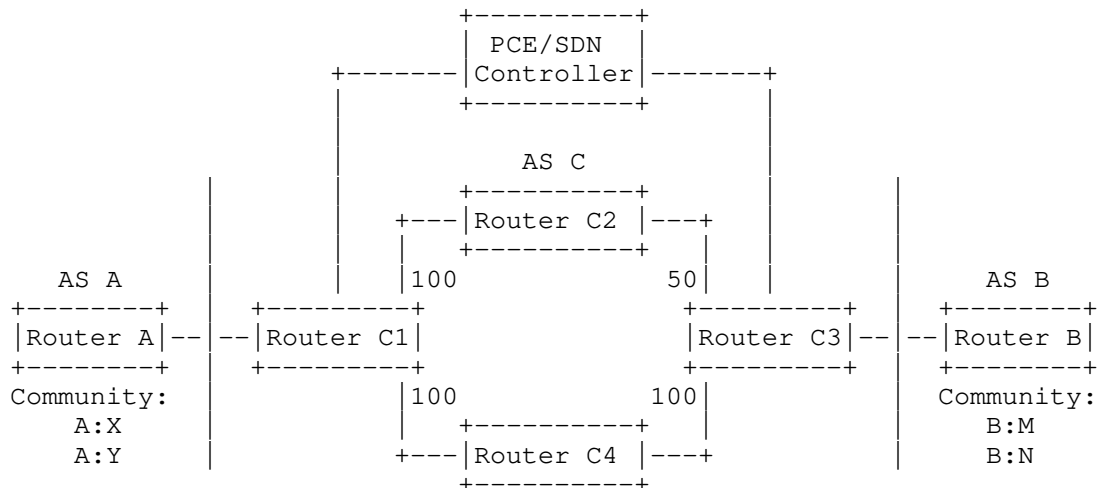


Figure 1: BGP Community based Traffic Collection

If the PCE/SDN controller in AS C can obtain the network traffic information at the BGP community granularity, it can steer some traffic related to some BGP communities (when we consider only the source or destination of the traffic), or some BGP community pairs (when we consider both the source and the destination of the traffic) from Path-1 to Path-2 according to the utilization of different paths. For instance, steer the traffic generated by community A:X from Path-1 to Path-2 by deploying a route policy at Router C1, or steer the traffic from community A:Y to community B:M from Path-1 to Path-2. Using the IEs defined in this document, IPFIX can export the BGP community information related to a specific traffic flow together with other flow information. The traffic information can then be accumulated at the BGP community granularity and used by the PCE/SDN controller to steer the appropriate traffic from Path-1 to Path-2.

4. IEs for BGP Standard Community

[RFC1997] defines the BGP Communities attribute, called BGP Standard Community in this document, which describes a group of routes sharing

some common properties. BGP Standard Community is treated as 32 bit value as stated in [RFC1997].

In order to export BGP standard community information along with other flow information defined by IPFIX, three new IEs are introduced. One is `bgpCommunity`, which is used to identify that the value in this IE is a BGP standard community. The other two are `bgpSourceCommunityList` and `bgpDestinationCommunityList`, which are both `basicList` [RFC6313] of `bgpCommunity`, and are used to export BGP standard community information corresponding to a specific flow's source and destination IP address respectively.

The detailed information of the three new IEs are shown in Section 9, IANA Considerations.

5. IEs for BGP Extended Community

[RFC4360] defines the BGP Extended Communities attribute, which provides a mechanism for labeling the information carried in BGP. Each Extended Community is encoded as an 8-octet quantity with the format defined in [RFC4360].

In order to export BGP Extended Community information together with other flow information by IPFIX, three new IEs are introduced. The first one is `bgpExtendedCommunity`, which is used to identify that the value in this IE is a BGP Extended Community. The other two are `bgpSourceExtendedCommunityList` and `bgpDestinationExtendedCommunityList`, which are both `basicList` [RFC6313] of `bgpExtendedCommunity`, and are used to export the BGP Extended Community information corresponding to a specific flow's source and destination IP address respectively.

The detailed information of the three new IEs are shown in Section 9, IANA Considerations.

6. IEs for BGP Large Community

[RFC8092] defines the BGP Large Communities attribute, which is suitable for use with all Autonomous System Numbers (ASNs) including four-octet ASNs. Each BGP Large Community is encoded as a 12-octet quantity with the format defined in [RFC8092].

In order to export BGP Large Community information together with other flow information by IPFIX, three new IEs are introduced. The first one is `bgpLargeCommunity`, which is used to identify that the value in this IE is a BGP Large Community. The other two are `bgpSourceLargeCommunityList` and `bgpDestinationLargeCommunityList`, which are both `basicList` [RFC6313] of `bgpLargeCommunity`, and are used

to export the BGP Large Community information corresponding to a specific flow's source and destination IP address respectively.

The detailed information of the three new IEs are shown in Section 9, IANA Considerations.

7. Operational Considerations

The maximum length of an IPFIX message is 65535 bytes as per [RFC7011], and the maximum length of a normal BGP message is 4096 bytes as per [RFC4271]. Since BGP communities, including standard, extended, and large communities, are BGP path attributes carried in BGP Update messages, the total length of these attributes can not exceed the length of a BGP message, i.e. 4096 bytes. So one IPFIX message with a maximum length of 65535 bytes has enough space to fit all the communities related to a specific flow, relating to both the source and destination IP addresses.

[I-D.ietf-idr-bgp-extended-messages] extends the maximum size of a BGP Update message to 65535 bytes. In that case, the BGP community information related to a specific flow could theoretically exceed the length of one IPFIX message. However, according to information regarding actual networks in the field, the number of BGP communities in one BGP route is usually no more than ten. Nevertheless, BGP speakers that support the extended message SHOULD only convey as many communities as possible without exceeding the 65536-byte limit of an IPFIX message. The Collector which receives an IPFIX message with maximum length and BGP communities contained in its data set SHOULD generate a warning or log message to indicate that the BGP communities may be truncated due to limited message space. In this case, it is recommended to configure the export policy of BGP communities to limit the BGP communities by including or excluding specific communities.

If needed, the IPFIX message length could be extended from 16 bits to 32 bits to solve this problem completely. The details of increasing the IPFIX message length is out of scope of this document.

To align with the size of the BGP extended community and large community attributes, the size of IE `bgpExtendedCommunity` and `bgpLargeCommunity` is 8 octets and 12 octets respectively. In the event that the `bgpExtendedCommunity` or `bgpLargeCommunity` IE is not of its expected size, the IPFIX Collector SHOULD ignore it. This is intended to protect implementations using BGP logic from calling their parsing routines with invalid lengths.

For the proper processing of the Exporter when it receives the template requesting to report the BGP community information (refer to

Appendix A for an example), the Exporter SHOULD obtain the corresponding BGP community information through BGP lookup using the corresponding source or destination IP address of the specific traffic flow. When exporting the IPFIX information to the Collector, the Exporter SHOULD include the corresponding BGP communities in the IPFIX message.

8. Security Considerations

This document defines new IEs for IPFIX. The same security considerations as for the IPFIX Protocol Specification [RFC7011] and Information Model [RFC7012] apply.

Systems processing BGP community information collected by IPFIX collectors need to be aware of the use of communities as an attack vector [Weaponizing-BGP], and only include BGP community information in their decisions where they are confident of its validity. Thus we can not assume that all BGP community information collected by IPFIX collectors is credible and accurate. It is RECOMMENDED to use only the IPFIX collected BGP community information that the processing system can trust, for example the BGP communities generated by the consecutive neighboring ASs within the same trust domain as the processing system (for instance, the consecutive neighboring ASs and the processing system are operated by one carrier).

[RFC7011] says that the storage of the information collected by IPFIX must be protected and confined its visibility to authorized users via technical as well as policy means to ensure the privacy of the information collected. [RFC7011] also provides mechanisms to ensure the confidentiality and integrity of IPFIX data transferred from an Exporting Process to a Collecting Process. The mechanism to authenticate IPFIX Collecting and Exporting Processes is provided in [RFC7011], too. If sensitive information is contained in the community information, the above recommendations and mechanisms are recommended to be used. No additional privacy risks are introduced by this standard.

9. IANA Considerations

This draft specifies the following IPFIX IEs to export BGP community information along with other flow information.

The Element IDs for these IEs are requested to be assigned by IANA. The following table is for IANA's use to place in each field in the registry.

ElementID	Name	Data Type	Data Type Semantics
-----------	------	-----------	---------------------

TBA1	bgpCommunity	unsigned32	identifier
TBA2	bgpSourceCommunityList	basicList	list
TBA3	bgpDestinationCommunityList	basicList	list
TBA4	bgpExtendedCommunity	octetArray	default
TBA5	bgpSourceExtendedCommunityList	basicList	list
TBA6	bgpDestinationExtendedCommunityList	basicList	list
TBA7	bgpLargeCommunity	octetArray	default
TBA8	bgpSourceLargeCommunityList	basicList	list
TBA9	bgpDestinationLargeCommunityList	basicList	list

ElementID	Description	Units
TBA1	BGP community as defined in [RFC1997]	
TBA2	basicList of zero or more bgpCommunity IEs, containing the BGP communities corresponding with source IP address of a specific flow	
TBA3	basicList of zero or more bgpCommunity IEs, containing the BGP communities corresponding with destination IP address of a specific flow	
TBA4	BGP Extended Community as defined in [RFC4360] The size of this IE MUST be 8 octets	
TBA5	basicList of zero or more bgpExtendedCommunity IEs, containing the BGP Extended Communities corresponding with source IP address of a specific flow	
TBA6	basicList of zero or more bgpExtendedCommunity IEs, containing the BGP Extended Communities corresponding with destination IP address of a specific flow	

TBA7	BGP Large Community as defined in [RFC8092] The size of this IE MUST be 12 octets.	
TBA8	basicList of zero or more bgpLargeCommunity IEs, containing the BGP Large Communities corresponding with source IP address of a specific flow	
TBA9	basicList of zero or more bgpLargeCommunity IEs, containing the BGP Large Communities corresponding with destination IP address of a specific flow	

ElementID	Range	References	Requester	Revision	date
TBA1		RFC1997	this draft	0	
TBA2		RFC6313,RFC1997	this draft	0	
TBA3		RFC6313,RFC1997	this draft	0	
TBA4		RFC4360	this draft	0	
TBA5		RFC6313,RFC4360	this draft	0	
TBA6		RFC6313,RFC4360	this draft	0	
TBA7		RFC8092	this draft	0	
TBA8		RFC6313,RFC8092	this draft	0	
TBA9		RFC6313,RFC8092	this draft	0	

Figure 2: IANA Considerations

10. Acknowledgements

The authors would like to thank Benoit Claise and Paul Aitken for their comments and suggestions to promote this document. We also thank Tianran Zhou, Warren Kumari, Jeffrey Haas, Ignas Bagdonas, Stewart Bryant, Paolo Lucente, Job Snijders, Jared Mauch, Rudiger Volk, and Andrew Malis for their discussion, comments, and suggestions to improve this document..

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6313] Claise, B., Dhandapani, G., Aitken, P., and S. Yates, "Export of Structured Data in IP Flow Information Export (IPFIX)", RFC 6313, DOI 10.17487/RFC6313, July 2011, <<https://www.rfc-editor.org/info/rfc6313>>.
- [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, <<https://www.rfc-editor.org/info/rfc7011>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

11.2. Informative References

- [Community-TE] Shao, W., Devienne, F., Iannone, L., and JL. Rougier, "On the use of BGP communities for fine-grained inbound traffic engineering", Computer Science 27392(1):476-487, November 2015.
- [I-D.ietf-idr-bgp-extended-messages] Bush, R., Patel, K., and D. Ward, "Extended Message support for BGP", draft-ietf-idr-bgp-extended-messages-27 (work in progress), December 2018.
- [IANA-IPFIX] "IP Flow Information Export (IPFIX) Entities", <<http://www.iana.org/assignments/ipfix/>>.
- [RFC1997] Chandra, R., Traina, P., and T. Li, "BGP Communities Attribute", RFC 1997, DOI 10.17487/RFC1997, August 1996, <<https://www.rfc-editor.org/info/rfc1997>>.

- [RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <<https://www.rfc-editor.org/info/rfc4271>>.
- [RFC4360] Sangli, S., Tappan, D., and Y. Rekhter, "BGP Extended Communities Attribute", RFC 4360, DOI 10.17487/RFC4360, February 2006, <<https://www.rfc-editor.org/info/rfc4360>>.
- [RFC4384] Meyer, D., "BGP Communities for Data Collection", BCP 114, RFC 4384, DOI 10.17487/RFC4384, February 2006, <<https://www.rfc-editor.org/info/rfc4384>>.
- [RFC4655] Farrel, A., Vasseur, J., and J. Ash, "A Path Computation Element (PCE)-Based Architecture", RFC 4655, DOI 10.17487/RFC4655, August 2006, <<https://www.rfc-editor.org/info/rfc4655>>.
- [RFC5982] Kobayashi, A., Ed. and B. Claise, Ed., "IP Flow Information Export (IPFIX) Mediation: Problem Statement", RFC 5982, DOI 10.17487/RFC5982, August 2010, <<https://www.rfc-editor.org/info/rfc5982>>.
- [RFC6183] Kobayashi, A., Claise, B., Muenz, G., and K. Ishibashi, "IP Flow Information Export (IPFIX) Mediation: Framework", RFC 6183, DOI 10.17487/RFC6183, April 2011, <<https://www.rfc-editor.org/info/rfc6183>>.
- [RFC7012] Claise, B., Ed. and B. Trammell, Ed., "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, DOI 10.17487/RFC7012, September 2013, <<https://www.rfc-editor.org/info/rfc7012>>.
- [RFC7854] Scudder, J., Ed., Fernando, R., and S. Stuart, "BGP Monitoring Protocol (BMP)", RFC 7854, DOI 10.17487/RFC7854, June 2016, <<https://www.rfc-editor.org/info/rfc7854>>.
- [RFC8092] Heitz, J., Ed., Snijders, J., Ed., Patel, K., Bagdonas, I., and N. Hilliard, "BGP Large Communities Attribute", RFC 8092, DOI 10.17487/RFC8092, February 2017, <<https://www.rfc-editor.org/info/rfc8092>>.
- [RFC8195] Snijders, J., Heasley, J., and M. Schmidt, "Use of BGP Large Communities", RFC 8195, DOI 10.17487/RFC8195, June 2017, <<https://www.rfc-editor.org/info/rfc8195>>.

[Weaponizing-BGP]

Streibelt, F., Lichtblau, F., Beverly, R., and et al.,
 "Weaponizing BGP Using Communities", November 2018,
 <<https://datatracker.ietf.org/meeting/103/materials/slides-103-grow-bgp-communities-spread-their-wings-01>>.

Appendix A. Encoding Example

In this section, we provide an example to show the encoding format for the new introduced IEs.

Flow information, including BGP communities, is shown in the following table. In this example, all the fields are reported by IPFIX.

Source IP	Destination IP	BGP community corresponding with Source IP	BGP community corresponding with Destination IP
1.1.1.1	2.2.2.2	1:1001,1:1002,8:1001	2:1002,8:1001
3.3.3.3	4.4.4.4	3:1001,3:1002,8:1001	4:1001,8:1001

Figure 3: Flow information including BGP communities

A.1. Template Record

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	2	3	4	5	6	7	8	9	0	1								
SET ID = 2										Length = 24																													
Template ID = 256										Field Count = 4																													
0	SourceIPv4Address = 8								Field length = 4																														
0	DestinationIPv4Address = 12								Field length = 4																														
0	bgpSourceCommunityList= TBA2								Field length = 0xFFFF																														
0	bgpDestinationCommunityList = TBA3								Field length = 0xFFFF																														

Figure 4: Template Record Encoding Format

In this example, the Template ID is 256, which will be used in the Data Record. The field length for `bgpSourceCommunityList` and `bgpDestinationCommunityList` is `0xFFFF`, which means the length of this IE is variable, and the actual length of this IE is indicated by the list length field in the basic list format as per [RFC6313].

A.2. Data Set

The data set is represented as follows:

```

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 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|          SET ID = 256          |          Length = 92          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          SourceIPv4Address = 1.1.1.1          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          DestinationIPv4Address = 2.2.2.2          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          255          |          List length = 17          |semantic=allof|
+-----+-----+-----+-----+-----+-----+-----+-----+
|          bgpCommunity = TBA1          |          Field Len = 4          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          BGP Source Community Value 1 = 1:1001          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          BGP Source Community Value 2 = 1:1002          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          BGP Source Community Value 3 = 8:1001          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          255          |          List length = 13          |semantic=allof|
+-----+-----+-----+-----+-----+-----+-----+-----+
|          bgpCommunity = TBA1          |          Field Len = 4          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          BGP Destination Community Value 1 = 2:1002          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          BGP Destination Community Value 2 = 8:1001          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          SourceIPv4Address = 3.3.3.3          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          DestinationIPv4Address = 4.4.4.4          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          255          |          List length = 17          |semantic=allof|
+-----+-----+-----+-----+-----+-----+-----+-----+
|          bgpCommunity = TBA1          |          Field Len = 4          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          BGP Source Community Value 1 = 3:1001          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          BGP Source Community Value 2 = 3:1002          |

```



```

+-----+
|      BGP Source Community Value 3  = 8:1001      |
+-----+
|      255      |      List length = 13      |semantic =allof|
+-----+
|      bgpCommunity = TBA1      |      Field Len = 4      |
+-----+
|      BGP Destination Community Value 1 = 4:1001  |
+-----+
|      BGP Destination Community Value 2 = 8:1001  |
+-----+

```

Figure 5: Data Set Encoding Format

Authors' Addresses

Zhenqiang Li
 China Mobile
 32 Xuanwumen West Ave, Xicheng District
 Beijing 100053
 China

Email: li_zhenqiang@hotmail.com

Rong Gu
 China Mobile
 32 Xuanwumen West Ave, Xicheng District
 Beijing 100053
 China

Email: gurong_cmcc@outlook.com

Jie Dong
 Huawei Technologies
 Huawei Campus, No. 156 Beiqing Rd.
 Beijing 100095
 China

Email: jie.dong@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2018

E. Lear
Cisco Systems
R. Droms
Google
D. Romascanu
June 15, 2018

Manufacturer Usage Description Specification
draft-ietf-opsawg-mud-25

Abstract

This memo specifies a component-based architecture for manufacturer usage descriptions (MUD). The goal of MUD is to provide a means for end devices to signal to the network what sort of access and network functionality they require to properly function. The initial focus is on access control. Later work can delve into other aspects.

This memo specifies two YANG modules, IPv4 and IPv6 DHCP options, an LLDP TLV, a URL, an X.509 certificate extension and a means to sign and verify the descriptions.

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 December 17, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. What MUD Doesn't Do	5
1.2. A Simple Example	5
1.3. Terminology	5
1.4. Determining Intended Use	6
1.5. Finding A Policy: The MUD URL	6
1.6. Processing of the MUD URL	7
1.7. Types of Policies	8
1.8. The Manufacturer Usage Description Architecture	10
1.9. Order of operations	11
2. The MUD Model and Semantic Meaning	12
2.1. The IETF-MUD YANG Module	13
3. MUD model definitions for the root mud container	14
3.1. mud-version	14
3.2. mud-url	15
3.3. to-device-policy and from-device-policy containers	15
3.4. last-update	15
3.5. cache-validity	15
3.6. is-supported	15
3.7. systeminfo	15
3.8. mfg-name, software-rev, model-name firmware-rev	16
3.9. extensions	16
4. Augmentation to the ACL Model	16
4.1. manufacturer	16
4.2. same-manufacturer	16
4.3. documentation	17
4.4. model	17
4.5. local-networks	17
4.6. controller	17
4.7. my-controller	18
4.8. direction-initiated	18
5. Processing of the MUD file	18
6. What does a MUD URL look like?	18
7. The MUD YANG Model	19
8. The Domain Name Extension to the ACL Model	25
8.1. src-dnsname	26
8.2. dst-dnsname	26
8.3. The ietf-acldns Model	26
9. MUD File Example	28

10. The MUD URL DHCP Option	30
10.1. Client Behavior	31
10.2. Server Behavior	31
10.3. Relay Requirements	32
11. The Manufacturer Usage Description (MUD) URL X.509 Extension	32
12. The Manufacturer Usage Description LLDP extension	34
13. Creating and Processing of Signed MUD Files	35
13.1. Creating a MUD file signature	36
13.2. Verifying a MUD file signature	36
14. Extensibility	37
15. Deployment Considerations	37
16. Security Considerations	38
17. IANA Considerations	40
17.1. YANG Module Registrations	41
17.2. DHCPv4 and DHCPv6 Options	41
17.3. PKIX Extensions	41
17.4. MIME Media-type Registration for MUD files	42
17.5. LLDP IANA TLV Subtype Registry	42
17.6. The MUD Well Known Universal Resource Name (URNs)	43
17.7. Extensions Registry	43
18. Acknowledgments	43
19. References	44
19.1. Normative References	44
19.2. Informative References	47
Appendix A. Changes from Earlier Versions	49
Appendix B. Default MUD nodes	52
Appendix C. A Sample Extension: DETNET-indicator	57
Authors' Addresses	60

1. Introduction

The Internet has largely been constructed for general purpose computers, those devices that may be used for a purpose that is specified by those who own the device. [RFC1984] presumed that an end device would be most capable of protecting itself. This made sense when the typical device was a workstation or a mainframe, and it continues to make sense for general purpose computing devices today, including laptops, smart phones, and tablets.

[RFC7452] discusses design patterns for, and poses questions about, smart objects. Let us then posit a group of objects that are specifically not intended to be used for general purpose computing tasks. These devices, which this memo refers to as Things, have a specific purpose. By definition, therefore, all other uses are not intended. If a small number of communication patterns follows from those small number of uses, the combination of these two statements can be restated as a manufacturer usage description (MUD) that can be applied at various points within a network. MUD primarily addresses

threats to the device rather than the device as a threat. In some circumstances, however, MUD may offer some protection in the latter case, depending on the MUD-URL is communicated, and how devices and their communications are authenticated.

We use the notion of "manufacturer" loosely in this context to refer to the entity or organization that will state how a device is intended to be used. For example, in the context of a lightbulb, this might indeed be the lightbulb manufacturer. In the context of a smarter device that has a built in Linux stack, it might be an integrator of that device. The key points are that the device itself is assumed to serve a limited purpose, and that there exists an organization in the supply chain of that device that will take responsibility for informing the network about that purpose.

The intent of MUD is to provide the following:

- o Substantially reduce the threat surface on a device to those communications intended by the manufacturer.
- o Provide a means to scale network policies to the ever-increasing number of types of devices in the network.
- o Provide a means to address at least some vulnerabilities in a way that is faster than the time it might take to update systems. This will be particularly true for systems that are no longer supported.
- o Keep the cost of implementation of such a system to the bare minimum.
- o Provide a means of extensibility for manufacturers to express other device capabilities or requirements.

MUD consists of three architectural building blocks:

- o A URL that can be used to locate a description;
- o The description itself, including how it is interpreted, and;
- o A means for local network management systems to retrieve the description.

MUD is most effective when the network is able to identify in some way the remote endpoints that Things will talk to.

In this specification we describe each of these building blocks and how they are intended to be used together. However, they may also be

used separately, independent of this specification, by local deployments for their own purposes.

1.1. What MUD Doesn't Do

MUD is not intended to address network authorization of general purpose computers, as their manufacturers cannot envision a specific communication pattern to describe. In addition, even those devices that have a single or small number of uses might have very broad communication patterns. MUD on its own is not for them either.

Although MUD can provide network administrators with some additional protection when device vulnerabilities exist, it will never replace the need for manufacturers to patch vulnerabilities.

Finally, no matter what the manufacturer specifies in a MUD file, these are not directives, but suggestions. How they are instantiated locally will depend on many factors and will be ultimately up to the local network administrator, who must decide what is appropriate in a given circumstances.

1.2. A Simple Example

A light bulb is intended to light a room. It may be remotely controlled through the network, and it may make use of a rendezvous service of some form that an application on a smart phone. What we can say about that light bulb, then, is that all other network access is unwanted. It will not contact a news service, nor speak to the refrigerator, and it has no need of a printer or other devices. It has no social networking friends. Therefore, an access list applied to it that states that it will only connect to the single rendezvous service will not impede the light bulb in performing its function, while at the same time allowing the network to provide both it and other devices an additional layer of protection.

1.3. Terminology

MUD: manufacturer usage description.

MUD file: a file containing YANG-based JSON that describes a Thing and associated suggested specific network behavior.

MUD file server: a web server that hosts a MUD file.

MUD manager: the system that requests and receives the MUD file from the MUD server. After it has processed a MUD file, it may direct changes to relevant network elements.

MUD controller: a synonym that has been used in the past for MUD manager.

MUD URL: a URL that can be used by the MUD manager to receive the MUD file.

Thing: the device emitting a MUD URL.

Manufacturer: the entity that configures the Thing to emit the MUD URL and the one who asserts a recommendation in a MUD file. The manufacturer might not always be the entity that constructs a Thing. It could, for instance, be a systems integrator, or even a component provider.

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.4. Determining Intended Use

The notion of intended use is in itself not new. Network administrators apply access lists every day to allow for only such use. This notion of white listing was well described by Chapman and Zwicky in [FW95]. Profiling systems that make use of heuristics to identify types of systems have existed for years as well.

A Thing could just as easily tell the network what sort of access it requires without going into what sort of system it is. This would, in effect, be the converse of [RFC7488]. In seeking a general solution, however, we assume that a device will implement functionality necessary to fulfill its limited purpose. This is basic economic constraint. Unless the network would refuse access to such a device, its developers would have no reason to provide the network any information. To date, such an assertion has held true.

1.5. Finding A Policy: The MUD URL

Our work begins with the device emitting a Universal Resource Locator (URL) [RFC3986]. This URL serves both to classify the device type and to provide a means to locate a policy file.

MUD URLs MUST use the HTTPS scheme [RFC7230].

In this memo three means are defined to emit the MUD URL, as follows:

- o A DHCP option[RFC2131],[RFC3315] that the DHCP client uses to inform the DHCP server. The DHCP server may take further actions, such as act as the MUD manager or otherwise pass the MUD URL along to the MUD manager.
- o An X.509 constraint. The IEEE has developed [IEEE802.1AR] that provides a certificate-based approach to communicate device characteristics, which itself relies on [RFC5280]. The MUD URL extension is non-critical, as required by IEEE 802.1AR. Various means may be used to communicate that certificate, including Tunnel Extensible Authentication Protocol (TEAP) [RFC7170].
- o Finally, a Link Layer Discovery Protocol (LLDP) frame is defined [IEEE802.1AB].

It is possible that there may be other means for a MUD URL to be learned by a network. For instance, some devices may already be fielded or have very limited ability to communicate a MUD URL, and yet can be identified through some means, such as a serial number or a public key. In these cases, manufacturers may be able to map those identifiers to particular MUD URLs (or even the files themselves). Similarly, there may be alternative resolution mechanisms available for situations where Internet connectivity is limited or does not exist. Such mechanisms are not described in this memo, but are possible. Implementors are encouraged to allow for this sort of flexibility of how MUD URLs may be learned.

1.6. Processing of the MUD URL

MUD managers that are able to do so SHOULD retrieve MUD URLs and signature files as per [RFC7230], using the GET method [RFC7231]. They MUST validate the certificate using the rules in [RFC2818], Section 3.1.

Requests for MUD URLs SHOULD include an "Accept" header ([RFC7231], Section 5.3.2) containing "application/mud+json", an "Accept-Language" header field ([RFC7231], Section 5.3.5), and a "User-Agent" header ([RFC7231], Section 5.5.3).

MUD managers SHOULD automatically process 3xx response status codes.

If a MUD manager is not able to fetch a MUD URL, other means MAY be used to import MUD files and associated signature files. So long as the signature of the file can be validated, the file can be used. In such environments, controllers SHOULD warn administrators when cache-validity expiry is approaching so that they may check for new files.

It may not be possible for a MUD manager to retrieve a MUD file at any given time. Should a MUD manager fail to retrieve a MUD file, it SHOULD consider the existing one safe to use, at least for a time. After some period, it SHOULD log that it has been unable to retrieve the file. There may be very good reasons for such failures, including the possibility that the MUD manager is in an off-line environment, the local Internet connection has failed, or the remote Internet connection has failed. It is also possible that an attacker is attempting to interfere with the deployment of a device. It is a local decision as to how to handle such circumstances.

1.7. Types of Policies

When the MUD URL is resolved, the MUD manager retrieves a file that describes what sort of communications a device is designed to have. The manufacturer may specify either specific hosts for cloud based services or certain classes for access within an operational network. An example of a class might be "devices of a specified manufacturer type", where the manufacturer type itself is indicated simply by the authority component (e.g, the domain name) of the MUD URL. Another example might be to allow or disallow local access. Just like other policies, these may be combined. For example:

- o Allow access to devices of the same manufacturer
- o Allow access to and from controllers via Constrained Application Protocol (COAP)[RFC7252]
- o Allow access to local DNS/NTP
- o Deny all other access

A printer might have a description that states:

- o Allow access for port IPP or port LPD
- o Allow local access for port HTTP
- o Deny all other access

In this way anyone can print to the printer, but local access would be required for the management interface.

The files that are retrieved are intended to be closely aligned to existing network architectures so that they are easy to deploy. We make use of YANG [RFC7950] because it provides accurate and adequate models for use by network devices. JSON[RFC8259] is used as a

serialization format for compactness and readability, relative to XML. Other formats may be chosen with later versions of MUD.

While the policy examples given here focus on access control, this is not intended to be the sole focus. By structuring the model described in this document with clear extension points, other descriptions could be included. One that often comes to mind is quality of service.

The YANG modules specified here are extensions of [I-D.ietf-netmod-acl-model]. The extensions to this model allow for a manufacturer to express classes of systems that a manufacturer would find necessary for the proper function of the device. Two modules are specified. The first module specifies a means for domain names to be used in ACLs so that devices that have their controllers in the cloud may be appropriately authorized with domain names, where the mapping of those names to addresses may rapidly change.

The other module abstracts away IP addresses into certain classes that are instantiated into actual IP addresses through local processing. Through these classes, manufacturers can specify how the device is designed to communicate, so that network elements can be configured by local systems that have local topological knowledge. That is, the deployment populates the classes that the manufacturer specifies. The abstractions below map to zero or more hosts, as follows:

Manufacturer: A device made by a particular manufacturer, as identified by the authority component of its MUD URL

same-manufacturer: Devices that have the same authority component of their MUD URL.

controller: Devices that the local network administrator admits to the particular class.

my-controller: Devices intended to serve as controllers for the MUD-URL that the Thing emitted.

local: The class of IP addresses that are scoped within some administrative boundary. By default it is suggested that this be the local subnet.

The "manufacturer" classes can be easily specified by the manufacturer, whereas controller classes are initially envisioned to be specified by the administrator.

Because manufacturers do not know who will be using their devices, it is important for functionality referenced in usage descriptions to be relatively ubiquitous and mature. For these reasons the YANG-based configuration in a MUD file is limited to either the modules specified or referenced in this document, or those specified in documented extensions.

1.8. The Manufacturer Usage Description Architecture

With these components laid out we now have the basis for an architecture. This leads us to ASCII art.

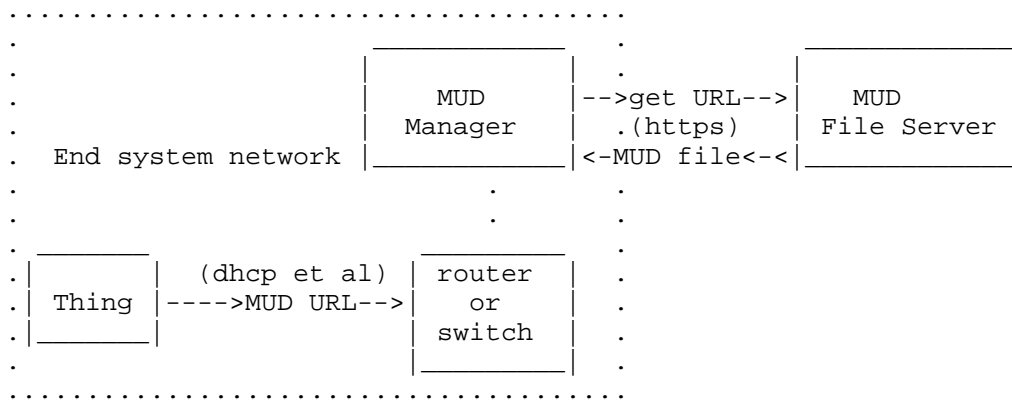


Figure 1: MUD Architecture

In the above diagram, the switch or router collects MUD URLs and forwards them to the MUD manager (a network management system) for processing. This happens in different ways, depending on how the URL is communicated. For instance, in the case of DHCP, the DHCP server might receive the URL and then process it. In the case of IEEE 802.1X [IEEE8021X], the switch would carry the URL via a certificate to the authentication server via EAP over Radius[RFC3748], which would then process it. One method to do this is TEAP, described in [RFC7170]. The certificate extension is described below.

The information returned by the MUD file server is valid for as long as the Thing is connected. There is no expiry. However, if the MUD manager has detected that the MUD file for a Thing has changed, it SHOULD update the policy expeditiously, taking into account whatever approval flow is required in a deployment. In this way, new recommendations from the manufacturer can be processed in a timely fashion.

The information returned by the MUD file server (a web server) is valid for the duration of the Thing's connection, or as specified in the description. Thus if the Thing is disconnected, any associated configuration in the switch can be removed. Similarly, from time to time the description may be refreshed, based on new capabilities or communication patterns or vulnerabilities.

The web server is typically run by or on behalf of the manufacturer. Its domain name is that of the authority found in the MUD URL. For legacy cases where Things cannot emit a URL, if the switch is able to determine the appropriate URL, it may proxy it. In the trivial case it may hardcode MUD-URL on a switch port or a map from some available identifier such as an L2 address or certificate hash to a MUD-URL.

The role of the MUD manager in this environment is to do the following:

- o receive MUD URLs,
- o fetch MUD files,
- o translate abstractions in the MUD files to specific network element configuration,
- o maintain and update any required mappings of the abstractions, and
- o update network elements with appropriate configuration.

A MUD manager may be a component of a AAA or network management system. Communication within those systems and from those systems to network elements is beyond the scope of this memo.

1.9. Order of operations

As mentioned above, MUD contains architectural building blocks, and so order of operation may vary. However, here is one clear intended example:

1. Thing emits URL.
2. That URL is forwarded to a MUD manager by the nearest switch (how this happens depends on the way in which the MUD URL is emitted).
3. The MUD manager retrieves the MUD file and signature from the MUD file server, assuming it doesn't already have copies. After validating the signature, it may test the URL against a web or domain reputation service, and it may test any hosts within the file against those reputation services, as it deems fit.

4. The MUD manager may query the administrator for permission to add the Thing and associated policy. If the Thing is known or the Thing type is known, it may skip this step.
5. The MUD manager instantiates local configuration based on the abstractions defined in this document.
6. The MUD manager configures the switch nearest the Thing. Other systems may be configured as well.
7. When the Thing disconnects, policy is removed.

2. The MUD Model and Semantic Meaning

A MUD file consists of a YANG model instance that has been serialized in JSON [RFC7951]. For purposes of MUD, the nodes that can be modified are access lists as augmented by this model. The MUD file is limited to the serialization of only the following YANG schema:

- o ietf-access-control-list [I-D.ietf-netmod-acl-model]
- o ietf-mud (this document)
- o ietf-acldns (this document)

Extensions may be used to add additional schema. This is described further on.

To provide the widest possible deployment, publishers of MUD files SHOULD make use of the abstractions in this memo and avoid the use of IP addresses. A MUD manager SHOULD NOT automatically implement any MUD file that contains IP addresses, especially those that might have local significance. The addressing of one side of an access list is implicit, based on whether it is applied as to-device-policy or from-device-policy.

With the exceptions of "name" of the ACL, "type", "name" of the ACE, and TCP and UDP source and destination port information, publishers of MUD files SHOULD limit the use of ACL model leaf nodes expressed to those found in this specification. Absent any extensions, MUD files are assumed to implement only the following ACL model features:

- o match-on-ipv4, match-on-ipv6, match-on-tcp, match-on-udp, match-on-icmp

Furthermore, only "accept" or "drop" actions SHOULD be included. A MUD manager MAY choose to interpret "reject" as "drop". A MUD manager SHOULD ignore all other actions. This is because

manufacturers do not have sufficient context within a local deployment to know whether reject is appropriate. That is a decision that should be left to a network administrator.

Given that MUD does not deal with interfaces, the support of the "ietf-interfaces" module [RFC8343] is not required. Specifically, the support of interface-related features and branches (e.g., interface-attachment and interface-stats) of the ACL YANG module is not required.

In fact, MUD managers MAY ignore any particular component of a description or MAY ignore the description in its entirety, and SHOULD carefully inspect all MUD descriptions. Publishers of MUD files MUST NOT include other nodes except as described in Section 3.9. See that section for more information.

2.1. The IETF-MUD YANG Module

This module is structured into three parts:

- o The first component, the "mud" container, holds information that is relevant to retrieval and validity of the MUD file itself, as well as policy intended to and from the Thing.
- o The second component augments the matching container of the ACL model to add several nodes that are relevant to the MUD URL, or otherwise abstracted for use within a local environment.
- o The third component augments the tcp-acl container of the ACL model to add the ability to match on the direction of initiation of a TCP connection.

A valid MUD file will contain two root objects, a "mud" container and an "acls" container. Extensions may add additional root objects as required. As a reminder, when parsing acls, elements within a "match" block are logically ANDed. In general, a single abstraction in a match statement should be used. For instance, it makes little sense to match both "my-controller" and "controller" with an argument, since they are highly unlikely to be the same value.

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is explained in [RFC8340].

```

module: ietf-mud
  +--rw mud!
    +--rw mud-version          uint8
    +--rw mud-url              inet:uri
    +--rw last-update          yang:date-and-time
    +--rw mud-signature?       inet:uri
    +--rw cache-validity?      uint8
    +--rw is-supported         boolean
    +--rw systeminfo?          string
    +--rw mfg-name?            string
    +--rw model-name?          string
    +--rw firmware-rev?        string
    +--rw software-rev?        string
    +--rw documentation?       inet:uri
    +--rw extensions*          string
    +--rw from-device-policy
      | +--rw acls
      | | +--rw access-list* [name]
      | | | +--rw name      -> /acl:acls/acl/name
      | +--rw to-device-policy
      | | +--rw acls
      | | | +--rw access-list* [name]
      | | | | +--rw name      -> /acl:acls/acl/name
    augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches:
      +--rw mud
        +--rw manufacturer?    inet:host
        +--rw same-manufacturer? empty
        +--rw model?           inet:uri
        +--rw local-networks?  empty
        +--rw controller?      inet:uri
        +--rw my-controller?    empty
      augment
        /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches
        /acl:l4/acl:tcp/acl:tcp:
          +--rw direction-initiated? direction

```

3. MUD model definitions for the root mud container

3.1. mud-version

This node specifies the integer version of the MUD specification.
 This memo specifies version 1.

3.2. mud-url

This URL identifies the MUD file. This is useful when the file and associated signature are manually uploaded, say, in an offline mode.

3.3. to-device-policy and from-device-policy containers

[I-D.ietf-netmod-acl-model] describes access-lists. In the case of MUD, a MUD file must be explicit in describing the communication pattern of a Thing, and that includes indicating what is to be permitted or denied in either direction of communication. Hence each of these containers indicates the appropriate direction of a flow in association with a particular Thing. They contain references to specific access-lists.

3.4. last-update

This is a date-and-time value of when the MUD file was generated. This is akin to a version number. Its form is taken from [RFC6991] which, for those keeping score, in turn was taken from Section 5.6 of [RFC3339], which was taken from [ISO.8601.1988].

3.5. cache-validity

This uint8 is the period of time in hours that a network management station MUST wait since its last retrieval before checking for an update. It is RECOMMENDED that this value be no less than 24 and MUST NOT be more than 168 for any Thing that is supported. This period SHOULD be no shorter than any period determined through HTTP caching directives (e.g., "cache-control" or "Expires"). N.B., expiring of this timer does not require the MUD manager to discard the MUD file, nor terminate access to a Thing. See Section 16 for more information.

3.6. is-supported

This boolean is an indication from the manufacturer to the network administrator as to whether or not the Thing is supported. In this context a Thing is said to not be supported if the manufacturer intends never to issue a firmware or software update to the Thing or never update the MUD file. A MUD manager MAY still periodically check for updates.

3.7. systeminfo

This is a textual UTF-8 description of the Thing to be connected. The intent is for administrators to be able to see a brief

displayable description of the Thing. It SHOULD NOT exceed 60 characters worth of display space.

3.8. mfg-name, software-rev, model-name firmware-rev

These optional fields are filled in as specified by [RFC8348]. Note that firmware-rev and software-rev MUST NOT be populated in a MUD file if the device can be upgraded but the MUD-URL cannot be. This would be the case, for instance, with MUD-URLs that are contained in 802.1AR certificates.

3.9. extensions

This optional leaf-list names MUD extensions that are used in the MUD file. Note that MUD extensions MUST NOT be used in a MUD file without the extensions being declared. Implementations MUST ignore any node in this file that they do not understand.

Note that extensions can either extend the MUD file as described in the previous paragraph, or they might reference other work. An extension example can be found in Appendix C.

4. Augmentation to the ACL Model

Note that in this section, when we use the term "match" we are referring to the ACL model "matches" node.

4.1. manufacturer

This node consists of a hostname that would be matched against the authority component of another Thing's MUD URL. In its simplest form "manufacturer" and "same-manufacturer" may be implemented as access-lists. In more complex forms, additional network capabilities may be used. For example, if one saw the line "manufacturer" : "flobbidy.example.com", then all Things that registered with a MUD URL that contained flobbity.example.com in its authority section would match.

4.2. same-manufacturer

This null-valued node is an equivalent for when the manufacturer element is used to indicate the authority that is found in another Thing's MUD URL matches that of the authority found in this Thing's MUD URL. For example, if the Thing's MUD URL were <https://bl.example.com/ThingV1>, then all devices that had MUD URL with an authority section of bl.example.com would match.

4.3. documentation

This URI consists of a URL that points to documentation relating to the device and the MUD file. This can prove particularly useful when the "controller" class is used, so that its use can be explained.

4.4. model

This string matches the entire MUD URL, thus covering the model that is unique within the context of the authority. It may contain not only model information, but versioning information as well, and any other information that the manufacturer wishes to add. The intended use is for devices of this precise class to match, to permit or deny communication between one another.

4.5. local-networks

This null-valued node expands to include local networks. Its default expansion is that packets must not traverse toward a default route that is received from the router. However, administrators may expand the expression as is appropriate in their deployments.

4.6. controller

This URI specifies a value that a controller will register with the MUD manager. The node then is expanded to the set of hosts that are so registered. This node may also be a URN. In this case, the URN describes a well known service, such as DNS or NTP, that has been standardized. Both of those URNs may be found in Section 17.6.

When "my-controller" is used, it is possible that the administrator will be prompted to populate that class for each and every model. Use of "controller" with a named class allows the user to populate that class only once for many different models that a manufacturer may produce.

Controller URIs MAY take the form of a URL (e.g. "http[s]://"). However, MUD managers MUST NOT resolve and retrieve such files, and it is RECOMMENDED that there be no such file at this time, as their form and function may be defined at a point in the future. For now, URLs should serve simply as class names and may be populated by the local deployment administrator.

Great care should be taken by MUD managers when invoking the controller class in the form of URLs. For one thing, it requires some understanding by the administrator as to when it is appropriate. Pre-registration in such classes by controllers with the MUD server

is encouraged. The mechanism to do that is beyond the scope of this work.

4.7. my-controller

This null-valued node signals to the MUD manager to use whatever mapping it has for this MUD URL to a particular group of hosts. This may require prompting the administrator for class members. Future work should seek to automate membership management.

4.8. direction-initiated

This MUST only be applied to TCP. This matches the direction in which a TCP connection is initiated. When direction initiated is "from-device", packets that are transmitted in the direction of a thing MUST be dropped unless the thing has first initiated a TCP connection. By way of example, this node may be implemented in its simplest form by looking at naked SYN bits, but may also be implemented through more stateful mechanisms.

When applied this matches packets when the flow was initiated in the corresponding direction. [RFC6092] specifies IPv6 guidance best practices. While that document is scoped specifically to IPv6, its contents are applicable for IPv4 as well.

5. Processing of the MUD file

To keep things relatively simple in addition to whatever definitions exist, we also apply two additional default behaviors:

- o Anything not explicitly permitted is denied.
- o Local DNS and NTP are, by default, permitted to and from the Thing.

An explicit description of the defaults can be found in Appendix B. These are applied AFTER all other explicit rules. Thus, a default behavior can be changed with a "drop" action.

6. What does a MUD URL look like?

MUD URLs are required to use the HTTPS scheme, in order to establish the MUD file server's identity and assure integrity of the MUD file.

Any "https://" URL can be a MUD URL. For example:

```
https://things.example.org/product_abc123/v5
https://www.example.net/mudfiles/temperature_sensor/
https://example.com/lightbulbs/colour/v1
```

A manufacturer may construct a MUD URL in any way, so long as it makes use of the "https" schema.

7. The MUD YANG Model

```
<CODE BEGINS>file "ietf-mud@2018-06-15.yang"
module ietf-mud {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-mud";
  prefix ietf-mud;

  import ietf-access-control-list {
    prefix acl;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import ietf-inet-types {
    prefix inet;
  }

  organization
    "IETF OPSAWG (Ops Area) Working Group";
  contact
    "WG Web: http://tools.ietf.org/wg/opsawg/
    WG List: opsawg@ietf.org
    Author: Eliot Lear
    lear@cisco.com
    Author: Ralph Droms
    rdroms@gmail.com
    Author: Dan Romascanu
    dromasca@gmail.com

    ";
  description
    "This YANG module defines a component that augments the
    IETF description of an access list. This specific module
    focuses on additional filters that include local, model,
    and same-manufacturer.

    This module is intended to be serialized via JSON and stored
    as a file, as described in RFC XXXX [RFC Editor to fill in with
    this document #]."
```

Copyright (c) 2016,2017 IETF Trust and the persons identified as the document authors. 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 (<http://trustee.ietf.org/license-info>). This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

```
revision 2018-06-15 {
  description
    "Initial proposed standard.";
  reference
    "RFC XXXX: Manufacturer Usage Description
    Specification";
}

typedef direction {
  type enumeration {
    enum to-device {
      description
        "packets or flows destined to the target
        Thing";
    }
    enum from-device {
      description
        "packets or flows destined from
        the target Thing";
    }
  }
  description
    "Which way are we talking about?";
}

container mud {
  presence "Enabled for this particular MUD URL";
  description
    "MUD related information, as specified
    by RFC-XXXX [RFC Editor to fill in].";
  uses mud-grouping;
}

grouping mud-grouping {
  description
    "Information about when support end(ed), and
    when to refresh";
```

```
leaf mud-version {
  type uint8;
  mandatory true;
  description
    "This is the version of the MUD
    specification.  This memo specifies version 1.";
}
leaf mud-url {
  type inet:uri;
  mandatory true;
  description
    "This is the MUD URL associated with the entry found
    in a MUD file.";
}
leaf last-update {
  type yang:date-and-time;
  mandatory true;
  description
    "This is intended to be when the current MUD file
    was generated.  MUD Managers SHOULD NOT check
    for updates between this time plus cache validity";
}
leaf mud-signature {
  type inet:uri;
  description
    "A URI that resolves to a signature as
    described in this specification.";
}
leaf cache-validity {
  type uint8 {
    range "1..168";
  }
  units "hours";
  default "48";
  description
    "The information retrieved from the MUD server is
    valid for these many hours, after which it should
    be refreshed.  N.B. MUD manager implementations
    need not discard MUD files beyond this period.";
}
leaf is-supported {
  type boolean;
  mandatory true;
  description
    "This boolean indicates whether or not the Thing is
    currently supported by the manufacturer.";
}
leaf systeminfo {
```

```
    type string;
    description
        "A UTF-8 description of this Thing.  This
        should be a brief description that may be
        displayed to the user to determine whether
        to allow the Thing on the
        network.";
}
leaf mfg-name {
    type string;
    description
        "Manufacturer name, as described in
        the ietf-hardware YANG module.";
}
leaf model-name {
    type string;
    description
        "Model name, as described in the
        ietf-hardware YANG module.";
}
leaf firmware-rev {
    type string;
    description
        "firmware-rev, as described in the
        ietf-hardware YANG module.  Note this field MUST
        NOT be included when the device can be updated
        but the MUD-URL cannot.";
}
leaf software-rev {
    type string;
    description
        "software-rev, as described in the
        ietf-hardware YANG module.  Note this field MUST
        NOT be included when the device can be updated
        but the MUD-URL cannot.";
}
leaf documentation {
    type inet:uri;
    description
        "This URL points to documentation that
        relates to this device and any classes that it uses
        in its MUD file.  A caution: MUD managers need
        not resolve this URL on their own, but rather simply
        provide it to the administrator.  Parsing HTML is
        not an intended function of a MUD manager.";
}
leaf-list extensions {
    type string {
```

```
        length "1..40";
    }
    description
        "A list of extension names that are used in this MUD
        file. Each name is registered with the IANA and
        described in an RFC.";
    }
    container from-device-policy {
        description
            "The policies that should be enforced on traffic
            coming from the device. These policies are not
            necessarily intended to be enforced at a single
            point, but may be rendered by the controller to any
            relevant enforcement points in the network or
            elsewhere.";
        uses access-lists;
    }
    container to-device-policy {
        description
            "The policies that should be enforced on traffic
            going to the device. These policies are not
            necessarily intended to be enforced at a single
            point, but may be rendered by the controller to any
            relevant enforcement points in the network or
            elsewhere.";
        uses access-lists;
    }
}

grouping access-lists {
    description
        "A grouping for access lists in the context of device
        policy.";
    container access-lists {
        description
            "The access lists that should be applied to traffic
            to or from the device.";
        list access-list {
            key "name";
            description
                "Each entry on this list refers to an ACL that
                should be present in the overall access list
                data model. Each ACL is identified by name and
                type.";
            leaf name {
                type leafref {
                    path "/acl:acls/acl:acl/acl:name";
                }
            }
        }
    }
}
```



```
        description
          "The name of the ACL for this entry.";
      }
  }
}

augment "/acl:acls/acl:acl/acl:aces/acl:ace/acl:matches" {
  description
    "adding abstractions to avoid need of IP addresses";
  container mud {
    description
      "MUD-specific matches.";
    leaf manufacturer {
      type inet:host;
      description
        "A domain that is intended to match the authority
        section of the MUD URL. This node is used to specify
        one or more manufacturers a device should
        be authorized to access.";
    }
    leaf same-manufacturer {
      type empty;
      description
        "This node matches the authority section of the MUD URL
        of a Thing. It is intended to grant access to all
        devices with the same authority section.";
    }
    leaf model {
      type inet:uri;
      description
        "Devices of the specified model type will match if
        they have an identical MUD URL.";
    }
    leaf local-networks {
      type empty;
      description
        "IP addresses will match this node if they are
        considered local addresses. A local address may be
        a list of locally defined prefixes and masks
        that indicate a particular administrative scope.";
    }
    leaf controller {
      type inet:uri;
      description
        "This node names a class that has associated with it
        zero or more IP addresses to match against. These
        may be scoped to a manufacturer or via a standard
```

```

        URN.";
    }
    leaf my-controller {
        type empty;
        description
            "This node matches one or more network elements that
             have been configured to be the controller for this
             Thing, based on its MUD URL.";
    }
}
}
}
augment "/acl:acls/acl:acl/acl:aces/acl:ace/acl:matches" +
    "/acl:l4/acl:tcp/acl:tcp" {
    description
        "add direction-initiated";
    leaf direction-initiated {
        type direction;
        description
            "This node matches based on which direction a
             connection was initiated. The means by which that
             is determined is discussed in this document.";
    }
}
}
}
}
<CODE ENDS>

```

8. The Domain Name Extension to the ACL Model

This module specifies an extension to IETF-ACL model such that domain names may be referenced by augmenting the "matches" node. Different implementations may deploy differing methods to maintain the mapping between IP address and domain name, if indeed any are needed. However, the intent is that resources that are referred to using a name should be authorized (or not) within an access list.

The structure of the change is as follows:

```

module: ietf-acldns
    augment /acl:acls/acl:acl/acl:aces/acl:ace/
        acl:matches/acl:l3/acl:ipv4/acl:ipv4:
            +--rw src-dnsname?    inet:host
            +--rw dst-dnsname?    inet:host
    augment /acl:acls/acl:acl/acl:aces/acl:ace/
        acl:matches/acl:l3/acl:ipv6/acl:ipv6:
            +--rw src-dnsname?    inet:host
            +--rw dst-dnsname?    inet:host

```

The choice of these particular points in the access-list model is based on the assumption that we are in some way referring to IP-related resources, as that is what the DNS returns. A domain name in our context is defined in [RFC6991]. The augmentations are replicated across IPv4 and IPv6 to allow MUD file authors the ability to control the IP version that the Thing may utilize.

The following node are defined.

8.1. src-dnsname

The argument corresponds to a domain name of a source as specified by inet:host. A number of means may be used to resolve hosts. What is important is that such resolutions be consistent with ACLs required by Things to properly operate.

8.2. dst-dnsname

The argument corresponds to a domain name of a destination as specified by inet:host See the previous section relating to resolution.

Note when using either of these with a MUD file, because access is associated with a particular Thing, MUD files MUST NOT contain either a src-dnsname in an ACL associated with from-device-policy or a dst-dnsname associated with to-device-policy.

8.3. The ietf-acldns Model

```
<CODE BEGINS>file "ietf-acldns@2018-06-15.yang"
module ietf-acldns {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-acldns";
  prefix ietf-acldns;

  import ietf-access-control-list {
    prefix acl;
  }
  import ietf-inet-types {
    prefix inet;
  }

  organization
    "IETF OPSAWG (Ops Area) Working Group";
  contact
    "WG Web: http://tools.ietf.org/wg/opsawg/
    WG List: opsawg@ietf.org
    Author: Eliot Lear
```

```
    lear@cisco.com
    Author: Ralph Droms
    rdroms@gmail.com
    Author: Dan Romascanu
    dromasca@gmail.com
    ";
description
    "This YANG module defines a component that augments the
    IETF description of an access list to allow DNS names
    as matching criteria.";

revision 2018-06-15 {
    description
        "Base version of dnsname extension of ACL model";
    reference
        "RFC XXXX: Manufacturer Usage Description
        Specification";
}

grouping dns-matches {
    description
        "Domain names for matching.";
    leaf src-dnsname {
        type inet:host;
        description
            "domain name to be matched against";
    }
    leaf dst-dnsname {
        type inet:host;
        description
            "domain name to be matched against";
    }
}

augment "/acl:acls/acl:acl/acl:aces/acl:ace/acl:matches" +
"/acl:l3/acl:ipv4/acl:ipv4" {
    description
        "Adding domain names to matching";
    uses dns-matches;
}
augment "/acl:acls/acl:acl/acl:aces/acl:ace/acl:matches" +
"/acl:l3/acl:ipv6/acl:ipv6" {
    description
        "Adding domain names to matching";
    uses dns-matches;
}
}
<CODE ENDS>
```

9. MUD File Example

This example contains two access lists that are intended to provide outbound access to a cloud service on TCP port 443.

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://lighting.example.com/lightbulb2000",
    "last-update": "2018-03-02T11:20:51+01:00",
    "cache-validity": 48,
    "is-supported": true,
    "systeminfo": "The BMS Example Lightbulb",
    "from-device-policy": {
      "access-lists": {
        "access-list": [
          {
            "name": "mud-76100-v6fr"
          }
        ]
      }
    },
    "to-device-policy": {
      "access-lists": {
        "access-list": [
          {
            "name": "mud-76100-v6to"
          }
        ]
      }
    }
  },
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "mud-76100-v6to",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "cl0-todev",
              "matches": {
                "ipv6": {
                  "ietf-acldns:src-dnsname": "test.example.com",
                  "protocol": 6
                },
              },
              "tcp": {
                "ietf-mud:direction-initiated": "from-device",

```

```

        "source-port": {
            "operator": "eq",
            "port": 443
        }
    },
    "actions": {
        "forwarding": "accept"
    }
}
]
},
{
    "name": "mud-76100-v6fr",
    "type": "ipv6-acl-type",
    "aces": [
        {
            "ace": [
                {
                    "name": "cl0-frdev",
                    "matches": {
                        "ipv6": {
                            "ietf-acldns:dst-dnsname": "test.example.com",
                            "protocol": 6
                        },
                        "tcp": {
                            "ietf-mud:direction-initiated": "from-device",
                            "destination-port": {
                                "operator": "eq",
                                "port": 443
                            }
                        }
                    },
                    "actions": {
                        "forwarding": "accept"
                    }
                }
            ]
        }
    ]
}
]
```

In this example, two policies are declared, one from the Thing and the other to the Thing. Each policy names an access list that applies to the Thing, and one that applies from. Within each access

list, access is permitted to packets flowing to or from the Thing that can be mapped to the domain name of "service.bms.example.com". For each access list, the enforcement point should expect that the Thing initiated the connection.

10. The MUD URL DHCP Option

The IPv4 MUD URL client option has the following format:

```
+-----+-----+-----+
| code | len | MUDstring
+-----+-----+-----+
```

Code `OPTION_MUD_URL_V4` (161) is assigned by IANA. `len` is a single octet that indicates the length of MUD string in octets. The MUD string is defined as follows:

```
MUDstring = mudurl [ " " reserved ]
mudurl = URI; a URL [RFC3986] that uses the "https" schema [RFC7230]
reserved = 1*( OCTET ) ; from [RFC5234]
```

The entire option MUST NOT exceed 255 octets. If a space follows the MUD URL, a reserved string that will be defined in future specifications follows. MUD managers that do not understand this field MUST ignore it.

The IPv6 MUD URL client option has the following format:

```

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 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|          OPTION_MUD_URL_V6          |          option-length          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     MUDstring                             |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

`OPTION_MUD_URL_V6` (112; assigned by IANA).

`option-length` contains the length of the MUDstring, as defined above, in octets.

The intent of this option is to provide both a new Thing classifier to the network as well as some recommended configuration to the routers that implement policy. However, it is entirely the purview

of the network system as managed by the network administrator to decide what to do with this information. The key function of this option is simply to identify the type of Thing to the network in a structured way such that the policy can be easily found with existing toolsets.

10.1. Client Behavior

A DHCPv4 client MAY emit a DHCPv4 option and a DHCPv6 client MAY emit DHCPv6 option. These options are singletons, as specified in [RFC7227]. Because clients are intended to have at most one MUD URL associated with them, they may emit at most one MUD URL option via DHCPv4 and one MUD URL option via DHCPv6. In the case where both v4 and v6 DHCP options are emitted, the same URL MUST be used.

10.2. Server Behavior

A DHCP server may ignore these options or take action based on receipt of these options. When a server consumes this option, it will either forward the URL and relevant client information (such as the gateway address or giaddr and requested IP address, and lease length) to a network management system, or it will retrieve the usage description itself by resolving the URL.

DHCP servers may implement MUD functionality themselves or they may pass along appropriate information to a network management system or MUD manager. A DHCP server that does process the MUD URL MUST adhere to the process specified in [RFC2818] and [RFC5280] to validate the TLS certificate of the web server hosting the MUD file. Those servers will retrieve the file, process it, create and install the necessary configuration on the relevant network element. Servers SHOULD monitor the gateway for state changes on a given interface. A DHCP server that does not provide MUD functionality and has forwarded a MUD URL to a MUD manager MUST notify the MUD manager of any corresponding change to the DHCP state of the client (such as expiration or explicit release of a network address lease).

Should the DHCP server fail, in the case when it implements the MUD manager functionality, any backup mechanisms SHOULD include the MUD state, and the server SHOULD resolve the status of clients upon its restart, similar to what it would do, absent MUD manager functionality. In the case where the DHCP server forwards information to the MUD manager, the MUD manager will either make use of redundant DHCP servers for information, or otherwise clear state based on other network information, such as monitoring port status on a switch via SNMP, Radius accounting, or similar mechanisms.

10.3. Relay Requirements

There are no additional requirements for relays.

11. The Manufacturer Usage Description (MUD) URL X.509 Extension

This section defines an X.509 non-critical certificate extension that contains a single Uniform Resource Locator (URL) that points to an on-line Manufacturer Usage Description concerning the certificate subject. URI must be represented as described in Section 7.4 of [RFC5280].

Any Internationalized Resource Identifiers (IRIs) MUST be mapped to URIs as specified in Section 3.1 of [RFC3987] before they are placed in the certificate extension.

The semantics of the URL are defined Section 6 of this document.

The choice of id-pe is based on guidance found in Section 4.2.2 of [RFC5280]:

These extensions may be used to direct applications to on-line information about the issuer or the subject.

The MUD URL is precisely that: online information about the particular subject.

In addition, a separate new extension is defined as id-pe-mudsigner. This contains the subject field of the signing certificate of the MUD file. Processing of this field is specified in Section 13.2.

The purpose of this signature is to make a claim that the MUD file found on the server is valid for a given device, independent of any other factors. There are several security considerations below in Section 16.

A new content-type id-ct-mud is also defined. While signatures are detached today, should a MUD file be transmitted as part of a CMS message, this content-type SHOULD be used.

The new extension is identified as follows:

<CODE BEGINS>

```
MUDURLExtnModule-2016 { iso(1) identified-organization(3) dod(6)
    internet(1) security(5) mechanisms(5) pkix(7)
    id-mod(0) id-mod-mudURLExtn2016(88) }
DEFINITIONS IMPLICIT TAGS ::= BEGIN
```

```
-- EXPORTS ALL --

IMPORTS

-- RFC 5912
EXTENSION
FROM PKIX-CommonTypes-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkixCommon-02(57) }

-- RFC 5912
id-ct
FROM PKIXCRMF-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-crmf2005-02(55) }

-- RFC 6268
CONTENT-TYPE
FROM CryptographicMessageSyntax-2010
    { iso(1) member-body(2) us(840) rsadsi(113549)
      pkcs(1) pkcs-9(9) smime(16) modules(0) id-mod-cms-2009(58) }

-- RFC 5912
id-pe, Name
FROM PKIX1Explicit-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-explicit-02(51) } ;

--
-- Certificate Extensions
--

MUDCertExtensions EXTENSION ::=
    { ext-MUDURL | ext-MUDsigner, ... }

ext-MUDURL EXTENSION ::=
    { SYNTAX MUDURLSyntax IDENTIFIED BY id-pe-mud-url }

id-pe-mud-url OBJECT IDENTIFIER ::= { id-pe 25 }

MUDURLSyntax ::= IA5String

ext-MUDsigner EXTENSION ::=
    { SYNTAX MUDsignerSyntax IDENTIFIED BY id-pe-mudsigner }
```

```

id-pe-mudsigner OBJECT IDENTIFIER ::= { id-pe TBD1 }

MUDsignerSyntax ::= Name

--
-- CMS Content Types
--

MUDContentTypes CONTENT-TYPE ::=
    { ct-mud, ... }

ct-mud CONTENT-TYPE ::=
    { -- directly include the content
      IDENTIFIED BY id-ct-mudtype }
    -- The binary data that is in the form
    -- 'application/mud+json' is directly encoded as the
    -- signed data. No additional ASN.1 encoding is added.

id-ct-mudtype OBJECT IDENTIFIER ::= { id-ct TBD2 }

END
<CODE ENDS>

```

While this extension can appear in either an 802.AR manufacturer certificate (IDevID) or deployment certificate (LDevID), of course it is not guaranteed in either, nor is it guaranteed to be carried over. It is RECOMMENDED that MUD manager implementations maintain a table that maps a Thing to its MUD URL based on IDevIDs.

12. The Manufacturer Usage Description LLDP extension

The IEEE802.1AB Link Layer Discovery Protocol (LLDP) is a one hop vendor-neutral link layer protocol used by end hosts network Things for advertising their identity, capabilities, and neighbors on an IEEE 802 local area network. Its Type-Length-Value (TLV) design allows for 'vendor-specific' extensions to be defined. IANA has a registered IEEE 802 organizationally unique identifier (OUI) defined as documented in [RFC7042]. The MUD LLDP extension uses a subtype defined in this document to carry the MUD URL.

The LLDP vendor specific frame has the following format:

TLV Type	len	OUI	subtype	MUDString
=127		= 00 00 5E	= 1	
(7 bits)	(9 bits)	(3 octets)	(1 octet)	(1-255 octets)

where:

- o TLV Type = 127 indicates a vendor-specific TLV
- o len - indicates the TLV string length
- o OUI = 00 00 5E is the organizationally unique identifier of IANA
- o subtype = 1 (to be assigned by IANA for the MUD URL)
- o MUD URL - the length MUST NOT exceed 255 octets

The intent of this extension is to provide both a new Thing classifier to the network as well as some recommended configuration to the routers that implement policy. However, it is entirely the purview of the network system as managed by the network administrator to decide what to do with this information. The key function of this extension is simply to identify the type of Thing to the network in a structured way such that the policy can be easily found with existing toolsets.

Hosts, routers, or other network elements that implement this option are intended to have at most one MUD URL associated with them, so they may transmit at most one MUD URL value.

Hosts, routers, or other network elements that implement this option may ignore these options or take action based on receipt of these options. For example they may fill in information in the respective extensions of the LLDP Management Information Base (LLDP MIB). LLDP operates in a one-way direction. LLDPDUs are not exchanged as information requests by one Thing and response sent by another Thing. The other Things do not acknowledge LLDP information received from a Thing. No specific network behavior is guaranteed. When a Thing consumes this extension, it may either forward the URL and relevant remote Thing information to a MUD manager, or it will retrieve the usage description by resolving the URL in accordance with normal HTTP semantics.

13. Creating and Processing of Signed MUD Files

Because MUD files contain information that may be used to configure network access lists, they are sensitive. To ensure that they have not been tampered with, it is important that they be signed. We make use of DER-encoded Cryptographic Message Syntax (CMS) [RFC5652] for this purpose.

13.1. Creating a MUD file signature

A MUD file MUST be signed using CMS as an opaque binary object. In order to make successful verification more likely, intermediate certificates SHOULD be included. The signature is stored at the location specified in the MUD file. Signatures are transferred using content-type "application/pkcs7-signature".

For example:

```
% openssl cms -sign -signer mancrtfile -inkey mankey \  
               -in mudfile -binary -outform DER -binary \  
               -certfile intermediatecert -out mudfile.p7s
```

Note: A MUD file may need to be re-signed if the signature expires.

13.2. Verifying a MUD file signature

Prior to processing the rest of a MUD file, the MUD manager MUST retrieve the MUD signature file by retrieving the value of "mud-signature" and validating the signature across the MUD file. The Key Usage Extension in the signing certificate MUST be present and have the bit digitalSignature(0) set. When the id-pe-mudsigner extension is present in a device's X.509 certificate, the MUD signature file MUST have been generated by a certificate whose subject matches the contents of that id-pe-mudsigner extension. If these conditions are not met, or if it cannot validate the chain of trust to a known trust anchor, the MUD manager MUST cease processing the MUD file until an administrator has given approval.

The purpose of the signature on the file is to assign accountability to an entity, whose reputation can be used to guide administrators on whether or not to accept a given MUD file. It is already common place to check web reputation on the location of a server on which a file resides. While it is likely that the manufacturer will be the signer of the file, this is not strictly necessary, and may not be desirable. For one thing, in some environments, integrators may install their own certificates. For another, what is more important is the accountability of the recommendation, and not just the relationship between the Thing and the file.

An example:

```
% openssl cms -verify -in mudfile.p7s -inform DER -content mudfile
```

Note the additional step of verifying the common trust root.

14. Extensibility

One of our design goals is to see that MUD files are able to be understood by as broad a cross-section of systems as is possible. Coupled with the fact that we have also chosen to leverage existing mechanisms, we are left with no ability to negotiate extensions and a limited desire for those extensions in any event. A such, a two-tier extensibility framework is employed, as follows:

1. At a coarse grain, a protocol version is included in a MUD URL. This memo specifies MUD version 1. Any and all changes are entertained when this version is bumped. Transition approaches between versions would be a matter for discussion in future versions.
2. At a finer grain, only extensions that would not incur additional risk to the Thing are permitted. Specifically, adding nodes to the mud container is permitted with the understanding that such additions will be ignored by unaware implementations. Any such extensions SHALL be standardized through the IETF process, and MUST be named in the "extensions" list. MUD managers MUST ignore YANG nodes they do not understand and SHOULD create an exception to be resolved by an administrator, so as to avoid any policy inconsistencies.

15. Deployment Considerations

Because MUD consists of a number of architectural building blocks, it is possible to assemble different deployment scenarios. One key aspect is where to place policy enforcement. In order to protect the Thing from other Things within a local deployment, policy can be enforced on the nearest switch or access point. In order to limit unwanted traffic within a network, it may also be advisable to enforce policy as close to the Internet as possible. In some circumstances, policy enforcement may not be available at the closest hop. At that point, the risk of lateral infection (infection of devices that reside near one another) is increased to the number of Things that are able to communicate without protection.

A caution about some of the classes: admission of a Thing into the "manufacturer" and "same-manufacturer" class may have impact on access of other Things. Put another way, the admission may grow the access-list on switches connected to other Things, depending on how access is managed. Some care should be given on managing that access-list growth. Alternative methods such as additional network segmentation can be used to keep that growth within reason.

Because as of this writing MUD is a new concept, one can expect a great many devices to not have implemented it. It remains a local deployment decision as to whether a device that is first connected should be allowed broad or limited access. Furthermore, as mentioned in the introduction, a deployment may choose to ignore a MUD policy in its entirety, but simply taken into account the MUD URL as a classifier to be used as part of a local policy decision.

Finally, please see directly below regarding device lifetimes and use of domain names.

16. Security Considerations

Based on how a MUD URL is emitted, a Thing may be able to lie about what it is, thus gaining additional network access. This can happen in a number of ways when a device emits a MUD URL using DHCP or LLDP, such as being inappropriately admitted to a class such as "same-manufacturer", given access to a device such as "my-controller", or being permitted access to an Internet resource, where such access would otherwise be disallowed. Whether that is the case will depend on the deployment. Implementations SHOULD be configurable to disallow additive access for devices using MUD-URLs that are not emitted in a secure fashion such as in a certificate. Similarly, implementations SHOULD NOT grant elevated permissions (beyond those of devices presenting no MUD policy) to devices which do not strongly bind their identity to their L2/L3 transmissions. When insecure methods are used by the MUD Manager, the classes SHOULD NOT contain devices that use both insecure and secure methods, in order to prevent privilege escalation attacks, and MUST NOT contain devices with the same MUD-URL that are derived from both strong and weak authentication methods.

Devices may forge source (L2/L3) information. Deployments should apply appropriate protections to bind communications to the authentication that has taken place. For 802.1X authentication, IEEE 802.1AE (MACsec) [IEEE8021AE] is one means by which this may happen. A similar approach can be used with 802.11i (WPA2) [IEEE80211i]. Other means are available with other lower layer technologies. Implementations using session-oriented access that is not cryptographically bound should take care to remove state when any form of break in the session is detected.

A rogue CA may sign a certificate that contains the same subject name as is listed in the MUDsigner field in the manufacturer certificate, thus seemingly permitting a substitute MUD file for a device. There are two mitigations available: first, if the signer changes, this may be flagged as an exception by the MUD manager. If the MUD file also changes, the MUD manager SHOULD seek administrator approval (it

should do this in any case). In all circumstances, the MUD manager MUST maintain a cache of trusted CAs for this purpose. When such a rogue is discovered, it SHOULD be removed.

Additional mitigations are described below.

When certificates are not present, Things claiming to be of a certain manufacturer SHOULD NOT be included in that manufacturer grouping without additional validation of some form. This will be relevant when the MUD manager makes use of primitives such as "manufacturer" for the purpose of accessing Things of a particular type. Similarly, network management systems may be able to fingerprint the Thing. In such cases, the MUD URL can act as a classifier that can be proven or disproven. Fingerprinting may have other advantages as well: when 802.1AR certificates are used, because they themselves cannot change, fingerprinting offers the opportunity to add artifacts to the MUD string in the form of the reserved field discussed in Section 10. The meaning of such artifacts is left as future work.

MUD managers SHOULD NOT accept a usage description for a Thing with the same MAC address that has indicated a change of the URL authority without some additional validation (such as review by a network administrator). New Things that present some form of unauthenticated MUD URL SHOULD be validated by some external means when they would be given increased network access.

It may be possible for a rogue manufacturer to inappropriately exercise the MUD file parser, in order to exploit a vulnerability. There are three recommended approaches to address this threat. The first is to validate that the signer of the MUD file is known to and trusted by the MUD manager. The second is to have a system do a primary scan of the file to ensure that it is both parseable and believable at some level. MUD files will likely be relatively small, to start with. The number of ACEs used by any given Thing should be relatively small as well. It may also be useful to limit retrieval of MUD URLs to only those sites that are known to have decent web or domain reputations.

Use of a URL necessitates the use of domain names. If a domain name changes ownership, the new owner of that domain may be able to provide MUD files that MUD managers would consider valid. There are a few approaches that can mitigate this attack. First, MUD managers SHOULD cache certificates used by the MUD file server. When a new certificate is retrieved for whatever reason, the MUD manager should check to see if ownership of the domain has changed. A fair programmatic approximation of this is when the name servers for the domain have changed. If the actual MUD file has changed, the MUD manager MAY check the WHOIS database to see if registration ownership

of a domain has changed. If a change has occurred, or if for some reason it is not possible to determine whether ownership has changed, further review may be warranted. Note, this remediation does not take into account the case of a Thing that was produced long ago and only recently fielded, or the case where a new MUD manager has been installed.

The release of a MUD URL by a Thing reveals what the Thing is, and provides an attacker with guidance on what vulnerabilities may be present.

While the MUD URL itself is not intended to be unique to a specific Thing, the release of the URL may aid an observer in identifying individuals when combined with other information. This is a privacy consideration.

In addressing both of these concerns, implementors should take into account what other information they are advertising through mechanisms such as mDNS[RFC6872], how a Thing might otherwise be identified, perhaps through how it behaves when it is connected to the network, whether a Thing is intended to be used by individuals or carry personal identifying information, and then apply appropriate data minimization techniques. One approach is to make use of TEAP [RFC7170] as the means to share information with authorized components in the network. Network elements may also assist in limiting access to the MUD URL through the use of mechanisms such as DHCPv6-Shield [RFC7610].

There is the risk of the MUD manager itself being spied on to determine what things are connected to the network. To address this risk, MUD managers may choose to make use of TLS proxies that they trust that would aggregate other information.

Please note that the security considerations mentioned in Section 4.7 of [I-D.ietf-netmod-rfc6087bis] are not applicable in this case because the YANG serialization is not intended to be accessed via NETCONF. However, for those who try to instantiate this model in a network element via NETCONF, all objects in each model in this draft exhibit similar security characteristics as [I-D.ietf-netmod-acl-model]. The basic purpose of MUD is to configure access, and so by its very nature can be disruptive if used by unauthorized parties.

17. IANA Considerations

[There was originally a registry entry for .well-known suffixes. This has been removed from the draft and may be marked as deprecated in the registry. RFC Editor: please remove this comment.]

17.1. YANG Module Registrations

The following YANG modules are requested to be registered in the "IANA Module Names" registry:

The ietf-mud module:

- o Name: ietf-mud
- o URN: urn:ietf:params:xml:ns:yang:ietf-mud
- o Prefix: ietf-mud
- o Registrant contact: The IESG
- o Reference: [RFCXXXX]

The ietf-acldns module:

- o Name: ietf-acldns
- o URI: urn:ietf:params:xml:ns:yang:ietf-acldns
- o Prefix: ietf-acldns
- o Registrant: the IESG
- o Reference: [RFCXXXX]

17.2. DHCPv4 and DHCPv6 Options

The IANA has allocated option 161 in the Dynamic Host Configuration Protocol (DHCP) and Bootstrap Protocol (BOOTP) Parameters registry for the MUD DHCPv4 option, and option 112 for DHCPv6, as described in Section 10.

17.3. PKIX Extensions

IANA is kindly requested to make the following assignments for:

- o The MUDURLExtnModule-2016 ASN.1 module in the "SMI Security for PKIX Module Identifier" registry (1.3.6.1.5.5.7.0).
- o id-pe-mud-url object identifier from the "SMI Security for PKIX Certificate Extension" registry (1.3.6.1.5.5.7.1).
- o id-pe-mudsigner object identifier from the "SMI Security for PKIX Certificate Extension" registry (TBD1).

- o id-ct-mudtype object identifier from the "SMI Security for S/MIME CMS Content Type" registry (TBD2).

The use of these values is specified in Section 11.

17.4. MIME Media-type Registration for MUD files

The following media-type is defined for transfer of MUD file:

- o Type name: application
- o Subtype name: mud+json
- o Required parameters: n/a
- o Optional parameters: n/a
- o Encoding considerations: 8bit; application/mud+json values are represented as a JSON object; UTF-8 encoding MUST be employed. [RFC3629]
- o Security considerations: See Security Considerations of RFCXXXX and [RFC8259] Section 12.
- o Interoperability considerations: n/a
- o Published specification: [RFCXXXX]
- o Applications that use this media type: MUD managers as specified by [RFCXXXX].
- o Fragment identifier considerations: n/a
- o Additional information:
 - Magic number(s): n/a
 - File extension(s): n/a
 - Macintosh file type code(s): n/a
- o Person & email address to contact for further information: Eliot Lear <lear@cisco.com>, Ralph Droms <rdroms@gmail.com>
- o Intended usage: COMMON
- o Restrictions on usage: none
- o Author:
 - Eliot Lear <lear@cisco.com>
 - Ralph Droms <rdroms@gmail.com>
- o Change controller: IESG
- o Provisional registration? (standards tree only): No.

17.5. LLDP IANA TLV Subtype Registry

IANA is requested to create a new registry for IANA Link Layer Discovery Protocol (LLDP) TLV subtype values. The recommended policy for this registry is Expert Review. The maximum number of entries in the registry is 256.

IANA is required to populate the initial registry with the value:

LLDP subtype value = 1 (All the other 255 values should be initially marked as 'Unassigned'.)

Description = the Manufacturer Usage Description (MUD) Uniform Resource Locator (URL)

Reference = < this document >

17.6. The MUD Well Known Universal Resource Name (URNs)

The following parameter registry is requested to be added in accordance with [RFC3553]

Registry name: "urn:ietf:params:mud" is requested.
Specification: this document
Repository: this document
Index value: Encoded identically to a TCP/UDP port service name, as specified in Section 5.1 of [RFC6335]

The following entries should be added to the "urn:ietf:params:mud" name space:

"urn:ietf:params:mud:dns" refers to the service specified by [RFC1123]. "urn:ietf:params:mud:ntp" refers to the service specified by [RFC5905].

17.7. Extensions Registry

The IANA is requested to establish a registry of extensions as follows:

Registry name: MUD extensions registry
Registry policy: Standards action
Standard reference: document
Extension name: UTF-8 encoded string, not to exceed 40 characters.

Each extension MUST follow the rules specified in this specification. As is usual, the IANA issues early allocations based in accordance with [RFC7120].

18. Acknowledgments

The authors would like to thank Einar Nilsen-Nygaard, who singlehandedly updated the model to match the updated ACL model, Bernie Volz, Tom Gindin, Brian Weis, Sandeep Kumar, Thorsten Dahm, John Bashinski, Steve Rich, Jim Bieda, Dan Wing, Joe Clarke, Henk Birkholz, Adam Montville, Jim Schaad, and Robert Sparks for their valuable advice and reviews. Russ Housley entirely rewrote

Section 11 to be a complete module. Adrian Farrel provided the basis for privacy considerations text. Kent Watsen provided a thorough review of the architecture and the YANG model. The remaining errors in this work are entirely the responsibility of the authors.

19. References

19.1. Normative References

- [I-D.ietf-netmod-acl-model]
Jethanandani, M., Huang, L., Agarwal, S., and D. Blair,
"Network Access Control List (ACL) YANG Data Model",
draft-ietf-netmod-acl-model-19 (work in progress), April
2018.
- [IEEE8021AB]
Institute for Electrical and Electronics Engineers, "IEEE
Standard for Local and Metropolitan Area Networks--
Station and Media Access Control Connectivity Discovery",
n.d..
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts -
Application and Support", STD 3, RFC 1123,
DOI 10.17487/RFC1123, October 1989,
<<https://www.rfc-editor.org/info/rfc1123>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2131] Droms, R., "Dynamic Host Configuration Protocol",
RFC 2131, DOI 10.17487/RFC2131, March 1997,
<<https://www.rfc-editor.org/info/rfc2131>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818,
DOI 10.17487/RFC2818, May 2000,
<<https://www.rfc-editor.org/info/rfc2818>>.
- [RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins,
C., and M. Carney, "Dynamic Host Configuration Protocol
for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July
2003, <<https://www.rfc-editor.org/info/rfc3315>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November
2003, <<https://www.rfc-editor.org/info/rfc3629>>.

- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <<https://www.rfc-editor.org/info/rfc3987>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC5911] Hoffman, P. and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, DOI 10.17487/RFC5911, June 2010, <<https://www.rfc-editor.org/info/rfc5911>>.
- [RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <<https://www.rfc-editor.org/info/rfc5912>>.

- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <<https://www.rfc-editor.org/info/rfc7120>>.
- [RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, DOI 10.17487/RFC7227, May 2014, <<https://www.rfc-editor.org/info/rfc7227>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7610] Gont, F., Liu, W., and G. Van de Velde, "DHCPv6-Shield: Protecting against Rogue DHCPv6 Servers", BCP 199, RFC 7610, DOI 10.17487/RFC7610, August 2015, <<https://www.rfc-editor.org/info/rfc7610>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <<https://www.rfc-editor.org/info/rfc8348>>.

19.2. Informative References

- [FW95] Chapman, D. and E. Zwicky, "Building Internet Firewalls", January 1995.
- [I-D.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-20 (work in progress), March 2018.
- [IEEE80211i]
Institute for Electrical and Electronics Engineers, "IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11-Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications- Amendment 6- Medium Access Control (MAC) Security Enhancements", 2004.
- [IEEE8021AE]
Institute for Electrical and Electronics Engineers, "IEEE Standard for Local and Metropolitan Area Networks- Media Access Control (MAC) Security", 2006.
- [IEEE8021AR]
Institute for Electrical and Electronics Engineers, "Secure Device Identity", 1998.
- [IEEE8021X]
Institute for Electrical and Electronics Engineers, "IEEE Standard for Local and metropolitan area networks--Port-Based Network Access Control", 2010.

- [ISO.8601.1988] International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO Standard 8601, June 1988.
- [RFC1984] IAB and IESG, "IAB and IESG Statement on Cryptographic Technology and the Internet", BCP 200, RFC 1984, DOI 10.17487/RFC1984, August 1996, <<https://www.rfc-editor.org/info/rfc1984>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <<https://www.rfc-editor.org/info/rfc3553>>.
- [RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <<https://www.rfc-editor.org/info/rfc6092>>.
- [RFC6872] Gurbani, V., Ed., Burger, E., Ed., Anjali, T., Abdelnur, H., and O. Festor, "The Common Log Format (CLF) for the Session Initiation Protocol (SIP): Framework and Information Model", RFC 6872, DOI 10.17487/RFC6872, February 2013, <<https://www.rfc-editor.org/info/rfc6872>>.
- [RFC7042] Eastlake 3rd, D. and J. Abley, "IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters", BCP 141, RFC 7042, DOI 10.17487/RFC7042, October 2013, <<https://www.rfc-editor.org/info/rfc7042>>.
- [RFC7170] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, DOI 10.17487/RFC7170, May 2014, <<https://www.rfc-editor.org/info/rfc7170>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

- [RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <<https://www.rfc-editor.org/info/rfc7452>>.
- [RFC7488] Boucadair, M., Penno, R., Wing, D., Patil, P., and T. Reddy, "Port Control Protocol (PCP) Server Selection", RFC 7488, DOI 10.17487/RFC7488, March 2015, <<https://www.rfc-editor.org/info/rfc7488>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.

Appendix A. Changes from Earlier Versions

RFC Editor to remove this section prior to publication.

Draft -19: * Edits after discussion with apps area to address reserved field for the future. * Correct systeminfo to be utf8. * Remove "hardware-rev" from list.

Draft -18: * Correct an error in the augment statement * Changes to the ACL model re ports.

Draft -17:

- o One editorial.

Draft -16

- o add mud-signature element based on review comments
- o redo mud-url
- o make clear that systeminfo uses UTF8

Draft -13 to -14:

- o Final WGLC comments and review comments
- o Move version from MUD-URL to Model
- o Have MUD-URL in model
- o Update based on update to draft-ietf-netmod-acl-model
- o Point to tree diagram draft instead of 6087bis.

Draft -12 to -13:

- o Additional WGLC comments

Draft -10 to -12:

These are based on WGLC comments:

- o Correct examples based on ACL model changes.
- o Change ordering nodes.
- o Additional explanatory text around systeminfo.
- o Change ordering in examples.
- o Make it VERY VERY VERY VERY clear that these are recommendations, not mandates.
- o DHCP -> NTP in some of the intro text.
- o Remove masa-server
- o "Things" to "network elements" in a few key places.
- o Reference to JSON YANG RFC added.

Draft -10 to -11:

- o Example corrections
- o Typo
- o Fix two lists.
- o Addition of 'any-acl' and 'mud-acl' in the list of allowed features.
- o Clarification of what should be in a MUD file.

Draft -09 to -10:

- o AD input.
- o Correct dates.
- o Add compliance sentence as to which ACL module features are implemented.

Draft -08 to -09:

- o Resolution of Security Area review, IoT directorate review, GenART review, YANG doctors review.
- o change of YANG structure to address mandatory nodes.
- o Terminology cleanup.
- o specify out extra portion of MUD-URL.
- o consistency changes.
- o improved YANG descriptions.
- o Remove extra revisions.
- o Track ACL model changes.
- o Additional cautions on use of ACL model; further clarifications on extensions.

Draft -07 to -08:

- o a number of editorials corrected.
- o definition of MUD file tweaked.

Draft -06 to -07:

- o Examples updated.
- o Additional clarification for direction-initiated.
- o Additional implementation guidance given.

Draft -06 to -07:

- o Update models to match new ACL model
- o extract directionality from the ACL, introducing a new device container.

Draft -05 to -06:

- o Make clear that this is a component architecture (Polk and Watson)
- o Add order of operations (Watson)

- o Add extensions leaf-list (Pritikin)
- o Remove previous-mud-file (Watson)
- o Modify text in last-update (Watson)
- o Clarify local networks (Weis, Watson)
- o Fix contact info (Watson)
- o Terminology clarification (Weis)
- o Advice on how to handle LDevIDs (Watson)
- o Add deployment considerations (Watson)
- o Add some additional text about fingerprinting (Watson)
- o Appropriate references to 6087bis (Watson)
- o Change systeminfo to a URL to be referenced (Lear)

Draft -04 to -05: * syntax error correction

Draft -03 to -04: * Re-add my-controller

Draft -02 to -03: * Additional IANA updates * Format correction in YANG. * Add reference to TEAP.

Draft -01 to -02: * Update IANA considerations * Accept Russ Housley rewrite of X.509 text * Include privacy considerations text * Redo the URL limit. Still 255 bytes, but now stated in the URL definition. * Change URI registration to be under urn:ietf:params

Draft -00 to -01: * Fix cert trust text. * change supportInformation to meta-info * Add an informational element in. * add urn registry and create first entry * add default elements

Appendix B. Default MUD nodes

What follows is the portion of a MUD file that permits DNS traffic to a controller that is registered with the URN "urn:ietf:params:mud:dns" and traffic NTP to a controller that is registered "urn:ietf:params:mud:ntp". This is considered the default behavior and the ACEs are in effect appended to whatever other "ace" entries that a MUD file contains. To block DNS or NTP one repeats the matching statement but replaces the "forwarding" action "accept" with "drop". Because ACEs are processed in the order they are

received, the defaults would not be reached. A MUD manager might further decide to optimize to simply not include the defaults when they are overridden.

Four "acl" list entries that implement default MUD nodes are listed below. Two are for IPv4 and two are for IPv6 (one in each direction for both versions of IP). Note that neither access-list name nor ace name need be retained or used in any way by local implementations, but are simply there for completeness' sake.

```
"ietf-access-control-list:acls": {
  "acl": [
    {
      "name": "mud-59776-v4to",
      "type": "ipv4-acl-type",
      "aces": {
        "ace": [
          {
            "name": "ent0-todev",
            "matches": {
              "ietf-mud:mud": {
                "controller": "urn:ietf:params:mud:dns"
              },
              "ipv4": {
                "protocol": 17
              },
              "udp": {
                "source-port": {
                  "operator": "eq",
                  "port": 53
                }
              }
            },
            "actions": {
              "forwarding": "accept"
            }
          },
          {
            "name": "ent1-todev",
            "matches": {
              "ietf-mud:mud": {
                "controller": "urn:ietf:params:mud:ntp"
              },
              "ipv4": {
                "protocol": 17
              },
              "udp": {
                "source-port": {
```

```

        "operator": "eq",
        "port": 123
      }
    },
    "actions": {
      "forwarding": "accept"
    }
  ]
},
{
  "name": "mud-59776-v4fr",
  "type": "ipv4-acl-type",
  "aces": {
    "ace": [
      {
        "name": "ent0-frdev",
        "matches": {
          "ietf-mud:mud": {
            "controller": "urn:ietf:params:mud:dns"
          },
          "ipv4": {
            "protocol": 17
          },
          "udp": {
            "destination-port": {
              "operator": "eq",
              "port": 53
            }
          }
        },
        "actions": {
          "forwarding": "accept"
        }
      },
      {
        "name": "ent1-frdev",
        "matches": {
          "ietf-mud:mud": {
            "controller": "urn:ietf:params:mud:ntp"
          },
          "ipv4": {
            "protocol": 17
          },
          "udp": {
            "destination-port": {

```

```

        "operator": "eq",
        "port": 123
      }
    },
    "actions": {
      "forwarding": "accept"
    }
  ]
},
{
  "name": "mud-59776-v6to",
  "type": "ipv6-acl-type",
  "aces": {
    "ace": [
      {
        "name": "ent0-todev",
        "matches": {
          "ietf-mud:mud": {
            "controller": "urn:ietf:params:mud:dns"
          },
          "ipv6": {
            "protocol": 17
          },
          "udp": {
            "source-port": {
              "operator": "eq",
              "port": 53
            }
          }
        },
        "actions": {
          "forwarding": "accept"
        }
      }
    ],
    "name": "ent1-todev",
    "matches": {
      "ietf-mud:mud": {
        "controller": "urn:ietf:params:mud:ntp"
      },
      "ipv6": {
        "protocol": 17
      },
      "udp": {
        "source-port": {

```



```

        "operator": "eq",
        "port": 123
      }
    },
    "actions": {
      "forwarding": "accept"
    }
  ]
},
{
  "name": "mud-59776-v6fr",
  "type": "ipv6-acl-type",
  "aces": {
    "ace": [
      {
        "name": "ent0-frdev",
        "matches": {
          "ietf-mud:mud": {
            "controller": "urn:ietf:params:mud:dns"
          },
          "ipv6": {
            "protocol": 17
          },
          "udp": {
            "destination-port": {
              "operator": "eq",
              "port": 53
            }
          }
        },
        "actions": {
          "forwarding": "accept"
        }
      },
      {
        "name": "ent1-frdev",
        "matches": {
          "ietf-mud:mud": {
            "controller": "urn:ietf:params:mud:ntp"
          },
          "ipv6": {
            "protocol": 17
          },
          "udp": {
            "destination-port": {

```

```

        "operator": "eq",
        "port": 123
      }
    },
    "actions": {
      "forwarding": "accept"
    }
  ]
}

```

Appendix C. A Sample Extension: DETNET-indicator

In this sample extension we augment the core MUD model to indicate whether the device implements DETNET. If a device claims not to use DETNET, but then later attempts to do so, a notification or exception might be generated. Note that this example is intended only for illustrative purposes.

Extension Name: "Example-Extension" (to be used in the extensions list)
 Standard: this document (but do not register the example)

This extension augments the MUD model to include a single node, using the following sample module that has the following tree structure:

```

module: ietf-mud-detext-example
  augment /ietf-mud:mud:
    +--rw is-detnet-required?  boolean

```

The model is defined as follows:

```

<CODE BEGINS>file "ietf-mud-detext-example@2018-06-15.yang"
module ietf-mud-detext-example {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-mud-detext-example";
  prefix ietf-mud-detext-example;

  import ietf-mud {
    prefix ietf-mud;
  }
}

```

```
organization
  "IETF OPSAWG (Ops Area) Working Group";
contact
  "WG Web: http://tools.ietf.org/wg/opsawg/
  WG List: opsawg@ietf.org
  Author: Eliot Lear
  lear@cisco.com
  Author: Ralph Droms
  rdroms@gmail.com
  Author: Dan Romascanu
  dromasca@gmail.com

  ";
description
  "Sample extension to a MUD module to indicate a need
  for DETNET support.";

revision 2018-06-15 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: Manufacturer Usage Description
    Specification";
}

augment "/ietf-mud:mud" {
  description
    "This adds a simple extension for a manufacturer
    to indicate whether DETNET is required by a
    device.";
  leaf is-detnet-required {
    type boolean;
    description
      "This value will equal true if a device requires
      detnet to properly function";
  }
}
}
}
<CODE ENDS>
```

Using the previous example, we now show how the extension would be expressed:

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://lighting.example.com/lightbulb2000",
    "last-update": "2018-03-02T11:20:51+01:00",
```

```
"cache-validity": 48,
"extensions": [
  "ietf-mud-detext-example"
],
"ietf-mud-detext-example:is-detnet-required": "false",
"is-supported": true,
"systeminfo": "The BMS Example Lightbulb",
"from-device-policy": {
  "access-lists": {
    "access-list": [
      {
        "name": "mud-76100-v6fr"
      }
    ]
  }
},
"to-device-policy": {
  "access-lists": {
    "access-list": [
      {
        "name": "mud-76100-v6to"
      }
    ]
  }
},
"ietf-access-control-list:acls": {
  "acl": [
    {
      "name": "mud-76100-v6to",
      "type": "ipv6-acl-type",
      "aces": {
        "ace": [
          {
            "name": "cl0-todev",
            "matches": {
              "ipv6": {
                "ietf-acldns:src-dnsname": "test.example.com",
                "protocol": 6
              },
              "tcp": {
                "ietf-mud:direction-initiated": "from-device",
                "source-port": {
                  "operator": "eq",
                  "port": 443
                }
              }
            }
          }
        ]
      }
    }
  ],
}
```

```

        "actions": {
            "forwarding": "accept"
        }
    }
]
}
},
{
    "name": "mud-76100-v6fr",
    "type": "ipv6-acl-type",
    "aces": {
        "ace": [
            {
                "name": "cl0-frdev",
                "matches": {
                    "ipv6": {
                        "ietf-acldns:dst-dnsname": "test.example.com",
                        "protocol": 6
                    },
                    "tcp": {
                        "ietf-mud:direction-initiated": "from-device",
                        "destination-port": {
                            "operator": "eq",
                            "port": 443
                        }
                    }
                },
                "actions": {
                    "forwarding": "accept"
                }
            }
        ]
    }
}
]
}
}

```

Authors' Addresses

Eliot Lear
Cisco Systems
Richtistrasse 7
Wallisellen CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Ralph Droms
Google
355 Main St., 5th Floor
Cambridge

Phone: +1 978 376 3731
Email: rdroms@gmail.com

Dan Romascanu

Phone: +972 54 5555347
Email: dromasca@gmail.com

OPS Area Working Group
Internet-Draft
Intended status: Informational
Expires: April 15, 2018

Q. Wu
W. Liu
Huawei Technologies
A. Farrel
Juniper Networks
October 12, 2017

Service Models Explained
draft-ietf-opsawg-service-model-explained-05

Abstract

The IETF has produced many modules in the YANG modeling language. The majority of these modules are used to construct data models to model devices or monolithic functions.

A small number of YANG modules have been defined to model services (for example, the Layer Three Virtual Private Network Service Model produced by the L3SM working group and documented in RFC 8049).

This document describes service models as used within the IETF, and also shows where a service model might fit into a Software Defined Networking architecture. Note that service models do not make any assumption of how a service is actually engineered and delivered for a customer; details of how network protocols and devices are engineered to deliver a service are captured in other modules that are not exposed through the Customer-Provider Interface.

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 April 15, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terms and Concepts	4
3. Using Service Models	6
3.1. Practical Considerations	8
4. Service Models in an SDN Context	8
5. Possible Causes of Confusion	11
6. Comparison With Other Work	13
6.1. Comparison With Network Service Models	13
6.2. Service Delivery and Network Element Model Work	15
6.3. Customer Service Model Work	15
6.4. The MEF Architecture	17
7. Further Concepts	18
7.1. Technology Agnostic	18
7.2. Relationship to Policy	18
7.3. Operator-Specific Features	19
7.4. Supporting Multiple Services	19
8. Security Considerations	20
9. Manageability Considerations	20
10. IANA Considerations	21
11. Acknowledgements	21
12. References	21
12.1. Normative References	21
12.2. Informative References	21
Authors' Addresses	23

1. Introduction

In recent years the number of modules written in the YANG modeling language [RFC6020] for configuration and monitoring has blossomed. Many of these are used for device-level configuration (for example,

[RFC7223]) or for control of monolithic functions or protocol instances (for example, [RFC7407]).

[RFC7950] makes a distinction between a "data model" which it defines as describing how data is represented and accessed, and a "YANG module" that defines hierarchies of schema nodes to make a self-contained and compilable block of definitions and inclusions. YANG structures data models into modules for ease of use.

Within the context of Software Defined Networking (SDN) [RFC7149] [RFC7426], YANG data modules may be used on the interface between a controller and network devices, and between network orchestrators and controllers. There may also be a hierarchy of such components with super controllers, domain controllers, and device controllers all exchanging information and instructions using YANG modules.

There has been interest in using YANG to define and document data models that describe services in a portable way that is independent of which network operator uses the model. For example, the Layer Three Virtual Private Network Service Model (L3SM) [RFC8049]. Such models may be used in manual and even paper-driven service request processes with a gradual transition to IT-based mechanisms. Ultimately they could be used in online, software-driven dynamic systems, and may be used as part of an SDN system.

This document explains the scope and purpose of service models within the IETF (and limited to within the scope of the IETF) and describes how a service model can be used by a network operator. Equally, this document clarifies what a service model is not, and dispels some common misconceptions.

The document also shows where a service model might fit into an SDN architecture, but it is important to note that a service model does not require or preclude the use of SDN. Note that service models do not make any assumption of how a service is actually engineered and delivered to a customer; details of how network protocols and devices are engineered to deliver a service are captured in other modules that are not exposed through the interface between the customer and the provider.

In summary, a service model is a formal representation of the data elements that describe a network service as that service is described to or requested by a customer of a network operator. Details included in the service model include a description of the service as experienced by the customer, but not features of how that service is delivered or realized by the service provider.

Other work on classifying YANG modules has been done in [RFC8199]. That document provides an important reference for this document, and also uses the term "service module". Section 6.1 in this document provides a comparison between these two uses of the same terminology.

2. Terms and Concepts

Readers should familiarize themselves with the description and classification of YANG modules provided in [RFC8199].

The following terms are used in this document:

Network Operator: This term is used to refer to the company that owns and operates one or more networks that provide Internet connectivity services and/or other services.

Customer: This term refers to someone who purchases a service (including connectivity) from a network operator. In the context of this document, a customer is usually a company that runs their own network or computing platforms and wishes to connect to the Internet or between sites. Such a customer may operate an enterprise network or a data center. Sometimes this term may also be used to refer to the individual in such a company who contracts to buy services from a network operator. A customer as described here is a separate commercial operation from the network operator, but some companies may operate with internal customers so that, for example, an IP/MPLS packet network may be the customer of an optical transport network.

Service: A network operator delivers one or more services to a customer. A service in the context of this document (sometimes called a Network Service) is some form of connectivity between customer sites and the Internet, or between customer sites across the network operator's network and across the Internet. However, a distinction should be drawn between the parameters that describe a service as included in a customer service model (see the definition of this term, below) and a Service Level Agreement (SLA) as discussed in Section 5 and Section 7.2.

A service may be limited to simple connectivity (such as IP-based Internet access), may be a tunnel (such as a virtual circuit), or may involve more complex connectivity (such as in a multi-site virtual private network). Services may be further enhanced by additional functions providing security, load-balancing, accounting, and so forth. Additionally, services usually include guarantees of quality, throughput, and fault reporting.

This document makes a distinction between a service as delivered to a customer (that is, the service as discussed on the interface between a customer and the network operator) and the service as realized within the network (as described in [RFC8199]). This distinction is discussed further in Section 6.

Readers may also refer to [RFC7297] for an example of how an IP connectivity service may be characterized.

Data Model: The concepts of information models and data models are described in [RFC3444]. That document defines a data model by contrasting it with the definition of an information model as follows:

The main purpose of an information model is to model managed objects at a conceptual level, independent of any specific implementations or protocols used to transport the data. The degree of specificity (or detail) of the abstractions defined in the information model depends on the modeling needs of its designers. In order to make the overall design as clear as possible, an information model should hide all protocol and implementation details. Another important characteristic of an information model is that it defines relationships between managed objects.

Data models, conversely, are defined at a lower level of abstraction and include many details. They are intended for implementors and include protocol-specific constructs.

As mentioned in Section 1, this document also uses the terms "data model" and "YANG module" as defined in [RFC7950].

Service Model: A service model is a specific type of data model. It describes a service and the parameters of the service in a portable way that can be used uniformly independent of the equipment and operating environment. The service model may be divided into two categories:

Customer Service Model: A customer service model is used to describe a service as offered or delivered to a customer by a network operator as shown in Figure 1. It can be used by a human (via a user interface such as a GUI, web form, or CLI) or by software to configure or request a service, and may equally be consumed by a human (such as via an order fulfillment system) or by a software component. Such models are sometimes referred to simply as "service models" [RFC8049]. A customer

service model is expressed in a YANG module as a core set of parameters that are common across network operators: additional features that are specific to the offerings of individual network operators would be defined in extensions or augmentations of the module. Except where specific technology details (such as encapsulations, or mechanisms applied on access links) are directly pertinent to the customer, customer service models are technology agnostic so that the customer does not have influence over or knowledge of how the network operator engineers the service.

An example of where such details are relevant to the customer is when they describe the behavior or interactions on the interface between the equipment at the customer site (often referred to as the Customer Edge or CE equipment) and the equipment at the network operator's site (usually referred to as the Provider Edge or PE equipment).

Service Delivery Model: A service delivery model is used by a network operator to define and manage how a service is engineered in the network. It can be used by a human operator (such as via a management station) or by a software tool to instruct network components. The YANG modules that encode such models are sometimes referred to as "network service YANG modules" [RFC8199] and are consumed by "external systems" such as Operations Support System (OSS). A service delivery module is expressed as a core set of parameters that are common across a network type and technology: additional features that are specific to the configuration of individual vendor equipment or proprietary protocols would be defined in extensions or augmentations of the module. Service delivery modules include technology-specific modules.

The distinction between a customer service model and a service delivery model needs to be clarified. The modules that encode a customer service model are not used to directly configure network devices, protocols, or functions: it is not something that is sent to network devices (i.e., routers or switches) for processing. Equally, the modules that encode a customer service model do not describes how a network operator realizes and delivers the service described by the module. This distinction is discussed further in later sections.

3. Using Service Models

As already indicated, customer service models are used on the interface between customers and network operators. This is shown simply in Figure 1.

The language in which a customer service model is described is a choice for whoever specifies the model. The IETF uses the YANG data modeling language defined in [RFC6020].

The encoding and communication protocol used to exchange a customer service model between customer and network operator are deployment- and implementation-specific. The IETF has standardized the NETCONF protocol [RFC6241] and the RESTCONF protocol [RFC8040] for interactions "on the wire" between software components with data encoded in XML or JSON. However, co-located software components might use an internal API, while systems with more direct human interactions might use web pages or even paper forms. Where direct human interaction comes into play, interface interactions may be realized via business practices and that may introduce some margin of error, thus raising the priority for automated, deterministic interfaces.

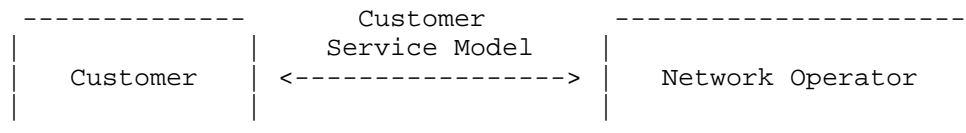


Figure 1: The Customer Service Models used on the Interface between Customers and Network Operators

How a network operator processes a customer's service request described with a customer service model depends on the commercial and operational tools, processes, and policies used by the network operator. These may vary considerably from one network operator to another.

However, the intent is that the network operator maps the service request into configuration and operational parameters that control one or more networks to deliver the requested services. That means that the network operator (or software run by the network operator) takes the information in the customer service model and determines how to deliver the service by enabling and configuring network protocols and devices. They may achieve this by constructing service delivery models and passing them to network orchestrators or controllers. The use of standard customer service models eases service delivery by means of automation.

3.1. Practical Considerations

The practicality of customer service models has been repeatedly debated. It has been suggested that network operators have radically different business modes and widely diverse commercial offerings making a common customer service model is impractical. However, the L3SM [RFC8049] results from the consensus of multiple individuals working at network operators and offers a common core of service options that can be augmented according to the needs of individual network operators.

It has also been suggested that there should be a single, base customer service module, and that details of individual services should be offered as extensions or augmentations of this. It is quite possible that a number of service parameters (such as the identity and postal address of a customer) will be common and it would be a mistake to define them multiple times, once in each customer service model. However, the distinction between a 'module' and a 'model' should be considered at this point: modules are how the data for models is logically broken out and documented especially for re-use in multiple models.

4. Service Models in an SDN Context

In a Software Defined Networking (SDN) system, the management of network resources and protocols is performed by software systems that determine how best to utilize the network. Figure 2 shows a sample architectural view of an SDN system where network elements are programmed by a component called an "SDN controller" (or "controller" for short), and where controllers are instructed by an orchestrator that has a wider view of the whole of, or part of, a network. The internal organization of an SDN control plane is deployment-specific.

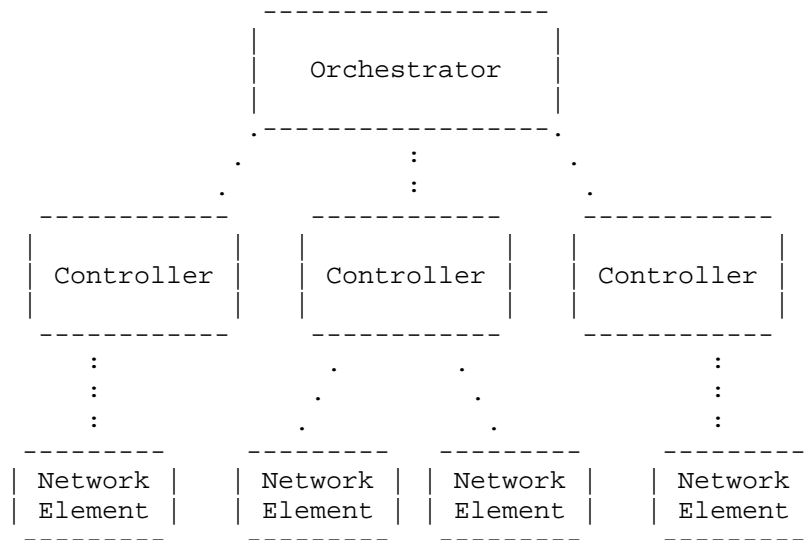


Figure 2: A Sample SDN Architecture

A customer's service request is (or should be) technology-agnostic. That is, a customer is unaware of the technology that the network operator has available to deliver the service and so the customer does not make requests specific to the underlying technology, but is limited to making requests specific to the service that is to be delivered. The orchestrator must map the service request to its view, and this mapping may include a choice of which networks and technologies to use depending on which service features have been requested.

One implementation option to achieve this mapping is to split the orchestration function between a "Service Orchestrator" and a "Network Orchestrator" as shown in Figure 3.

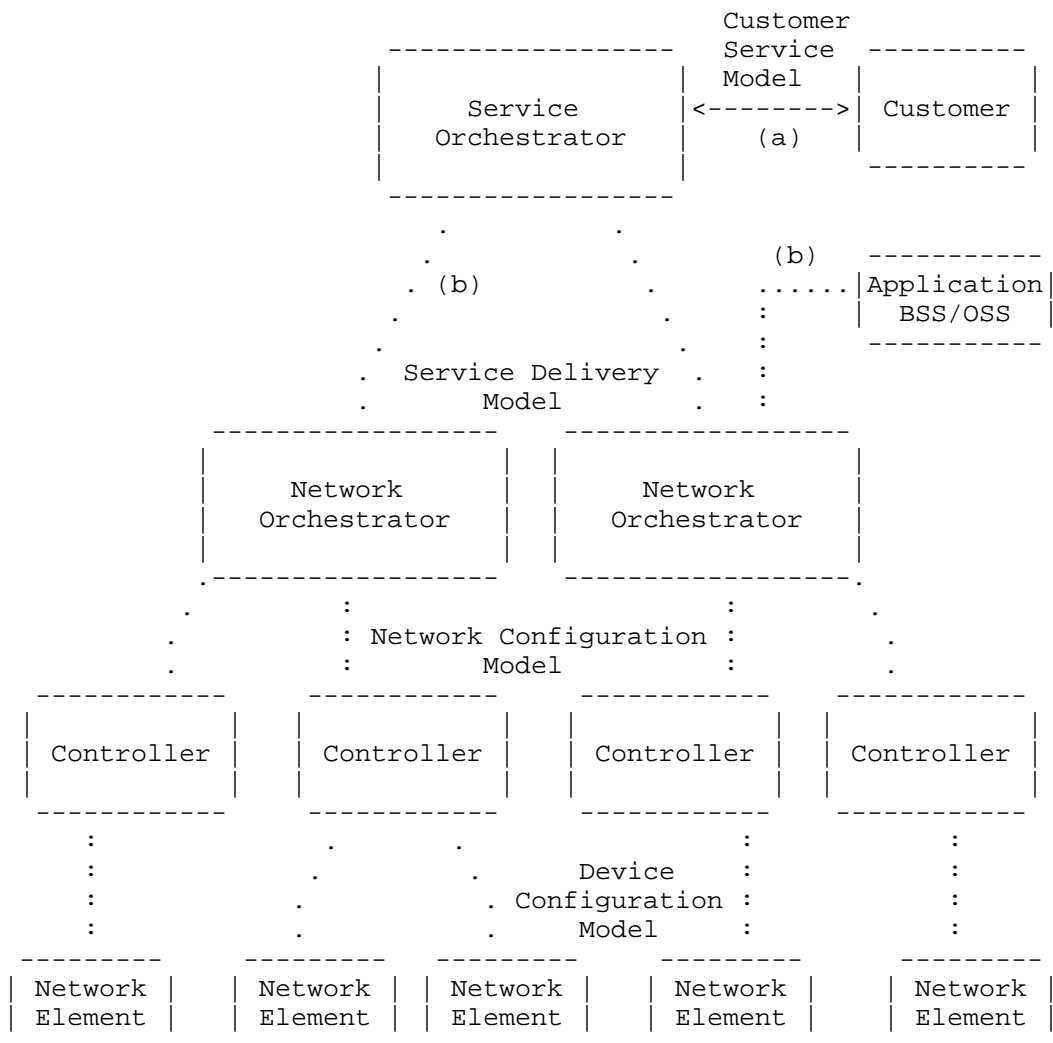


Figure 3: An Example SDN Architecture with a Service Orchestrator

Figure 3 also shows where different data models might be applied within the architecture. The Device Configuration Models are used by a Controller to set parameters on an individual network element. The Network Configuration Models are used by a Network Orchestrator to instruct Controllers (that may each be responsible for multiple network elements) how to configure parts of a network.

The split between control components that exposes a "service interface" that is present in many figures showing extended SDN architectures:

- o Figure 1 of [RFC7426] shows a separation of the "Application Plane", the "Network Services Abstraction Layer (NSAL)", and the "Control Plane". It marks the "Service Interface" as situated between the NSAL and the control plane.
- o Figure 1 of [RFC7491] shows an interface between an "Application Service Coordinator" and an "Application-Based Network Operations Controller".
- o Figure 1 of [RFC8199] shows an interface from an OSS or a Business Support System (BSS) that is expressed in "Network Service YANG Modules".

This can all lead to some confusion around the definition of a "service interface" and a "service model". Some previous literature considers the interface northbound of the Network Orchestrator (labeled "(b)" in Figure 3) to be a "service interface" used by an application, but the service described at this interface is network-centric and is aware of many features such as topology, technology, and operator policy. Thus, we make a distinction between this type of service interface and the more abstract service interface (labeled "(a)" in Figure 3) where the service is described by a service model and the interaction is between customer and network operator. Further discussion of this point is provided in Section 5.

5. Possible Causes of Confusion

In discussing service models, there are several possible causes of confusion:

- o The services we are discussing are connectivity services provided by network operators to customers achieved by manipulating the network resources of the operator's network. This is a completely different thing to "Foo as a Service" (for example, Storage as a Service (SaaS)) where a service provider offers reachability to a value-added service that is provided at some location in the network using other resources (compute, storage, ...) that are not part of the network itself. The confusion arises not only because of the use of the word "service" in both cases, but also because network operators may offer both types of service to their customers.
- o Network operation is normally out of scope in the discussion of services between a network operator and a customer. That means

that the customer service model does not reveal to the customer anything about how the network operator delivers the service, nor does the model expose details of technology or network resources used to provide the service (all of these details could, in any case, be considered security vulnerabilities. For example, in the simple case of point-to-point virtual link connectivity provided by a network tunnel (such as an MPLS pseudowire) the network operator does not expose the path through the network that the tunnel follows. Of course, this does not preclude the network operator from taking guidance from the customer (such as to avoid routing traffic through a particular country) or from disclosing specific details (such as might be revealed by a route trace), but these are not standard features of the service as described in the customer service model.

- o The network operator may use further data models (service delivery models) that help to describe how the service is realized in the network. These models might be used on the interface between the Service Orchestrator and the Network Orchestrator as shown in Figure 3 and might include many of the pieces of information from the customer service model alongside protocol parameters and device configuration information. [RFC8199] also terms these data models as "service models" and encode them as "Network Service YANG Modules" and a comparison is provided in Section 6.1. It is important that the Service Orchestrator should be able to map from a customer service model to these service delivery models, but they are not the same things.
- o Commercial terms (such as cost per byte, per minute, scoped by quality and type of service, and related to payment terms) are generally not a good subject for standardization. It is possible that some network operators will enhance standard customer service models to include commercial information, but the way this is done is likely to vary widely between network operators. Thus, this feature is out of scope for standardized customer service models.
- o Service Level Agreements (SLAs) have a high degree of overlap with the definition of services present in customer service models. Requests for specific bandwidth, for example, might be present in a customer service model, and agreement to deliver a service is a commitment to the description of the service in the customer service model. However, SLAs typically include a number of fine-grained details about how services are allowed to vary, by how much, and how often. SLAs are also linked to commercial terms with penalties and so forth, and thus are also not good topics for standardization. As with commercial terms, it is expected that some network operators will enhance standard customer service models to include SLA parameters either using their own work or

depending on material from standards bodies that specialize in this topic, but this feature is out of scope for the IETF's customer service models.

If a network operator chooses to express an SLA using a data model, that model might be referenced as an extension or an augmentation of the customer service model.

6. Comparison With Other Work

Other work has classified YANG modules, produced parallel architectures, and developed a range of YANG modules. This section briefly examines that other work and shows how it fits with the description of service models introduced in this document.

6.1. Comparison With Network Service Models

As previously noted, [RFC8199] provides a classification of YANG modules. It introduces the term "Network Service YANG Module" to identify the type of module used to "describe the configuration, state data, operations and notifications of abstract representations of services implemented on one or multiple network elements." These modules are used to construct the service delivery models as described in this document; that is, they are the modules used on the interface between the Service Orchestrator or OSS/BSS and the Network Orchestrator as shown in Figure 3.

Figure 1 of [RFC8199] can be modified to make this more clear and to add an additional example of a Network Service YANG module as shown in Figure 4. As can be seen, the highest classification of modules collects those that are used to deliver operations support and business support. These might be consumed by an Operations Support System (OSS) or a Business Support System (BSS), and a Service Orchestrator may form part of an OSS/BSS or may be a separate component. This highest layer in the figure is divided into the Customer Service Modules that are used to describe services to a customer as discussed in this document, and other modules that describe further OSS/BSS function such as billing and SLAs.

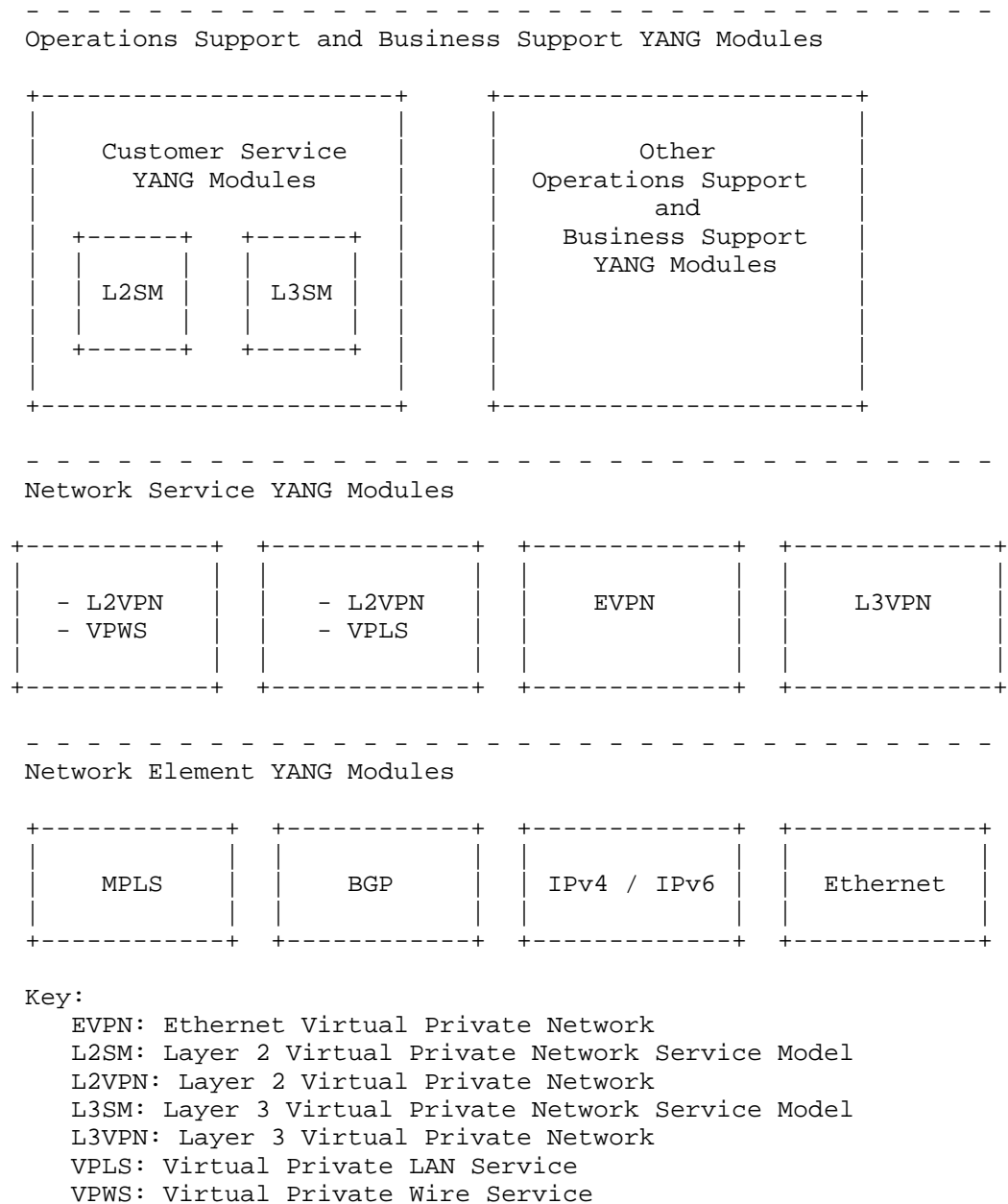


Figure 4: YANG Module Abstraction Layers Showing Customer Service Modules

6.2. Service Delivery and Network Element Model Work

A number of IETF working groups are developing YANG modules related to services. These models focus on how the network operator configures the network through protocols and devices to deliver a service. Some of these models are classified as network service delivery models (also called service delivery models or network configuration models depending on the level at which they are pitched), while others have details that are related to specific element configuration and so are classed as network element models (also called device models).

A sample set of these models is listed here:

- o [I-D.dhjain-bess-bgp-l3vpn-yang] defines a YANG module that can be used to configure and manage BGP L3VPNs.
- o [I-D.ietf-bess-l2vpn-yang] documents a data model that it is expected will be used by the management tools run by the network operators in order to manage and monitor the network resources that they use to deliver L2VPN services.
- o [I-D.ietf-bess-evpn-yang] defines YANG modules for delivering an Ethernet VPN service.

6.3. Customer Service Model Work

Several initiatives within the IETF are developing customer service models. The L3SM presents the L3VPN service as described by a network operator to a customer. Figure 5, which is reproduced from [RFC8049], shows that the L3SM is a customer service model as described in this document.

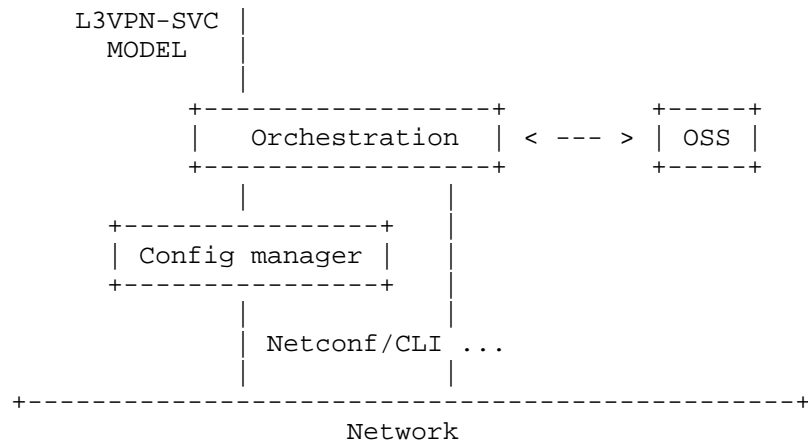


Figure 5: The L3SM Service Architecture

A Layer Two VPN service model (L2SM) is defined in [I-D.ietf-l2sm-l2vpn-service-model]. That model's usage is described as in Figure 6 which is a reproduction of Figure 5 from that document. As can be seen, the L2SM is a customer service model as described in this document.

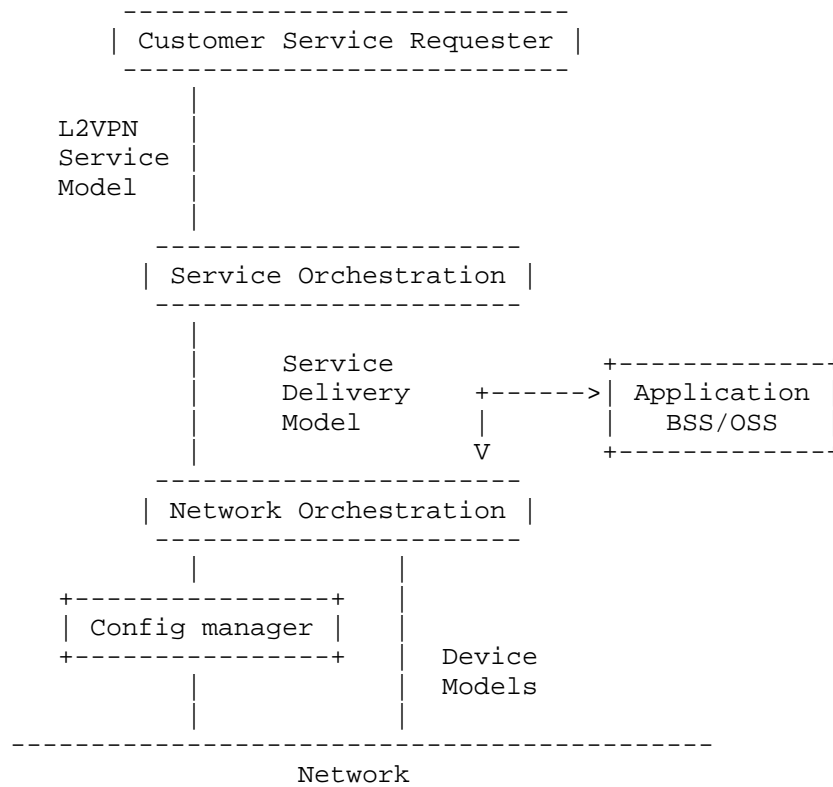


Figure 6: The L2SM Service Architecture

6.4. The MEF Architecture

The MEF Forum has developed an architecture for network management and operation. It is documented as the Lifecycle Service Orchestration (LSO) Reference Architecture and illustrated in Figure 2 of [MEF-55].

The work of the MEF Forum embraces all aspects of Lifecycle Service Orchestration including billing, SLAs, order management, and life-cycle management. The IETF's work on service models is typically smaller offering a simple, self-contained service YANG module. This does not invalidate either approach: it only observes that they are different approaches.

7. Further Concepts

This section introduces a few further, more advanced concepts

7.1. Technology Agnostic

Service models should generally be technology agnostic. That is to say, the customer should not care how the service is provided so long as the service is delivered.

However, some technologies reach the customer site and make a difference to the type of service delivered. Such features do need to be described in the service model.

Two examples are:

- o The data passed between customer equipment and network operator equipment will be encapsulated in a specific way, and that data plane type forms part of the service.
- o Protocols that are run between customer equipment and network operator equipment (for example, Operations, Administration, and Maintenance protocols, protocols for discovery, or protocols for exchanging routing information) need to be selected and configured as part of the service description.

7.2. Relationship to Policy

Policy appears as a crucial function in many places during network orchestration. A Service Orchestrator will, for example, apply the network operator's policies to determine how to provide a service for a particular customer (possibly considering commercial terms). However, the policies within a service model are limited to those over which a customer has direct influence and that are acted on by the network operator.

The policies that express desired behavior of services on occurrence of specific events are close to SLA definitions: they should only be included in the base service model where they are common to all network operators' offerings. Policies that describe who at a customer may request or modify services (that is, authorization) are close to commercial terms: they, too, should only be included in the base service model where they are common to all network operators' offerings.

As with commercial terms and SLAs discussed in Section 5, it is expected that some network operators will enhance standard customer service models to include policy parameters either using their own

work or depending on specific policy models built in the IETF or other standards bodies.

Nevertheless, policy is so important that all service models should be designed to be easily extensible to allow policy components to be added and associated with services as needed.

7.3. Operator-Specific Features

When work on the L3SM was started, there was some doubt as to whether network operators would be able to agree on a common description of the services that they offer to their customers because, in a competitive environment, each markets the services in a different way with different additional features. However, the working group was able to agree on a core set of features that multiple network operators were willing to consider as "common". They also understood that, should an individual network operator want to describe additional features (operator-specific features), they could do so by extending or augmenting the L3SM model.

Thus, when a basic description of a core service is agreed and documented in a service model, it is important that that model should be easily extended or augmented by each network operator so that the standardized model can be used in a common way and only the operator-specific features varied from one environment to another.

7.4. Supporting Multiple Services

Network operators will, in general, offer many different services to their customers. Each would normally be the subject of a separate service model.

Whether each service model is handled by a specialized Service Orchestrator able to provide tuned behavior for a specific service or whether all service models are handled by a single Service Orchestrator is an implementation and deployment choice.

It is expected that, over time, certain elements of the service models will be seen to repeat in each model. An example of such an element is the postal address of the customer.

It is anticipated that, while access to such information from each service model is important, the data will be described in its own module and may form part of the service model either by inclusion or by index.

8. Security Considerations

The interface between customer and service provider is a commercial interface and needs to be subject to appropriate confidentiality. Additionally, knowledge of what services are provided to a customer or delivered by a network operator may supply information that can be used in a variety of security attacks. The service model itself will expose security-related parameters for the specific service where the related function is available to the customer.

Clearly, the ability to modify information exchanges between customer and network operator may result in bogus requests, unwarranted billing, and false expectations. Furthermore, in an automated system, modifications to service requests or the injection of bogus requests may lead to attacks on the network and delivery of customer traffic to the wrong place.

Therefore it is important that the protocol interface used to exchange service request information between customer and network operator is subject to authorization, authentication, and encryption. Clearly, the level of abstraction provided by a service model protects the operator from unwarranted visibility into their network, and the fact that it is entirely up to the operator how they deliver the service provides additional protection.

Equally, all external interfaces, such as any of those between the functional components in Figure 3 needs to be correctly secured. This document discusses modeling the information, not how it is exchanged.

9. Manageability Considerations

This whole document discusses issues related to network management and control.

It is important to observe that automated service provisioning resulting from use of a customer service model may result in rapid and significant changes in traffic load within a network and that that might have an effect on other services carried in a network.

It is expected, therefore, that a Service Orchestration component has awareness of other service commitments, that the Network Orchestration component will not commit network resources to fulfill a service unless doing so is appropriate, and that a feedback loop will be provided to report on degradation of the network that will impact the service.

The operational state of a service does not form part of a customer service model. However, it is likely that a network operator may want to report some state information about various components of the service, and that could be achieved through extensions to the core service model just as SLA extensions could be made as described in Section 5.

10. IANA Considerations

This document makes no requests for IANA action.

11. Acknowledgements

Thanks to Daniel King, Xian Zhang, Michael Scharf, Med Boucadair, Luis Miguel Contreras Murillo, Joe Salowey, Benoit Claise, Robert Sparks, Tom Petch, David Sinicrope, and Deborah Brungard for useful review and comments.

Thanks to Dean Bogdanovic, Tianran Zhou, and Carl Moberg for their help coordinating with [RFC8199].

Many thanks to Jerry Bonner for spotting a tiny, one-word, but critical typo.

12. References

12.1. Normative References

- [RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003, <<https://www.rfc-editor.org/info/rfc3444>>.
- [RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <<https://www.rfc-editor.org/info/rfc8199>>.

12.2. Informative References

- [I-D.dhjain-bess-bgp-l3vpn-yang] Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S., Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model for BGP/MPLS L3 VPNs", draft-dhjain-bess-bgp-l3vpn-yang-02 (work in progress), August 2016.

- [I-D.ietf-bess-evpn-yang]
Brissette, P., Sajassi, A., Shah, H., Li, Z.,
Tiruveedhula, K., Hussain, I., and J. Rabadan, "Yang Data
Model for EVPN", draft-ietf-bess-evpn-yang-02 (work in
progress), March 2017.
- [I-D.ietf-bess-l2vpn-yang]
Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B.,
and K. Tiruveedhula, "YANG Data Model for MPLS-based
L2VPN", draft-ietf-bess-l2vpn-yang-07 (work in progress),
October 2017.
- [I-D.ietf-l2sm-l2vpn-service-model]
Wen, B., Fioccola, G., Xie, C., and L. Jalil, "A YANG Data
Model for L2VPN Service Delivery", draft-ietf-l2sm-l2vpn-
service-model-03 (work in progress), September 2017.
- [MEF-55] MEF Forum, "Service Operations Specification MEF 55 :
Lifecycle Service Orchestration (LSO) Reference
Architecture and Framework", March 2016.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for
the Network Configuration Protocol (NETCONF)", RFC 6020,
DOI 10.17487/RFC6020, October 2010,
<<https://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,
<<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7149] Boucadair, M. and C. Jacquenet, "Software-Defined
Networking: A Perspective from within a Service Provider
Environment", RFC 7149, DOI 10.17487/RFC7149, March 2014,
<<https://www.rfc-editor.org/info/rfc7149>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface
Management", RFC 7223, DOI 10.17487/RFC7223, May 2014,
<<https://www.rfc-editor.org/info/rfc7223>>.
- [RFC7297] Boucadair, M., Jacquenet, C., and N. Wang, "IP
Connectivity Provisioning Profile (CPP)", RFC 7297,
DOI 10.17487/RFC7297, July 2014,
<<https://www.rfc-editor.org/info/rfc7297>>.
- [RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for
SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407,
December 2014, <<https://www.rfc-editor.org/info/rfc7407>>.

- [RFC7426] Haleplidis, E., Ed., Pentikousis, K., Ed., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, DOI 10.17487/RFC7426, January 2015, <<https://www.rfc-editor.org/info/rfc7426>>.
- [RFC7491] King, D. and A. Farrel, "A PCE-Based Architecture for Application-Based Network Operations", RFC 7491, DOI 10.17487/RFC7491, March 2015, <<https://www.rfc-editor.org/info/rfc7491>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8049] Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8049, DOI 10.17487/RFC8049, February 2017, <<https://www.rfc-editor.org/info/rfc8049>>.

Authors' Addresses

Qin Wu
Huawei Technologies

Email: bill.wu@huawei.com

Will Liu
Huawei Technologies

Email: liushucheng@huawei.com

Adrian Farrel
Juniper Networks

Email: afarrel@juniper.net

opsawg
Internet-Draft
Intended status: Standards Track
Expires: January 1, 2018

Z. Li
China Mobile
P. Aitken
Brocade Communications Systems, Inc.
June 30, 2017

Extended Length Message Support for IP Flow Information Export (IPFIX)
draft-li-opsawg-ipfix-extended-message-00

Abstract

The specification of the IP Flow Information Export (IPFIX) Protocol [RFC7011] defines an IPFIX Message length of 16 bits. As new Information Elements (IEs) are introduced in IPFIX to export long information, such as the BGP community information [I-D.ietf-opsawg-ipfix-bgp-community], an IPFIX Message no longer has sufficient space to fit all the information of a specific flow. This document updates the IPFIX specification by extending the IPFIX Message length from 16 bits to 32 bits. For backwards compatibility, a new version of IPFIX (i.e., 11) is introduced to support the 32-bit Message length.

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 <http://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 January 1, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	2
3. IPFIX Extended Length Message	3
3.1. IPFIX Extended Length Message Header	3
3.2. IPFIX Extended Length Set Header	3
3.3. IPFIX Extended Variable-Length IE	4
4. Transport Protocol Considerations	4
5. Security Considerations	5
6. IANA Considerations	5
7. Acknowledgements	5
8. References	5
8.1. Normative References	5
8.2. Informative References	5
Authors' Addresses	6

1. Introduction

The IP Flow Information Export (IPFIX) Protocol [RFC7011] provides network administrators with traffic flow information using the Information Elements (IEs) defined in IANA's IPFIX registry [IANA-IPFIX]. [RFC7011] specifies an IPFIX Message length of 16 bits. As new IEs are introduced in IPFIX to export long information, such as the BGP community information [I-D.ietf-opsawg-ipfix-bgp-community], one IPFIX Message no longer has sufficient space to fit all the information of a specific flow. The maximum IPFIX message size needs to be extended beyond 65535 octets. This document updates the IPFIX specification by extending the IPFIX Message length from 16 bits to 32 bits, which means the maximum IPFIX message size is 4 giga bytes. For backwards compatibility, a new version of IPFIX (i.e., 11) is introduced to support the 32-bit Message length.

2. Terminology

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 [RFC2119].

The goal of this document is to allow the export of IPFIX messages up to 4294967295 (4 giga) octets to satisfy the two situations that may result in the IPFIX message beyond 65535 octets. One is the IPFIX message consisting of many small fields where each field is 8-bits or 16-bits long. The other one is the IPFIX message consisting of at least one IE which is longer than 65535 octets, such as the IEs defined in [I-D.ietf-opsawg-ipfix-bgp-community].

To satisfy the two situations, this document extends the following length fields to 32 bits: the Length field in the IPFIX Message Header, the Length field in the Set Header and the Length field in the variable-length IE.

The IPFIX Message Header with a 32 bits length field is called the IPFIX Extended Length Message Header, whose format is shown in Figure 1. The version number MUST be 11. The length field is 32-bits long. The meanings and other specifications of the fields in the Extended Length Message Header are in accordance with [RFC7011]. Please refer to Figure F in [RFC7011].

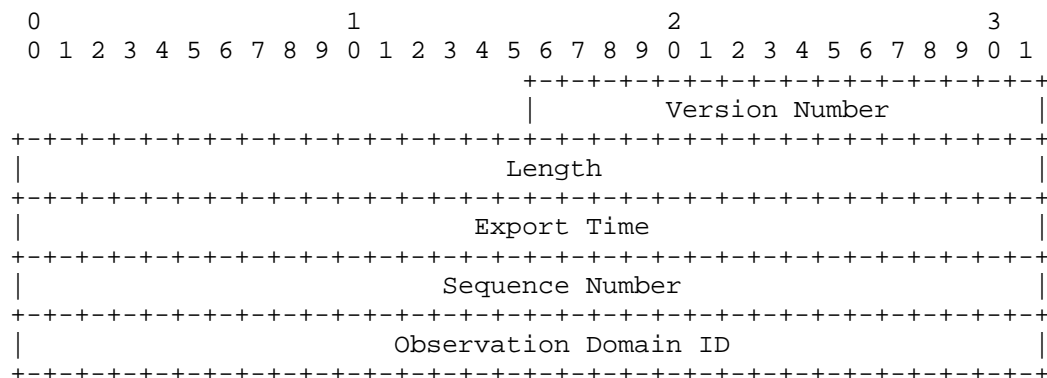


Figure 1: IPFIX Extended Length Message Header Format

The IPFIX Set Header with a 32 bits length field is called the IPFIX Extended Length Set Header, whose format is shown in Figure 2. The meanings and other specifications of the fields in the Extended Length Set Header are in accordance with [RFC7011]. Please refer to Figure 1 in [RFC7011].

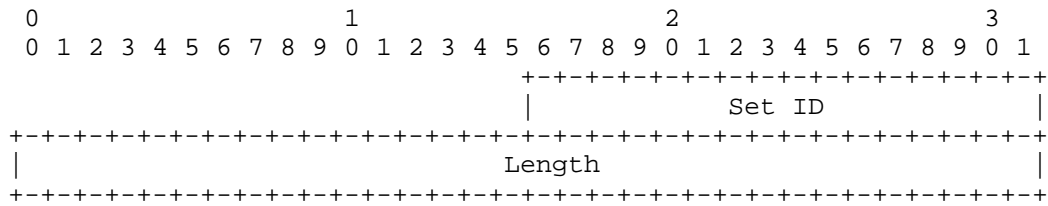


Figure 2: Extended Length Set Header Format

3.3. IPFIX Extended Variable-Length IE

The Extended Variable-Length IE allows export of variable-length IEs with size greater than or equal to 65535 octets, the length field of which is extended to 32 bits as shown in Figure 3. The meanings and other specifications of the fields in the Extended Variable-Length IE are in accordance with [RFC7011]. Please refer to Figure S in [RFC7011].

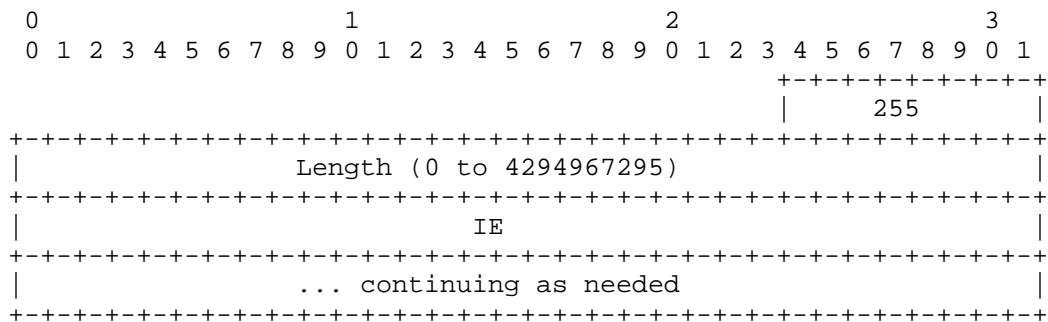


Figure 3: Extended Variable-Length IE

4. Transport Protocol Considerations

As per section 10 of RFC7011, the IPFIX Protocol is transport protocol independent. SCTP [RFC4960] using the Partially Reliable SCTP (PR-SCTP) extension as specified in [RFC3758] MUST be implemented by all compliant implementations. UDP [RFC768] MAY also be implemented by compliant implementations. TCP [RFC793] MAY also be implemented by compliant implementations. The Collecting Process of a compliant implementation supporting IPFIX Extended Length Message MUST be able to handle IPFIX Message lengths of up to 4294967295 octets.

5. Security Considerations

This extension to IPFIX does not change IPFIX's underlying security issues, please refer to RFC7011.

6. IANA Considerations

A new IPFIX Version Number value of 11 is reserved in IANA's IPFIX registry [IANA-IPFIX] for the IPFIX Extended Length Message specified in this document.

7. Acknowledgements

The authors would like to thank Ignas Bagdonas and Stewart Bryant for their constructive discussion.

8. References

8.1. Normative References

[IANA-IPFIX]

IANA, "IPFIX Information Elements registry",
<<http://www.iana.org/assignments/ipfix/ipfix.xml>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

[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>>.

8.2. Informative References

[I-D.ietf-opsawg-ipfix-bgp-community]

Li, Z., Gu, R., and J. Dong, "Export BGP community information in IP Flow Information Export (IPFIX)", draft-ietf-opsawg-ipfix-bgp-community-02 (work in progress), June 2017.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, DOI 10.17487/RFC3758, May 2004, <<http://www.rfc-editor.org/info/rfc3758>>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol",
RFC 4960, DOI 10.17487/RFC4960, September 2007,
<<http://www.rfc-editor.org/info/rfc4960>>.

Authors' Addresses

Zhenqiang Li
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing 100053
China

Email: li_zhenqiang@hotmail.com

Paul Aitken
Brocade Communications Systems, Inc.
19a Canning Street, Level 3
Edinburgh, Scotland EH3 8EG
United Kingdom

Email: paitken@brocade.com

OPSAWG Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2017

N. Sambo
M. Dallaglio
P. Castoldi
Scuola Superiore Sant'Anna
G. Fioccola
A. Di Giglio
Telecom Italia
F. Cugini
CNIT
G. Bernini
P. Giardina
Nextworks
June 27, 2017

Extending YANG for events, actions, and finite state machine
draft-sambo-opsawg-ccamp-supra-ext-yang-fsm-00

Abstract

Network operators and service providers are facing the challenge of deployment of systems from different vendors while looking for a trade-off among transmission performance, network device reuse, and capital expenditure without the need of being tied to single vendor equipment. The deployment and operation of more dynamic and programmable transport optical network infrastructures can be driven by adopting model-driven and software-defined control and management paradigms. In this context, YANG enables to compile a set of consistent vendor-neutral data models for optical networks and components based on actual operational needs emerging from heterogeneous use cases. This document extends YANG from data to functional modeling in order to describe events, operations, and finite state machine of YANG-defined network elements. The proposed models can be applied in the context of optical networks to pre-instruct data plane devices (e.g., an optical transponder) on the actions to be performed (e.g., code adaptation) in case some events, such as physical layer degradations, occur.

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 <http://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 December 29, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Conventions used in this document	3
3. Terminology	4
4. Example of application	4
5. Extending YANG for events and reactions	7
6. Extending YANG for finite state machine (FSM)	9
7. Implementation for the considered use case of application . .	9
8. Appendix	10
8.1. YANG model for events and actions - Tree	10
8.2. YANG model for FSM - Tree	11
8.3. YANG model for events and actions - Code	12
8.4. YANG model for FSM - Code	16
8.5. Example of values for the YANG model	19
9. Acknowledgements	19
10. Security Considerations	20
11. References	20
11.1. Normative References	20
11.2. Informative References	20
Authors' Addresses	21

1. Introduction

Networks are evolving toward more programmability, flexibility, and multi-vendor interoperability. Multi-vendor interoperability can be applied in the context of nodes, i.e. a node composed of components provided by different vendors (named white box) is assembled under the same control system. This way, operators can optimize costs and network performance without the need of being tied to single vendor equipment. NETCONF protocol RFC6241 [RFC6241] based on YANG data modeling language RFC6020 [RFC6020] is emerging as a candidate Software Defined Networking (SDN) enabled protocol. First, NETCONF supports both control and management functionalities, thus permits high programmability. Then, YANG enables data modeling in a vendor-neutral way. Some recent works have provided YANG models to describe attributes of links (e.g., identification), nodes (e.g., connectivity matrix), media channels, and transponders (e.g., supported forward error correction - FEC) of networks

([I-D.ietf-i2rs-yang-network-topo] [I-D.vergara-ccamp-flexigrid-yang] [I-D.zhang-ccamp-ll-topo-yang]), also including optical technologies. Such draft mainly refers to elastic optical networks (EONs), i.e. optical networks based on flexible grid where circuits with different bandwidth requirements are switched. EONs are expected to employ flexible transponders, i.e. transponders supporting multiple bit rates, multiple modulation formats, and multiple codes. Such transponders permits the (re-) configuration of the bit rate value based on traffic requirements, as well as the configuration of the modulation format and code based on the physical characteristics of a path (e.g., quadrature phase shift keying is more robust than 16 quadrature amplitude modulation). This document extends YANG from data to functional modeling in order to describe events, operations, and finite state machine of YANG-defined network elements. Such models can be applied to a case of transponder reconfiguration in EONs. In particular, the model enables a centralized remote network controller (managed by a network operator) to instruct a transponder controller about the actions to perform when certain events (e.g., failures) occur. The actions to be taken and the events can be re-programmed on the device.

2. Conventions used in this document

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 RFC2119 [RFC2119].

3. Terminology

ABNO: Application-Based Network Operations

BER: Bit Error Rate

EON: Elastic Optical Network

FEC: Forward Error Correction

FSM: Finite State Machine

NETCONF: Network Configuration Protocol

OAM: Operation Administration and Maintenance

SDN: Software Defined Network

YANG: Yet Another Network Generator

4. Example of application

Flexible transponders enable several settings of transmission parameters' configuration, through the support of multiple modulation formats and forward error correction (FEC) schemes. This way, transmission parameters can be (re-)configured based on the physical layer conditions. The YANG model presented in this draft enables to pre-program reconfiguration settings of data plane devices in case of failures or physical layer degradations. In particular, soft failures are assumed. Soft failures imply transmission performance degradation, in turns a bit error rate (BER) increase, e.g. due to the ageing of some network devices. Without loosing generality, the ABNO architecture is assumed for the control and management of EONs (RFC7491 [RFC7491]). Considering the state of the art, when pre-FEC BER passes above a predefined threshold, it is expected that an alarm is sent to the OAM Handler, which communicates with the ABNO controller that may trigger an SDN controller (that could be the Provisioning Manager of ABNO RFC7491 [RFC7491]) for computing new transmission parameters. The involved ABNO modules are shown in the simplified ABNO architecture of Fig. 1. Then, transponders are reconfigured. When alarms related to several connections impacted by the soft failure are generated, this procedure may be particularly time consuming. The related workflow for transponder reconfiguration is shown in Fig. 2. The proposed model enables an SDN controller to instruct the transponder about reconfiguration of new transmission parameters values if a soft failure occurs. This can be done before the failure occurs (e.g., during the connection instantiation phase or during the connection service), so that data plane devices can

promptly reconfigure themselves without querying the SDN controller to trigger an on-demand recovery. This is expected to speed up the recovery process from soft failures. The related flow chart is shown in Fig. 3.

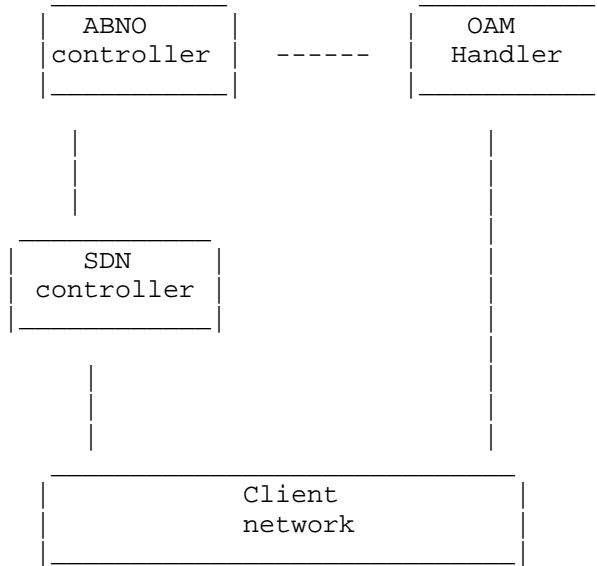


Figure 1: Assumed ABNO functional modules

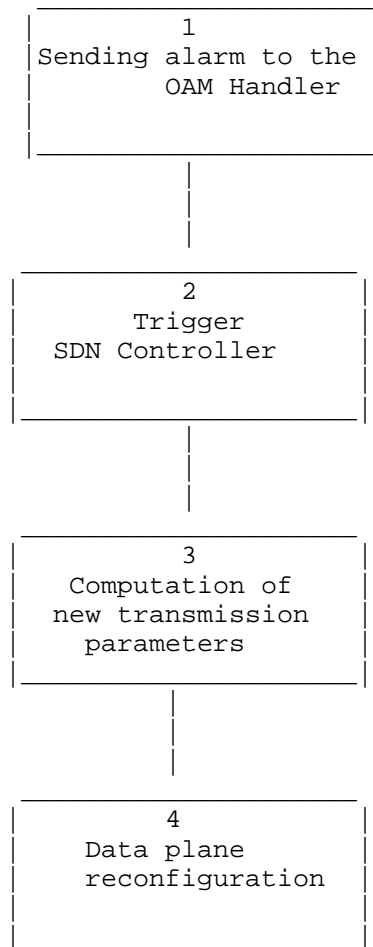


Figure 2: Flow chart of the expected state-of-the-art approach

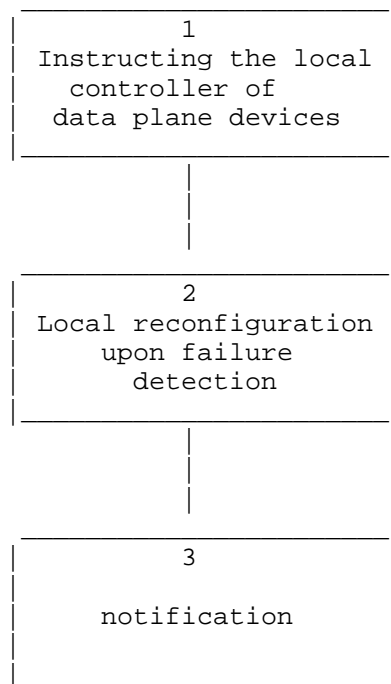


Figure 3: Flow chart of the approach exploiting YANG models in this draft

5. Extending YANG for events and reactions

The model extends YANG to define a list of events associated with specific reactions. The related code and tree are shown in the Appendix.

<event>: this element defines an event and it is composed by a set of leaves' attributes as follows.

- <name>:** this attribute defines the name of the event.
- <type>:** this attribute defines the type of the event from a pool of possible event types predefined inside the YANG model. Together with the **<name>** attribute, it uniquely identifies the event.
- <description>:** this optional attribute is a "string" describing the event
- <filters>:** this leaf is a list that enhances the description of an event. Given that an event does not necessary means a particular degradation or faults, this list can be used to define thresholds to express a measure

of the event.

- <filter>: this leaf of <filters> defines a threshold to characterize the event.
- <filter-id>: this leaf of <filters> define the identifier number associated with the <filter> attribute.

<reaction>: this attribute defines a list of operations to take if the event occurs.

- <operations>: this list defines the set of operations that have to be taken if the event occurs.
 - <id>: this leaf of <operations> defines the identifier number of an operation.
 - <type>: this leaf of <operations> defines the type of an operation.
 - <simple>: this leaf defines (differently from <conditional> detailed below) an operation that has to be directly executed.
 - <execute>: this attribute recalls an RPC encapsulating the effective task (operation) to be executed by the data plane hardware.
 - <next-operation>: this attribute defines the identification number of a next operation that has to be taken.
 - <conditional>: this leaf enables a check ("true" or "false") to be verified before executing the operation. Based on the check, the proper attributes <execute> and <next-operation> are considered.
 - <statement>: this leaf of <conditional> defines the condition to be verified before executing the operation.
 - <true>: this leaf of <conditional> defines a result of the check associated to <statement>. Proper <execute> and <next-operation> attributes are associated with this result of the check.
 - <false>: this leaf of <conditional> defines a result of the check associated to <statement>. Proper <execute> and <next-operation> attributes are

associated with this result of the check.

6. Extending YANG for finite state machine (FSM)

This model extends the one of the events and reactions by adding the state information and state transition. More precisely, the model defines a list of states associated with events. Each state has a description attribute and it is identified through an id. Each state includes a list of events as defined in the event model, with the additional next-state attribute, which points to the next state. The related code and tree are shown in the Appendix.

<current-state>: it defines the current state of the FSM.
<states>: this element defines the FSM as follows.
 <state>: this list defines all the FSM states.
 <id>: this leaf attribute of <state> defines the identifier of the state
 <name>: this leaf attribute of <state> defines the name of the state
 <description>: this leaf is a "string" describing the state
 <events>: this attribute is the one described in the previous section. In particular, this attribute defines a list of events that may induce a transition to another state in the FSM.
 <next-state>: this attribute is included in the model <events> and defines the next state of FSM when an operation is executed.

7. Implementation for the considered use case of application

The models defined in this document are an extension of YANG through functions, events, and FSM, besides data modeling. These models can be used to enable a centralized network controller, managed by a network operator, to instruct data plane hardware on its reconfiguration if some events, such as a failure or physical layer degradation, occur. As an example, an optical signal impacted by a soft failure (i.e., a physical layer degradation inducing a pre forward error correction bit error rate increase - pre-FEC) can be maintained by adapting the FEC of the signal itself. This action to be taken and, more in general operations to be executed depending on critical events, can be (re-) programmed on the transponder by (re-) sending a NETCONF <edit-config> message to the device controller including a FSM defined by the YANG model. Such a system has the main goal to speed up the reaction of the network to certain events/

faults and to alleviate the workload of the centralized controller. The speed up derives from the fact that the centralized controller is able to pre-compute and pre-configure on the network devices the actions to take when an event occurs taking into account a global view and knowledge of the network. In this way, the device is already aware of the actions to be locally applied to reconfigure a connection, avoiding to inform the controller and to wait for the response indicating what to do. Consequently, part of the workload is also removed from the centralized controller. When the reaction is successfully completed in the data plane, the centralized controller can be notified about the faults and the taken action. A flexible transponder supporting two FEC types, 7% and 20%, is considered. A two-states FSM is also assumed. The states have <name> attribute set to "Steady" and "Fec-Baud-Adapt", respectively. In the "Steady" state, the signal is in a healthy condition, adopting a 7% FEC, with a pre-FEC BER below an assigned threshold of 9×10^{-4} . A transition from this state can be triggered by the event with <name>=BER_CHANGE and <filter-type>= 9×10^{-4} , thus expressing a change of the pre-FEC BER above the threshold. In case the pre-FEC BER exceeds 9×10^{-4} due to a soft failure, the state machine evolves to the "Fec-Baud-Adapt" state and an adaptation to a more robust FEC of 20% (executed by the attribute <execute>) is performed. The system can return to the "Steady" state if the pre-FEC BER goes below another pre-defined threshold and the FEC is reconfigured to 7%.

8. Appendix

This appendix reports the YANG models code and the related tree.

8.1. YANG model for events and actions - Tree

```
+--rw events
  +--rw event [name type]
    +--rw name          string
    +--rw type           event-type
    +--rw description?   string
    +--rw filters
      | +--rw filter [filter-id]
      |   +--rw filter-id   yp:filter-id
    +--rw reaction
      +--rw operation [id]
        +--rw id           event-id-type
        +--rw type         enumeration
        +--rw conditional
          | +--rw statement   string
          | +--rw true
          | | +--rw execute
          | | +--rw next-operation?   event-id-type
          | +--rw false
          |   +--rw execute
          |   +--rw next-operation?   event-id-type
        +--rw simple
          +--rw execute
          +--rw next-operation?   event-id-type
```

8.2. YANG model for FSM - Tree

```

+--rw current-state?  leafref
+--rw states
  +--rw state [id]
    +--rw id                state-id-type
    +--rw name              string
    +--rw description?     string
    +--rw events
      +--rw event [name type]
        +--rw name          string
        +--rw type          event-type
        +--rw description?  string
        +--rw filters
          +--rw filter [filter-id]
            +--rw filter-id  yp:filter-id
        +--rw reaction
          +--rw operation [id]
            +--rw id          event-id-type
            +--rw type        enumeration
            +--rw conditional
              +--rw statement string
              +--rw true
                +--rw execute
                +--rw next-operation?  event-id-type
                +--rw next-state?     leafref
              +--rw false
                +--rw execute
                +--rw next-operation?  event-id-type
                +--rw next-state?     leafref
          +--rw simple
            +--rw execute
            +--rw next-operation?  event-id-type
            +--rw next-state?     leafref

```

8.3. YANG model for events and actions - Code

```

module events {
  namespace "http://sssup.it/events";
  prefix ev;

  import ietf-yang-push {
    prefix yp;
  }

  organization
    "Scuola Superiore Sant'Anna Network and Services Laboratory";

  contact
    " Editor: Matteo Dallaglio

```



```
        <mailto:m.dallaglio@sssup.it>
    ";

description
    "This module contains a YANG definitions of events and generic
    reactions.";

revision 2016-03-15 {
    description "Initial Revision.";
    reference
        "RFC xxxx: A YANG data model for the description of events and
        reactions";
}

// identity statements

identity EVENT {
    description "Base for all types of event";
}

identity ON_CHANGE {
    base EVENT;
    description
        "The event when the database changes.";
}

// typedef statements

typedef event-type {
    type identityref {
        base EVENT;
    }
}

typedef event-id-type {
    type uint32;
}

// grouping statements
grouping operation-block {
    leaf id {
        type event-id-type;
    }
    leaf type {
        type enumeration {
            enum CONDITIONAL_OP;
            enum SIMPLE_OP;
        }
    }
}
```

```
    }
    mandatory true;
  }

  grouping execution-top {
    anyxml execute {
      description "Represent the action to perform";
    }
    leaf next-operation {
      type event-id-type;
      description "the id of the next operation to execute";
    }
  }

  container conditional {
    when "../type = 'CONDITIONAL_OP'";
    leaf statement {
      type string;
      mandatory true;
      description
        "The statement to be evaluated before execution.
        E.g. if a=b";
    }
    container true {
      uses execution-top;
    }
    container false {
      uses execution-top;
    }
  }

  container simple {
    when "../type = 'SIMPLE_OP'";
    description
      "Simple execution of an action without checking any condition";
    uses execution-top;
  }
}

grouping operation-top {
  list operation {
    key "id";
    ordered-by user;
    uses operation-block;
  }
}

grouping on-change {
```

```
description
  "Event occurring when a modification of one or more
  objects occurs";

container filters {
  description
    "This container contains a list of configurable filters
    that can be applied to subscriptions. This facilitates
    the reuse of complex filters once defined.";
  list filter {
    key "filter-id";
    description
      "A list of configurable filters that can be applied to
      subscriptions.";
    leaf filter-id {
      type yp:filter-id;
      description
        "An identifier to differentiate between filters.";
    }
    uses yp:datatree-filter;
  }
}

grouping event-top {
  leaf name {
    type string;
    mandatory true;
  }

  leaf type {
    type event-type;
    mandatory true;
  }

  leaf description {
    type string;
  }

  // list of all possible events
  uses on-change {
    when "type = 'ON_CHANGE'";
  }

  container reaction {
    uses operation-top;
  }
}
```

```
grouping events-top {
  container events {
    list event {
      key "name type";
      uses event-top;
    }
  }
}

// data definition statements

uses events-top;

// extension statements

// feature statements

// augment statements

// rpc statements

// notification statements

} // module events
```

8.4. YANG model for FSM - Code

```
module finite-state-machine {
  namespace "http://sssup.it/fsm";
  prefix fsm;

  import events {
    prefix ev;
  }

  organization
    "Scuola Superiore Sant'Anna Network and Services Laboratory";

  contact
    " Editor: Matteo Dallaglio
      <mailto:m.dallaglio@sssup.it>
    ";

  description
    "This module contains a YANG definitions of a generic finite state
```

```

machine.";

revision 2016-03-15 {
  description "Initial Revision.";
  reference
    "RFC xxxx:";
}

// identity statements

// typedef statements

typedef state-id-type {
  type uint32;
}

// grouping statements
grouping state-top {
  leaf id {
    type state-id-type;
  }

  leaf name {
    type string;
  }

  leaf description {
    type string;
  }

  grouping next-state-top {
    leaf next-state {
      type leafref {
        path "../..//..//..//..//..//..//..//states/state/id";
      }
      description "Id of the next state";
    }
  }
}

uses ev:events-top {
  augment "events/event/reaction/operation/conditional/true" {
    uses next-state-top;
  }
  augment "events/event/reaction/operation/conditional/false" {
    uses next-state-top;
  }
}

```

```
augment "events/event/reaction/operation/simple" {  
    //uses next-state-top;  
    leaf next-state {  
        type leafref {  
            path "../.../../.../../.../../states/state/id";  
        }  
        description "Id of the next state";  
    }  
}  
}
```

```
grouping states-top {
  leaf current-state {
    type leafref {
      path "../states/state/id";
    }
  }

  container states {
    list state {
      key "id";
      uses state-top;
    }
  }
}
```

```
// data definition statements
```

```
uses states-top;
```

```
// extension statements
```

```
// feature statements
```

```
// augment statements.
```

```
// rpc statements
```

```
// notification statements
```

```
}//module fsm
```

8.5. Example of values for the YANG model

FIELD NAME	YANG DATA TYPE	VALUE
Current State	leafref	"an existing state id in the FSM"
State		
id	uint32	1
name	string	Steady
description	string	"whatever string"
event		
name	string	"whatever string"
type	enum	BER_CHANGE
description	string	"whatever string"
filter		
filter-id	uint32	2
filter-type	anyxml or xpath	BER>0.0009
reaction		
id	uint32	3
type	enum	SIMPLE
statement	string	"whatever string"
execute	anyxml	"this recalls an RPC where the FEC value is expressed"
next-operation	uint32	NULL
next-state	leafref	"an existing state id in the FSM"

9. Acknowledgements

This work has been partially supported by the European Commission through the H2020 ORCHESTRA (Optical performanCe monitoring enabling dynamic networks using a Holistic cross-layEr, Self-configurable Truly flexible approach, grant agreement no: H2020-645360) project. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

10. Security Considerations

TBD

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [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>>.
- [RFC7491] King, D. and A. Farrel, "A PCE-Based Architecture for Application-Based Network Operations", RFC 7491, DOI 10.17487/RFC7491, March 2015, <<http://www.rfc-editor.org/info/rfc7491>>.

11.2. Informative References

- [I-D.ietf-i2rs-yang-network-topo] Clemm, A., Medved, J., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A Data Model for Network Topologies", draft-ietf-i2rs-yang-network-topo-13 (work in progress), June 2017.
- [I-D.vergara-ccamp-flexigrid-yang] Madrid, U., Perdices, D., Lopezalvarez, V., Dios, O., King, D., Lee, Y., and G. Galimberti, "YANG data model for Flexi-Grid Optical Networks", draft-vergara-ccamp-flexigrid-yang-04 (work in progress), March 2017.
- [I-D.zhang-ccamp-l1-topo-yang] zhenghaomian@huawei.com, z., Fan, Z., Sharma, A., and X. Liu, "A YANG Data Model for Optical Transport Network Topology", draft-zhang-ccamp-l1-topo-yang-07 (work in progress), April 2017.

Authors' Addresses

Nicola Sambo
Scuola Superiore Sant'Anna
Via Moruzzi 1
Pisa 56124
Italy

Email: nicola.sambo@sssup.it

Matteo Dallaglio
Scuola Superiore Sant'Anna
Via Moruzzi 1
Pisa 56124
Italy

Email: matteo.dallaglio@sssup.it

Piero Castoldi
Scuola Superiore Sant'Anna
Via Moruzzi 1
Pisa 56124
Italy

Email: piero.castoldi@sssup.it

Giuseppe Fioccola
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: giuseppe.fioccola@telecomitalia.it

Andrea Di Giglio
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: andrea.digiglio@telecomitalia.it

Filippo Cugini
CNIT
Via Moruzzi 1
Pisa 56124
Italy

Email: filippo.cugini@cnit.it

Giacomo Bernini
Nextworks
Via Livornese 1027
Pisa 56122
Italy

Email: g.bernini@nextworks.it

Pietro G. Giardina
Nextworks
Via Livornese 1027
Pisa 56122
Italy

Email: p.giardina@nextworks.it

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

S. Sivakumar
Cisco Systems
M. Boucadair
Orange
S. Vinapamula
Juniper Networks
July 3, 2017

YANG Data Model for Network Address Translation (NAT)
draft-sivakumar-yang-nat-07

Abstract

For the sake of network automation and the need for programming NAT function in particular, a data model for configuring and managing the NAT device is essential. This document defines a YANG data model for the NAT function. Both the NAT44 and NAT64 are covered in this document.

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 <http://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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	2
1.2. Tree Diagrams	2
2. Overview of the NAT YANG Data Model	3
3. NAT YANG Module	9
4. Sample Examples	34
5. Security Considerations	34
6. IANA Considerations	34
7. Acknowledgements	35
8. References	35
8.1. Normative References	35
8.2. Informative References	36
Authors' Addresses	37

1. Introduction

This document defines a data model for Network Address Translation (NAT) using the YANG data modeling language [RFC6020]. Traditional NAT is defined in [RFC2663] and Carrier Grade NAT is defined in [RFC6888]. This document covers the NAT features in both documents. This document also covers the NAT64 as defined in [RFC6146].

This document assumes [RFC4787][RFC5382][RFC5508] are enabled by default.

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 [RFC2119].

The usage of the term "NAT device" in this document refer to any NAT44 and NAT64 devices. This document uses the term "Session" as it is defined in [RFC2663] and the term BIB as it is defined in [RFC6146].

1.2. Tree Diagrams

The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.

- o Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- o Abbreviations before data node names: "rw" means configuration (read-write), "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" a container with presence, and "*" denotes a "list" or "leaf-list".
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Overview of the NAT YANG Data Model

The NAT data model is designed to cover both configuration and state retrieval, nevertheless this document covers dynamic (implicit) mapping while PCP-related functionality to instruct dynamic explicit mapping is defined in [I-D.boucadair-pcp-yang].

In order to cover, in particular, both NAT64 and NAT44 flavors, the NAT mapping structure allows to include an IPv4 or IPv6 address as an internal IP address. Remaining fields are common to both NAT schemes. NPTv6 is also in scope [RFC6296].

This document assumes [RFC4787][RFC5382][RFC5508] are enabled by default. Also, the data model relies on the recommendations in [RFC6888] and [RFC7857].

A single NAT device can have multiple NAT instances; each responsible to service a group of internal hosts. This document does make any assumption how internal hosts are attached to a given NAT instance.

The data model assumes that each NAT instance can: be enable/disable, be provisioned with a dedicated configuration data, and maintain its own mapping table.

This data model assumes that blocks of IP global addresses can be provisioned to the NAT function. These blocks may be contiguous or non-contiguous [RFC6888].

A NAT function can either assign individual port numbers or port sets (e.g., [RFC7753]). Both features are supported in the YANG data model.

To accommodate deployments where [RFC6302] is not enabled, this YANG model allows to instruct a NAT function to log the destination port number. The reader may refer to [I-D.ietf-behave-ipfix-nat-logging] which provides the templates to log the destination ports.

This document does not cover the following functionalities:

- o Dynamic explicit mappings.
- o DSCP-related operations.
- o Deterministic NAT assignment scheme [RFC7422].

The tree structure of the NAT data model is provided below:

```

module: ietf-nat
  +--rw nat-config
  |   +--rw nat-instances
  |   |   +--rw nat-instance* [id]
  |   |   |   +--rw id                               uint32
  |   |   |   +--rw name?                             string
  |   |   |   +--rw enable?                           boolean
  |   |   |   +--rw external-ip-address-pool* [pool-id]
  |   |   |   |   +--rw pool-id                       uint32
  |   |   |   |   +--rw external-ip-pool?             inet:ipv4-prefix
  |   |   |   +--rw subscriber-mask-v6?               uint8
  |   |   |   +--rw subscriber-mask-v4* [sub-mask-id]
  |   |   |   |   +--rw sub-mask-id                   uint32
  |   |   |   |   +--rw sub-mask                     inet:ipv4-prefix
  |   |   |   +--rw paired-address-pooling?           boolean
  |   |   |   +--rw nat-mapping-type?                 enumeration
  |   |   |   +--rw nat-filtering-type?               enumeration
  |   |   |   +--rw port-quota?                       uint16
  |   |   |   +--rw port-set
  |   |   |   |   +--rw port-set-enable?             boolean
  |   |   |   |   +--rw port-set-size?               uint16
  |   |   |   |   +--rw port-set-timeout?            uint32
  |   |   |   +--rw port-allocation-type?             enumeration
  |   |   |   +--rw address-roundrobin-enable?        boolean
  |   |   |   +--rw udp-timeout?                      uint32
  |   |   |   +--rw tcp-idle-timeout?                 uint32
  |   |   |   +--rw tcp-trans-open-timeout?           uint32
  |   |   |   +--rw tcp-trans-close-timeout?          uint32
  |   |   |   +--rw tcp-in-syn-timeout?               uint32
  |   |   |   +--rw fragment-min-timeout?             uint32
  |   |   |   +--rw icmp-timeout?                    uint32
  |   |   |   +--rw per-port-timeout* [port-number]
  |   |   |   |   +--rw port-number                 inet:port-number
  |   |   |   |   +--rw port-timeout                 inet:port-number
  |   |   |   +--rw hold-down-timeout?                uint32

```

```

+--rw logging-info
|   +--rw destination-address      inet:ipv4-prefix
|   +--rw destination-port         inet:port-number
+--rw connection-limit
|   +--rw limit-per-subscriber?    uint32
|   +--rw limit-per-vrf?          uint32
|   +--rw limit-per-subnet?       inet:ipv4-prefix
|   +--rw limit-per-instance      uint32
|   +--rw limit-per-udp           uint32
|   +--rw limit-per-tcp           uint32
|   +--rw limit-per-icmp          uint32
+--rw mapping-limit
|   +--rw limit-per-subscriber?    uint32
|   +--rw limit-per-vrf?          uint32
|   +--rw limit-per-subnet?       inet:ipv4-prefix
|   +--rw limit-per-instance      uint32
|   +--rw limit-per-transport     uint8
+--rw ftp-alg-enable?             boolean
+--rw dns-alg-enable?             boolean
+--rw tftp-alg-enable?            boolean
+--rw msrpc-alg-enable?           boolean
+--rw netbios-alg-enable?         boolean
+--rw rcmd-alg-enable?            boolean
+--rw ldap-alg-enable?            boolean
+--rw sip-alg-enable?             boolean
+--rw rtsp-alg-enable?            boolean
+--rw h323-alg-enable?            boolean
+--rw all-algs-enable?            boolean
+--rw notify-pool-usage
|   +--rw pool-id?                uint32
|   +--rw notify-pool-hi-threshold percent
|   +--rw notify-pool-low-threshold? percent
+--rw nat64-prefixes* [nat64-prefix-id]
|   +--rw nat64-prefix-id         uint32
|   +--rw nat64-prefix?           inet:ipv6-prefix
|   +--rw destination-ipv4-prefix* [ipv4-prefix-id]
|       +--rw ipv4-prefix-id      uint32
|       +--rw ipv4-prefix?        inet:ipv4-prefix
+--rw mapping-table
|   +--rw mapping-entry* [index]
|       +--rw index               uint32
|       +--rw type?               enumeration
|       +--rw internal-src-address inet:ip-address
|       +--rw internal-src-port
|           +--rw (port-type)?
|               +--:(single-port-number)
|               |   +--rw single-port-number?    inet:port-number
|               +--:(port-range)

```

```

|         |--rw start-port-number?      inet:port-number
|         |--rw end-port-number?        inet:port-number
+--rw external-src-address      inet:ipv4-address
+--rw external-src-port
|   |--rw (port-type)?
|   |   |--:(single-port-number)
|   |   |   |--rw single-port-number?    inet:port-number
|   |   |--:(port-range)
|   |       |--rw start-port-number?      inet:port-number
|   |       |--rw end-port-number?        inet:port-number
+--rw transport-protocol        uint8
+--rw internal-dst-address?      inet:ipv4-prefix
+--rw internal-dst-port
|   |--rw (port-type)?
|   |   |--:(single-port-number)
|   |   |   |--rw single-port-number?    inet:port-number
|   |   |--:(port-range)
|   |       |--rw start-port-number?      inet:port-number
|   |       |--rw end-port-number?        inet:port-number
+--rw external-dst-address?      inet:ipv4-address
+--rw external-dst-port
|   |--rw (port-type)?
|   |   |--:(single-port-number)
|   |   |   |--rw single-port-number?    inet:port-number
|   |   |--:(port-range)
|   |       |--rw start-port-number?      inet:port-number
|   |       |--rw end-port-number?        inet:port-number
+--rw lifetime                  uint32
+--ro nat-state
+--ro nat-instances
+--ro nat-instance* [id]
+--ro id                        int32
+--ro nat-capabilities
|   +--ro nat44-support?         boolean
|   +--ro nat64-support?         boolean
|   +--ro nptv6-support?         boolean
|   +--ro static-mapping-support? boolean
|   +--ro port-set-support?       boolean
|   +--ro port-randomization-support? boolean
|   +--ro port-range-preservation-support? boolean
|   +--ro port-preservation-suport? boolean
|   +--ro port-parity-preservation-support? boolean
|   +--ro address-roundrobin-support? boolean
|   +--ro ftp-alg-support?        boolean
|   +--ro dns-alg-support?        boolean
|   +--ro tftp-support?          boolean
|   +--ro msrpc-alg-support?      boolean
|   +--ro netbios-alg-support?    boolean

```



```

| +--ro rcmd-alg-support?                boolean
| +--ro ldap-alg-support?               boolean
| +--ro sip-alg-support?                boolean
| +--ro rtsp-alg-support?               boolean
| +--ro h323-alg-support?               boolean
| +--ro paired-address-pooling-support?  boolean
| +--ro endpoint-independent-mapping-support?  boolean
| +--ro address-dependent-mapping-support?  boolean
| +--ro address-and-port-dependent-mapping-support?  boolean
| +--ro endpoint-independent-filtering-support?  boolean
| +--ro address-dependent-filtering?      boolean
| +--ro address-and-port-dependent-filtering?  boolean
+--ro nat-current-config
| +--ro external-ip-address-pool* [pool-id]
| | +--ro pool-id                      uint32
| | +--ro external-ip-pool?            inet:ipv4-prefix
| +--ro subscriber-mask-v6?            uint8
| +--ro subscriber-mask-v4* [sub-mask-id]
| | +--ro sub-mask-id                  uint32
| | +--ro sub-mask                     inet:ipv4-prefix
| +--ro paired-address-pooling?         boolean
| +--ro nat-mapping-type?               enumeration
| +--ro nat-filtering-type?             enumeration
| +--ro port-quota?                     uint16
| +--ro port-set
| | +--ro port-set-enable?              boolean
| | +--ro port-set-size?                uint16
| | +--ro port-set-timeout?             uint32
| +--ro port-allocation-type?           enumeration
| +--ro address-roundrobin-enable?      boolean
| +--ro udp-timeout?                    uint32
| +--ro tcp-idle-timeout?                uint32
| +--ro tcp-trans-open-timeout?          uint32
| +--ro tcp-trans-close-timeout?         uint32
| +--ro tcp-in-syn-timeout?              uint32
| +--ro fragment-min-timeout?            uint32
| +--ro icmp-timeout?                    uint32
| +--ro per-port-timeout* [port-number]
| | +--ro port-number                  inet:port-number
| | +--ro port-timeout                  inet:port-number
| +--ro hold-down-timeout?               uint32
| +--ro logging-info
| | +--ro destination-address            inet:ipv4-prefix
| | +--ro destination-port              inet:port-number
| +--ro connection-limit
| | +--ro limit-per-subscriber?          uint32
| | +--ro limit-per-vrf?                 uint32
| | +--ro limit-per-subnet?              inet:ipv4-prefix

```

```

|--ro limit-per-instance          uint32
+--ro limit-per-udp               uint32
+--ro limit-per-tcp              uint32
+--ro limit-per-icmp             uint32
+--ro mapping-limit
| +--ro limit-per-subscriber?    uint32
| +--ro limit-per-vrf?          uint32
| +--ro limit-per-subnet?       inet:ipv4-prefix
| +--ro limit-per-instance      uint32
| +--ro limit-per-transport     uint8
+--ro ftp-alg-enable?            boolean
+--ro dns-alg-enable?            boolean
+--ro tftp-alg-enable?           boolean
+--ro msrpc-alg-enable?          boolean
+--ro netbios-alg-enable?        boolean
+--ro rcmd-alg-enable?           boolean
+--ro ldap-alg-enable?           boolean
+--ro sip-alg-enable?            boolean
+--ro rtsp-alg-enable?           boolean
+--ro h323-alg-enable?           boolean
+--ro all-algs-enable?           boolean
+--ro notify-pool-usage
| +--ro pool-id?                 uint32
| +--ro notify-pool-hi-threshold percent
| +--ro notify-pool-low-threshold? percent
+--ro nat64-prefixes* [nat64-prefix-id]
| +--ro nat64-prefix-id          uint32
| +--ro nat64-prefix?            inet:ipv6-prefix
| +--ro destination-ipv4-prefix* [ipv4-prefix-id]
|   +--ro ipv4-prefix-id        uint32
|   +--ro ipv4-prefix?          inet:ipv4-prefix
+--ro mapping-table
+--ro mapping-entry* [index]
| +--ro index                    uint32
| +--ro type?                    enumeration
| +--ro internal-src-address     inet:ip-address
| +--ro internal-src-port
| | +--ro (port-type)?
| | | +--:(single-port-number)
| | | | +--ro single-port-number?  inet:port-number
| | | +--:(port-range)
| | | | +--ro start-port-number?    inet:port-number
| | | | +--ro end-port-number?      inet:port-number
+--ro external-src-address      inet:ipv4-address
+--ro external-src-port
| +--ro (port-type)?
| | +--:(single-port-number)
| | | +--ro single-port-number?    inet:port-number

```

```

    |
    |      +---:(port-range)
    |      |      +--ro start-port-number?      inet:port-number
    |      |      +--ro end-port-number?        inet:port-number
+--ro transport-protocol      uint8
+--ro internal-dst-address?   inet:ipv4-prefix
+--ro internal-dst-port
    |      +--ro (port-type)?
    |      |      +---:(single-port-number)
    |      |      |      +--ro single-port-number?      inet:port-number
    |      |      +---:(port-range)
    |      |      |      +--ro start-port-number?      inet:port-number
    |      |      |      +--ro end-port-number?        inet:port-number
+--ro external-dst-address?   inet:ipv4-address
+--ro external-dst-port
    |      +--ro (port-type)?
    |      |      +---:(single-port-number)
    |      |      |      +--ro single-port-number?      inet:port-number
    |      |      +---:(port-range)
    |      |      |      +--ro start-port-number?      inet:port-number
    |      |      |      +--ro end-port-number?        inet:port-number
    |      +--ro lifetime      uint32
+--ro statistics
    +--ro total-mappings?      uint32
    +--ro total-tcp-mappings?  uint32
    +--ro total-udp-mappings?  uint32
    +--ro total-icmp-mappings? uint32
    +--ro pool-stats
        +--ro pool-id?          uint32
        +--ro address-allocated? uint32
        +--ro address-free?      uint32
    +--ro port-stats
        +--ro ports-allocated?  uint32
        +--ro ports-free?       uint32

notifications:
    +---n nat-event
        +--ro id?                -> /nat-state/nat-instances/nat-instance/i
d
        +--ro notify-pool-threshold percent

```

3. NAT YANG Module

<CODE BEGINS> file "ietf-nat@2017-07-03.yang"

```

module ietf-nat {
  namespace "urn:ietf:params:xml:ns:yang:ietf-nat";
  //namespace to be assigned by IANA

```

```
    prefix "nat";
      import ietf-inet-types {
        prefix "inet";
      }
organization "IETF NetMod Working Group";
contact
  "Senthil Sivakumar <ssenthil@cisco.com>
  Mohamed Boucadair <mohamed.boucadair@orange.com>
  Suresh Vinapamula <sureshk@juniper.net>";

description
  "This module is a YANG module for NAT implementations
  (including both NAT44 and NAT64 flavors).

  Copyright (c) 2017 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
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";

revision 2017-07-03 {
  description "Integrates comments from D. Wing and T. Zhou.";
  reference "-06";
}

revision 2015-09-08 {
  description "Fixes few YANG errors.";
  reference "-02";
}

revision 2015-09-07 {
  description "Completes the NAT64 model.";
  reference "01";
}

revision 2015-08-29 {
  description "Initial version.";
  reference "00";
}

typedef percent {
```

```
        type uint8 {
            range "0 .. 100";
        }
        description
            "Percentage";
    }

    /*
     * Grouping
     */

    grouping timeouts {
        description
            "Configure values of various timeouts.";

        leaf udp-timeout {
            type uint32;
            default 300;
            description
                "UDP inactivity timeout.";
        }

        leaf tcp-idle-timeout {
            type uint32;
            default 7440;
            description
                "TCP Idle timeout, as per RFC 5382 should be no
                 2 hours and 4 minutes.";
        }

        leaf tcp-trans-open-timeout {
            type uint32;
            units "seconds";
            default 240;
            description
                "The value of the transitory open connection
                 idle-timeout.
                 Section 2.1 of [RFC7857] clarifies that a NAT
                 should provide different configurable
                 parameters for configuring the open and
                 closing idle timeouts.
                 To accommodate deployments that consider
                 a partially open timeout of 4 minutes as being
                 excessive from a security standpoint, a NAT may
                 allow the configured timeout to be less than
                 4 minutes.
                 However, a minimum default transitory connection
                 idle-timeout of 4 minutes is recommended.";
        }
    }
}
```

```
}

leaf tcp-trans-close-timeout {
    type uint32;
    units "seconds";
    default 240;
    description
        "The value of the transitory close connection
        idle-timeout.
        Section 2.1 of [RFC7857] clarifies that a NAT
        should provide different configurable
        parameters for configuring the open and
        closing idle timeouts.";
}

leaf tcp-in-syn-timeout {
    type uint32;
    default 6;
    description
        "6 seconds, as defined in [RFC5382].";
}

leaf fragment-min-timeout {
    type uint32;
    default 2;
    description
        "As long as the NAT has available resources,
        the NAT allows the fragments to arrive
        over fragment-min-timeout interval.
        The default value is inspired from RFC6146.";
}

leaf icmp-timeout {
    type uint32;
    default 60;
    description
        "60 seconds, as defined in [RFC5508].";
}

list per-port-timeout {
    key port-number;

    description
        "Some NATs are configurable with short timeouts
        for some ports, e.g., as 10 seconds on
        port 53 (DNS) and NTP (123) and longer timeouts
        on other ports.";
```

```
        leaf port-number {
            type inet:port-number;
            description
                " A port number.";
        }

        leaf port-timeout {
            type inet:port-number;
            mandatory true;
            description
                "Timeout for this port";
        }
    }

    leaf hold-down-timeout {
        type uint32;
        units "seconds";
        default 120;
        description
            "Hold down timer. Ports in the
            hold down pool are not reassigned until
            this timer expires.
            The length of time and the maximum
            number of ports in this state must be
            configurable by the administrator
            [RFC6888]. This is necessary in order
            to prevent collisions between old
            and new mappings and sessions. It ensures
            that all established sessions are broken
            instead of redirected to a different peer.
            The default value is defined in REQ#8
            from [RFC6888].";
    }
}

// port numbers: single or port range

grouping port-number {
    description
        "Individual port or a range of ports.";

    choice port-type {
        default single-port-number;
        description
            "Port type: single or port-range.";

        case single-port-number {
            leaf single-port-number {
```

```

        type inet:port-number;
        description
            "Used for single port numbers.";
    }
}

case port-range {
    leaf start-port-number {
        type inet:port-number;
        description
            "Begining of the port range.";
    }

    leaf end-port-number {
        type inet:port-number;
        description
            "End of the port range.";
    }
}
}

grouping mapping-entry {
    description
        "NAT mapping entry.";

    leaf index {
        type uint32;
        description
            "A unique identifier of a mapping entry.";
    }

    leaf type {
        type enumeration {
            enum "static" {
                description
                    "The mapping entry is manually configured.";
            }

            enum "dynamic" {
                description
                    "This mapping is created by an outgoing
                    packet.";
            }
        }
    }
    description
        "Indicates the type of a mapping entry. E.g.,
        a mapping can be: static or dynamic";
}

```



```
    }

    leaf internal-src-address {
        type inet:ip-address;
        mandatory true;
        description
            "Corresponds to the source IPv4/IPv6 address
             of the IPv4 packet";
    }

    container internal-src-port {
        description
            "Corresponds to the source port of the
             IPv4 packet.";
        uses port-number;
    }

    leaf external-src-address {
        type inet:ipv4-address;
        mandatory true;
        description
            "External IPv4 address assigned by NAT";
    }

    container external-src-port {
        description
            "External source port number assigned by NAT.";
        uses port-number;
    }

    leaf transport-protocol {
        type uint8;
        mandatory true;
        description
            "Upper-layer protocol associated with this mapping.
             Values are taken from the IANA protocol registry.
             For example, this field contains 6 (TCP) for a TCP
             mapping or 17 (UDP) for a UDP mapping.";
    }

    leaf internal-dst-address {
        type inet:ipv4-prefix;
        description
            "Corresponds to the destination IPv4 address
             of the IPv4 packet, for example, some NAT
             implementation support translating both source
             and destination address and ports referred to as
             Twice NAT";
    }
```

```
    }

    container internal-dst-port {
        description
            "Corresponds to the destination port of the
             IPv4 packet.";
        uses port-number;
    }

    leaf external-dst-address {
        type inet:ipv4-address;
        description
            "External destination IPv4 address";
    }

    container external-dst-port {
        description
            "External source port number.";
        uses port-number;
    }

    leaf lifetime {
        type uint32;
        mandatory true;
        description
            "Lifetime of the mapping.
             Tracks the connection that is
             fully-formed (e.g., 3WHS TCP
             is completd.";
    }
}

grouping nat-parameters {
    description
        "NAT parameters for a given instance";

    list external-ip-address-pool {
        key pool-id;

        description
            "Pool of external IP addresses used to service
             internal hosts.
             Both contiguous and non-contiguous pools
             can be configured for NAT.";

        leaf pool-id {
            type uint32;
            description
```

```
        "An identifier of the address pool.";
    }

    leaf external-ip-pool {
        type inet:ipv4-prefix;
        description
            "An IPv4 prefix used for NAT purposes.";
    }
}

leaf subscriber-mask-v6 {
    type uint8 {
        range "0 .. 128";
    }
    description
        "The subscriber-mask is an integer that indicates
        the length of significant bits to be applied on
        the source IP address (internal side) to
        unambiguously identify a CPE.

        Subscriber-mask is a system-wide configuration
        parameter that is used to enforce generic
        per-subscriberrpolicies (e.g., port-quota).

        The enforcement of these generic policies does not
        require the configuration of every subscriber's
        prefix.

        Example: suppose the 2001:db8:100:100::/56 prefix
        is assigned to a NAT64 serviced CPE. Suppose also
        that 2001:db8:100:100::1 is the IPv6 address used
        by the client that resides in that CPE. When the
        NAT64 receives a packet from this client,
        it applies the subscriber-mask (e.g., 56) on
        the source IPv6 address to compute the associated
        prefix for this client (2001:db8:100:100::/56).
        Then, the NAT64 enforces policies based on that
        prefix (2001:db8:100:100::/56), not on the exact
        source IPv6 address."
}

list subscriber-mask-v4 {

    key sub-mask-id;

    description
```

```
    "IPv4 subscriber mask.";

    leaf sub-mask-id {
        type uint32;
        description
            "An identifier of the subscriber masks.";
    }
    leaf sub-mask {
        type inet:ipv4-prefix;
        mandatory true;
        description
            "The IP address subnets that matches
            should be translated. E.g., If the
            private realms that are to be translated
            by NAT would be 192.0.2.0/24";
    }
}

leaf paired-address-pooling {
    type boolean;
    default true;
    description
        "Paired address pooling is indicating to NAT
        that all the flows from an internal IP
        address must be assigned the same external
        address. This is defined in RFC 4007.";
}

leaf nat-mapping-type {
    type enumeration {
        enum "eim" {
            description
                "endpoint-independent-mapping.
                Refer section 4 of RFC 4787.";
        }

        enum "adm" {
            description
                "address-dependent-mapping.
                Refer section 4 of RFC 4787.";
        }

        enum "edm" {
            description
                "address-and-port-dependent-mapping.
                Refer section 4 of RFC 4787.";
        }
    }
}
```

```
        description
            "Indicates the type of a NAT mapping.";
    }
    leaf nat-filtering-type {
        type enumeration {
            enum "eif" {
                description
                    "endpoint-independent- filtering.
                     Refer section 5 of RFC 4787.";
            }

            enum "adf" {
                description
                    "address-dependent- filtering.
                     Refer section 5 of RFC 4787.";
            }

            enum "edf" {
                description
                    "address-and-port-dependent- filtering.
                     Refer section 5 of RFC 4787.";
            }
        }
        description
            "Indicates the type of a NAT filtering.";
    }

    leaf port-quota {
        type uint16;
        description
            "Configures a port quota to be assigned per
             subscriber. It corresponds to the maximum
             number of ports to be used by a subscriber.";
    }

    container port-set {
        description
            "Manages port-set assignments.";

        leaf port-set-enable {
            type boolean;
            description
                "Enable/Disable port set assignment.";
        }

        leaf port-set-size {
            type uint16;
            description
```

```
        "Indicates the size of assigned port
        sets.";
    }

    leaf port-set-timeout {
        type uint32;
        description
            "Inactivty timeout for port sets.";
    }
}

leaf port-allocation-type {
    type enumeration {
        enum "random" {
            description
                "Port port randomization.";
        }

        enum "port-preservation" {
            description
                "Indicates whether the PCP server should
                preserve the internal port number.";
        }

        enum "port-range-preservation" {
            description
                "Indicates whether the NAT device should
                preserve the internal port range.";
        }

        enum "port-parity-preservation" {
            description
                "Indicates whether the PCP server should
                preserve the port parity of the
                internal port number.";
        }
    }
    description
        "Indicates the type of a NAT mapping.";
}

leaf address-roundrobin-enable {
    type boolean;
    description
        "Enable/disable address allocation
        round robin.";
}
```

```
uses timeouts;
container logging-info {
  description
    "Information about Logging NAT events";

  leaf destination-address {
    type inet:ipv4-prefix;
    mandatory true;
    description
      "Address of the collector that receives
       the logs";
  }
  leaf destination-port {
    type inet:port-number;
    mandatory true;
    description
      "Destination port of the collector.";
  }
}
container connection-limit {
  description
    "Information on the config parameters that
     rate limit the translations based on various
     criteria";

  leaf limit-per-subscriber {
    type uint32;
    description
      "Maximum number of NAT mappings per
       subscriber.";
  }
  leaf limit-per-vrf {
    type uint32;
    description
      "Maximum number of NAT mappings per
       VLAN/VRF.";
  }
  leaf limit-per-subnet {
    type inet:ipv4-prefix;
    description
      "Maximum number of NAT mappings per
       subnet.";
  }
  leaf limit-per-instance {
    type uint32;
    mandatory true;
    description
```

```
        "Maximum number of NAT mappings per
        instance.";
    }
    leaf limit-per-udp {
        type uint32;
        mandatory true;
        description
            "Maximum number of UDP NAT mappings per
            instance.";
    }
    leaf limit-per-tcp {
        type uint32;
        mandatory true;
        description
            "Maximum number of TCP NAT mappings per
            instance.";
    }
    leaf limit-per-icmp {
        type uint32;
        mandatory true;
        description
            "Maximum number of ICMP NAT mappings per
            instance.";
    }
}
container mapping-limit {
    description
        "Information on the config parameters that
        rate limit the mappings based on various
        criteria";

    leaf limit-per-subscriber {
        type uint32;
        description
            "Maximum number of NAT mappings per
            subscriber.";
    }
    leaf limit-per-vrf {
        type uint32;
        description
            "Maximum number of NAT mappings per
            VLAN/VRF.";
    }
    leaf limit-per-subnet {
        type inet:ipv4-prefix;
        description
            "Maximum number of NAT mappings per
```



```
        subnet.";
    }
    leaf limit-per-instance {
        type uint32;
        mandatory true;
        description
            "Maximum number of NAT mappings per
            instance.";
    }

    leaf limit-per-transport {
        type uint32;
        mandatory true;
        description
            "Maximum number of NAT mappings per
            transport protocol.";
    }
}
leaf ftp-alg-enable {
    type boolean;
    description
        "Enable/Disable FTP ALG";
}

leaf dns-alg-enable {
    type boolean;
    description
        "Enable/Disable DNSALG";
}

leaf tftp-alg-enable {
    type boolean;
    description
        "Enable/Disable TFTP ALG";
}

leaf msrpc-alg-enable {
    type boolean;
    description
        "Enable/Disable MS-RPC ALG";
}

leaf netbios-alg-enable {
    type boolean;
    description
        "Enable/Disable NetBIOS ALG";
}
```

```
leaf rcmd-alg-enable {
    type boolean;
    description
        "Enable/Disable rcmd ALG";
}

leaf ldap-alg-enable {
    type boolean;
    description
        "Enable/Disable LDAP ALG";
}

leaf sip-alg-enable {
    type boolean;
    description
        "Enable/Disable SIP ALG";
}

leaf rtsp-alg-enable {
    type boolean;
    description
        "Enable/Disable RTSP ALG";
}

leaf h323-alg-enable {
    type boolean;
    description
        "Enable/Disable H323 ALG";
}

leaf all-algs-enable {
    type boolean;
    description
        "Enable/Disable all the ALGs";
}

container notify-pool-usage {
    description
        "Notification of Pool usage when certain criteria
        is met";

    leaf pool-id {
        type uint32;
        description
            "Pool-ID for which the notification
            criteria is defined";
    }
}
```

```
leaf notify-pool-hi-threshold {
    type percent;
    mandatory true;
    description
        "Notification must be generated when the
        defined high threshold is reached.
        For example, if a notification is
        required when the pool utilization reaches
        90%, this configuration parameter must
        be set to 90%";
}

leaf notify-pool-low-threshold {
    type percent;
    description
        "Notification must be generated when the defined
        low threshold is reached.
        For example, if a notification is required when
        the pool utilization reaches below 10%,
        this configuration parameter must be set to
        10%";
}
}
list nat64-prefixes {
    key nat64-prefix-id;

    description
        "Provides one or a list of NAT64 prefixes
        With or without a list of destination IPv4 prefixes.

        Destination-based Pref64::/n is discussed in
        Section 5.1 of [RFC7050]). For example:
        192.0.2.0/24 is mapped to 2001:db8:122:300::/56.
        198.51.100.0/24 is mapped to 2001:db8:122::/48.";

    leaf nat64-prefix-id {
        type uint32;
        description
            "An identifier of the NAT64 prefix.";
    }

    leaf nat64-prefix {
        type inet:ipv6-prefix;
        default "64:ff9b::/96";
        description
            "A NAT64 prefix. Can be NSP or WKP [RFC6052].";
    }
}
```

```
    list destination-ipv4-prefix {
        key ipv4-prefix-id;

        description
            "An IPv4 prefix/address.";

        leaf ipv4-prefix-id {
            type uint32;
            description
                "An identifier of the IPv4 prefix/address.";
        }

        leaf ipv4-prefix {
            type inet:ipv4-prefix;
            description
                "An IPv4 address/prefix. ";
        }
    }
} //nat-parameters group

container nat-config {
    description
        "NAT";

    container nat-instances {
        description
            "nat instances";

        list nat-instance {

            key "id";

            description
                "A NAT instance.";

            leaf id {
                type uint32;
                description
                    "NAT instance identifier [RFC7659].";
            }

            leaf name {
                type string;
                description
                    "A name associated with the NAT instance.";
            }
        }
    }
}
```

```
    leaf enable {
      type boolean;
      description
        "Status of the the NAT instance.";
    }

    uses nat-parameters;

    container mapping-table {
      description
        "NAT dynamic mapping table used to track
        sessions";

      list mapping-entry {
        key "index";
        description
          "NAT mapping entry.";
        uses mapping-entry;
      }
    }
  }
}

/*
 * NAT State
 */

container nat-state {

  config false;

  description
    "nat-state";

  container nat-instances {
    description
      "nat instances";

    list nat-instance {
      key "id";

      description
        "nat instance";

      leaf id {
        type int32;
        description
```

```
        "The identifier of the nat instance.";
    }

    container nat-capabilities {
        description
            "NAT Capabilities";

        leaf nat44-support {
            type boolean;
            description
                "Indicates NAT44 support";
        }

        leaf nat64-support {
            type boolean;
            description
                "Indicates NAT64 support";
        }

        leaf nptv6-support {
            type boolean;
            description
                "Indicates NPTv6 support";
        }

        leaf static-mapping-support {
            type boolean;
            description
                "Indicates whether static mappings are
                supported.";
        }

        leaf port-set-support {
            type boolean;
            description
                "Indicates port set assignment
                support ";
        }

        leaf port-randomization-support {
            type boolean;
            description
                "Indicates whether port randomization is
                supported.";
        }

        leaf port-range-preservation-support {
            type boolean;
        }
    }
}
```

```
        description
        "Indicates whether port range
        preservation is supported.";
    }

    leaf port-preservation-suport {
        type boolean;
        description
        "Indicates whether port preservation
        is supported.";
    }

    leaf port-parity-preservation-support {
        type boolean;
        description
        "Indicates whether port parity
        preservation is supported.";
    }

    leaf address-roundrobin-support {
        type boolean;
        description
        "Indicates whether address allocation
        round robin is supported.";
    }

    leaf ftp-alg-support {
        type boolean;
        description
        "Indicates whether FTP ALG is supported";
    }

    leaf dns-alg-support {
        type boolean;
        description
        "Indicates whether DNSALG is supported";
    }

    leaf tftp-support {
        type boolean;
        description
        "Indicates whether TFTP ALG is supported";
    }

    leaf msrpc-alg-support {
        type boolean;
        description
        "Indicates whether MS-RPC ALG is supported";
    }
```

```
    }

    leaf netbios-alg-support {
        type boolean;
        description
            "Indicates whether NetBIOS ALG is supported";
    }

    leaf rcmd-alg-support {
        type boolean;
        description
            "Indicates whether rcmd ALG is supported";
    }

    leaf ldap-alg-support {
        type boolean;
        description
            "Indicates whether LDAP ALG is supported";
    }

    leaf sip-alg-support {
        type boolean;
        description
            "Indicates whether SIP ALG is supported";
    }

    leaf rtsp-alg-support {
        type boolean;
        description
            "Indicates whether RTSP ALG is supported";
    }

    leaf h323-alg-support {
        type boolean;
        description
            "Indicates whether H323 ALG is supported";
    }

    leaf paired-address-pooling-support {
        type boolean;
        description
            "Indicates whether paired-address-pooling is
            supported";
    }

    leaf endpoint-independent-mapping-support {
        type boolean;
        description
```



```
        "Indicates whether endpoint-independent-mapping
        in Section 4 of RFC 4787 is supported.";
    }

    leaf address-dependent-mapping-support {
        type boolean;
        description
            "Indicates whether endpoint-independent-mapping
            in Section 4 of RFC 4787 is supported.";
    }

    leaf address-and-port-dependent-mapping-support {
        type boolean;
        description
            "Indicates whether endpoint-independent-mapping in
            section 4 of RFC 4787 is supported.";
    }

    leaf endpoint-independent-filtering-support {
        type boolean;
        description
            "Indicates whether endpoint-independent-mapping in
            section 5 of RFC 4787 is supported.";
    }

    leaf address-dependent-filtering {
        type boolean;
        description
            "Indicates whether endpoint-independent-mapping in
            section 5 of RFC 4787 is supported.";
    }

    leaf address-and-port-dependent-filtering {
        type boolean;
        description
            "Indicates whether endpoint-independent-mapping in
            section 5 of RFC 4787 is supported.";
    }
}

    container nat-current-config {
        description
            "current config";

        uses nat-parameters;
    }
```

```
container mapping-table {
  description
    "Mapping table";
  list mapping-entry {
    key "index";
    description
      "mapping entry";
    uses mapping-entry;
  }
}

container statistics {
  description
    "Statistics related to the NAT instance";

  leaf total-mappings {
    type uint32;
    description
      "Total number of NAT Mappings present
      at the time. This includes all the
      static and dynamic mappings";
  }
  leaf total-tcp-mappings {
    type uint32;
    description
      "Total number of TCP Mappings present
      at the time.";
  }
  leaf total-udp-mappings {
    type uint32;
    description
      "Total number of UDP Mappings present
      at the time.";
  }
  leaf total-icmp-mappings {
    type uint32;
    description
      "Total number of ICMP Mappings present
      at the time.";
  }
  container pool-stats {
    description
      "Statistics related to Pool usage";
    leaf pool-id {
      type uint32;
      description
        "Unique Identifier that represents
        a pool";
    }
  }
}
```

```

    }
    leaf address-allocated {
        type uint32;
        description
            "Number of allocated addresses in
            the pool";
    }
    leaf address-free {
        type uint32;
        description
            "Number of free addresses in
            the pool. The sum of free
            addresses and allocated
            addresses are the total
            addresses in the pool";
    }
    container port-stats {
        description
            "Statistics related to port
            usage.";

        leaf ports-allocated {
            type uint32;
            description
                "Number of allocated ports
                in the pool";
        }

        leaf ports-free {
            type uint32;
            description
                "Number of free addresses
                in the pool";
        }
    }
}
} //statistics
} //nat-instance
} //nat-instances
} //nat-state
/*
 * Notifications
 */
notification nat-event {
    description
        "Notifications must be generated when the defined
        high/low threshold is reached. Related configuration
        parameters must be provided to trigger

```

```
        the notifications.";

    leaf id {
        type leafref {
            path
                "/nat-state/nat-instances/"
                + "nat-instance/id";
        }
        description
            "NAT instance ID.";
    }

    leaf notify-pool-threshold {
        type percent;
        mandatory true;
        description
            "A treshhold has been fired.";
    }
}
} //module nat
<CODE ENDS>
```

4. Sample Examples

TBC

5. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the support of SSH is mandatory to implement secure transport [RFC6242]. The NETCONF access control model [RFC6536] provides means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and contents.

All data nodes defined in the YANG module which can be created, modified and deleted (i.e., config true, which is the default). These data nodes are considered sensitive. Write operations (e.g., edit-config) applied to these data nodes without proper protection can negatively affect network operations.

6. IANA Considerations

This document requests IANA to register the following URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-nat
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document requests IANA to register the following YANG module in the "YANG Module Names" registry [RFC6020].

name: ietf-nat
namespace: urn:ietf:params:xml:ns:yang:ietf-nat
prefix: nat
reference: RFC XXXX

7. Acknowledgements

Many thanks to Dan Wing and Tianran Zhou for the review.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<http://www.rfc-editor.org/info/rfc4787>>.
- [RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <<http://www.rfc-editor.org/info/rfc5382>>.
- [RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, DOI 10.17487/RFC5508, April 2009, <<http://www.rfc-editor.org/info/rfc5508>>.

- [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>>.
- [RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <<http://www.rfc-editor.org/info/rfc6146>>.
- [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>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC6888] Perreault, S., Ed., Yamagata, I., Miyakawa, S., Nakagawa, A., and H. Ashida, "Common Requirements for Carrier-Grade NATs (CGNs)", BCP 127, RFC 6888, DOI 10.17487/RFC6888, April 2013, <<http://www.rfc-editor.org/info/rfc6888>>.
- [RFC7857] Penno, R., Perreault, S., Boucadair, M., Ed., Sivakumar, S., and K. Naito, "Updates to Network Address Translation (NAT) Behavioral Requirements", BCP 127, RFC 7857, DOI 10.17487/RFC7857, April 2016, <<http://www.rfc-editor.org/info/rfc7857>>.

8.2. Informative References

- [I-D.boucadair-pcp-yang]
Boucadair, M., Jacquenet, C., Sivakumar, S., and S. Vinapamula, "YANG Data Models for the Port Control Protocol (PCP)", draft-boucadair-pcp-yang-04 (work in progress), May 2017.
- [I-D.ietf-behave-ipfix-nat-logging]
Sivakumar, S. and R. Penno, "IPFIX Information Elements for logging NAT Events", draft-ietf-behave-ipfix-nat-logging-13 (work in progress), January 2017.

- [RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, DOI 10.17487/RFC2663, August 1999, <<http://www.rfc-editor.org/info/rfc2663>>.
- [RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011, <<http://www.rfc-editor.org/info/rfc6296>>.
- [RFC6302] Durand, A., Gashinsky, I., Lee, D., and S. Sheppard, "Logging Recommendations for Internet-Facing Servers", BCP 162, RFC 6302, DOI 10.17487/RFC6302, June 2011, <<http://www.rfc-editor.org/info/rfc6302>>.
- [RFC7422] Donley, C., Grundemann, C., Sarawat, V., Sundaresan, K., and O. Vautrin, "Deterministic Address Mapping to Reduce Logging in Carrier-Grade NAT Deployments", RFC 7422, DOI 10.17487/RFC7422, December 2014, <<http://www.rfc-editor.org/info/rfc7422>>.
- [RFC7659] Perreault, S., Tsou, T., Sivakumar, S., and T. Taylor, "Definitions of Managed Objects for Network Address Translators (NATs)", RFC 7659, DOI 10.17487/RFC7659, October 2015, <<http://www.rfc-editor.org/info/rfc7659>>.
- [RFC7753] Sun, Q., Boucadair, M., Sivakumar, S., Zhou, C., Tsou, T., and S. Perreault, "Port Control Protocol (PCP) Extension for Port-Set Allocation", RFC 7753, DOI 10.17487/RFC7753, February 2016, <<http://www.rfc-editor.org/info/rfc7753>>.

Authors' Addresses

Senthil Sivakumar
Cisco Systems
7100-8 Kit Creek Road
Research Triangle Park, North Carolina 27709
USA

Phone: +1 919 392 5158
Email: ssenthil@cisco.com

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Suresh Vinapamula
Juniper Networks
1133 Innovation Way
Sunnyvale 94089
USA

OPSAWG
Internet-Draft
Intended status: Informational
Expires: December 21, 2017

H. Song, Ed.
J. Gong
Huawei Technologies Co., Ltd
June 19, 2017

Requirements for Interactive Query with Dynamic Network Probes
draft-song-opsawg-dnp4iq-01

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 and configurable data plane is becoming a reality, it can be extended 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.

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 <http://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 December 21, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Motivation for Interactive Query with DNP	3
3. Use Cases	5
3.1. In-Situ OAM with User Defined Data Collection	6
3.2. DDoS Detection	6
3.3. Elephant Flow Identification	6
3.4. Network Congestion Monitoring	7
4. Enabling Technologies for DNP	7
5. Dynamic Network Probes	9
5.1. DNP Types	11
5.1.1. Node Based	11
5.1.2. Path Based	12
5.1.3. Flow Based	13
6. Interactive Query Architecture	13
7. Requirements for IQ with DNP	14
8. Considerations for IQ with DNP	15
8.1. Technical Challenges	15
8.2. Standard Consideration	16
9. Security Considerations	16
10. IANA Considerations	16
11. Acknowledgments	16
12. Informative References	16
Authors' Addresses	18

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 preprocess 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.

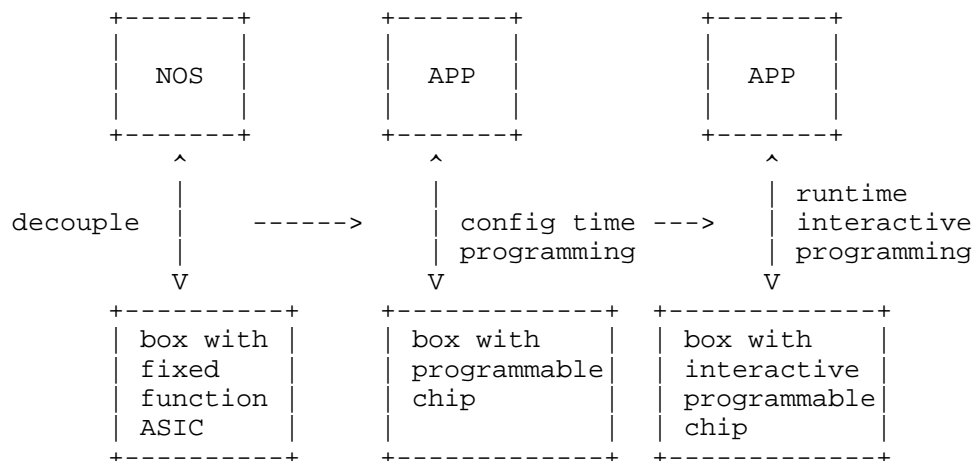


Figure 1: Towards User Programmable Data Plane

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.

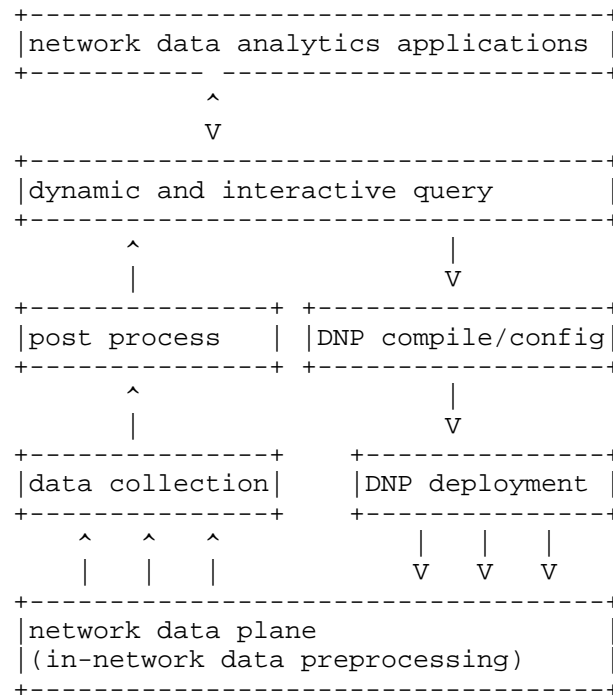


Figure 2: Architecture of IQ with DNP

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:

- o 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.
- o DNPs can be deployed through some standard south bound interface and protocols such as gRPC, NETCONF, etc.
- o The interactive query should not modify the forwarding behavior. The API should provide the necessary isolation.

- o 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.
- o The system should support multiple parallel queries from multiple applications.
- o One application can deploy different DNPs to a set of network nodes and these DNPs work jointly to finish a function.
- o 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:

- o 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.
- o 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.
- o 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