Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 13, 2020

K. Inamdar
S. Narayanan
C. Jennings
Cisco Systems

September 10, 2019

Automatic Peering for SIP Trunks
draft-kinamdar-dispatch-sip-audo-peer-00

Abstract

This draft specifies a configuration workflow to enable enterprise Session Initiation Protocol (SIP) networks to solicit the capability set of a SIP service provider network. The capability set can subsequently be used to configure features and services on the enterprise edge element, such as a Session Border Controller (SBC), to ensure smooth peering between enterprise and service provider networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The deployment of a Session Initiation Protocol (SIP)-based infrastructure in enterprise and service provider communication networks is increasing at a rapid pace. Consequently, direct IP peering between enterprise and service provider networks is quickly replacing traditional methods of interconnection between enterprise and service provider networks.

Currently published standards provide a strong foundation over which direct IP peering can be realized. However, given the sheer number of these standards, it is often not clear which behavioral subsets,

extensions to baseline protocols and operating principles ought to be implemented by service provider and enterprise networks to ensure successful peering.

The SIP Connect technical recommendations aim to solve this problem by providing a master reference that promotes seamless peering between enterprise and service provider SIP networks. However, despite the extensive set of implementation rules and operating guidelines, interoperability issues between service provider and enterprise networks persist. This is in large part because service providers and equipment manufacturers aren't required to enforce the guidelines of the technical specifications and have a fair degree of freedom to deviate from them. Consequently, enterprise administrators usually undertake a fairly rigorous regimen of testing, analysis and troubleshooting to arrive at a configuration block that ensures seamless service provider peering.

Another set of interoperability problems arise when enterprise administrators are required to translate a set of technical recommendations from service providers to configuration blocks across one or more devices in the enterprise, which is usually an error prone exercise. Additionally, such technical recommendations might not be nuanced enough to intuitively allow the generation of specific configuration blocks.

This draft introduces a mechanism using which an enterprise network can solicit a detailed capability set from a SIP service provider; the detailed capability set can subsequently be used by automaton or an administrator to generate configuration blocks across one or more devices within the enterprise to ensure successful service provider peering.

2. Conventions and Terminology

The The keywords "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]

3. Reference Architecture

Figure 1 illustrates a reference architecture that may be deployed to support the mechanism described in this document. The enterprise network consists of a SIP-PBX, media endpoints and a Session Border Controller. It may also include additional components such as application servers for voicemail, recording, fax etc. At a high level, the service provider consists of a SIP signaling entity (SP-SSE), a media entity and a HTTP(S) server.

```
    +-----------------------------------------------------+
    | +---------------+         +-----------------------+ |
    | |               |         |                       | |
    | | +----------+  |         |   +-------+           | |
    | | |   Cap    |  | HTTPS   |   |       |           | |
    | | |  Server  |<-|---------|-->|       |           | |
    | | |          |  |         |   |       |   +-----+ | |
    | | +----------+  |         |   |       |   | SIP | | |
    | |               |         |   |       |<->| PBX | | |
    | |               |         |   |       |   +-----+ | |
    | | +----------+  |         |   |  SBC  |           | |
    | | |          |  |   SIP   |   |       |           | |
    | | | SP - SSE |<-|---------|-->|       |   +-----+ | |
    | | |          |  |         |   |       |<->| M.E.| | |
    | | +----------+  |         |   |       |   |     | | |
    | |               |         |   |       |   +-----+ | |
    | | +----------+  |  SRTP   |   |       |           | |
    | | |  Media   |<-|---------|-->+-------+           | |
    | | +----------+  |         |                       | |
    | +---------------+         +-----------------------+ |
|                                                     |
    +-----------------------------------------------------+
```

Figure 1: Reference Architecture

This draft makes use of the following terminology:

- o Enterprise Network: A communications network infrastructure deployed by an enterprise which interconnects with the service provider network over SIP. The enterprise network could include devices such as application servers, endpoints, call agents and edge devices among others.

- o Edge Device: A device that is the last hop in the enterprise network and that is the transit point for traffic entering and leaving the enterprise. An edge device is typically a back-to-back user agent (B2BUA) such as a Session Border Controller (SBC).

- o Service Provider Network: A communications network infrastructure deployed by service providers. In the context of this draft, the service provider network is accessible over SIP for the establishment, modification and termination of calls and accessible over HTTP(S) for the transfer of the capability set document. The service provider network is also referred to as a SIP Service Provider (SSP) or Internet Telephony Service Provider (ITSP) network. These networks typically interconnect with other service provider networks over SIP or ISDN.

- Call Control: Call Control within a telephony networks refers to software that is responsible for delivering its core functionality. Call control not only provides the basic functionality of setting up, sustaining and terminating calls, but also provides the necessary control and logic required for additional services within the telephony network.

- Capability Server: A server hosted in the service provider network, such that this server is the target for capability set document requests from the enterprise network.

- Capability Set: This specification uses the term capability set (or capability set document) to refer collectively to a set of characteristics within the service provider network, which when communicated to the enterprise network allows it to obtain sufficient information required to peer successfully with the service provider network.

4. Configuration Workflow

A workflow that facilitates an enterprise network to solicit the capability set of a SIP service provider ought to take into account the following considerations:

- The configuration workflow must be based on a protocol or a set of protocols commonly used between enterprise and service provider telephony networks.

- The configuration workflow must be flexible enough to allow the service provider network to dynamically offload different capability sets to different enterprise networks based on the identity of the enterprise network.

- Capability set documents obtained as a result of the configuration workflow must be conducive to easy parsing by automaton. Subsequently, automaton may be used for generation of appropriate configuration blocks.

Taking the above considerations into account, this document proposes a Hypertext Transfer Protocol (HTTP)-based workflow using which the enterprise network can solicit and ultimately obtain the service provider capability set. The enterprise network creates a well formed HTTPS GET request to solicit the service provider capability set. Subsequently, the HTTPS response from the SIP service provider includes the capability set. The capability set is encoded in either XML or JSON, thus ensuring that the response can be easily parsed by automaton.

There are alternative mechanisms using which the SIP service provider can offload its capability set. For example, the Session Initiation Protocol (SIP) can be extended to define a new event package [RFC6665], such that the enterprise network can establish a SIP subscription with the service provider for its capability set; the SIP service provider can subsequently use the SIP NOTIFY request to communicate its capability set or any state deltas to its baseline capability set. This mechanism is likely to result in a significant amount of operational complexity. For example, not only would this workflow require enterprise and service provider equipment manufacturers to upgrade their software stacks, but it would also create a significant amount of ambiguity in terms of which device in the service provider network handles subscriptions and generates notifications. Yet another example of an alternative mechanism would be for service providers and enterprise equipment manufacturers to agree on YANG models [RFC6020] that enable configuration to be pushed over NETCONF [RFC6241] to enterprise networks from a centralized source hosted in service provider networks. The presence of proprietary software logic for call and media handling in enterprise devices would preclude the generation of a "one-size-fits-all" YANG model. Additionally, service provider networks pushing configuration to enterprises devices might lead to the loss of implementation autonomy on the part of the enterprise network.

5. Overview of Operations

To solicit the capability set of the SIP service provider, the edge element in the enterprise network generates a well-formed HTTPS GET request. There are two reasons why it makes sense for the enterprise edge element to generate the HTTPs request:

1. Edge elements are devices that normalize any mismatches between the enterprise and service provider networks in the media and signaling planes. As a result, when the capability set is received from the SIP service provider network, the edge element can generate appropriate configuration blocks (possibly across multiple devices) to enable smooth IP peering.
2. Given that edge elements are configured to "talk" to networks external to the enterprise, the complexity in terms of NAT traversal and firewall configuration would be minimal.

The HTTPs GET request is targeted at a capability server that is managed by the SIP service provider such that this server processes, and on successfully processing the request, includes the capability set document in the response. The capability set document is constructed according the guidelines of the YANG model described in this draft. The capability set document included in a successful response is formatted in either XML or JSON. The formatting depends

on the value of the "Accept" header field of the HTTP GET request. More details about the formatting of the HTTP request and response are provided in Section 6

Figure 1 provides a reference architecture in which this workflow may be implemented. The architecture depicted in Figure 1 consists of an enterprise telephony network and a SIP service provider network, such that the enterprise network attempts to provision SIP trunking services for the first time. For the sake of simplicity, the enterprise and service provider networks are decomposed into their core constituent elements.

6. HTTP Transport

This section describes the use of HTTP as a transport protocol for the peering workflow. This workflow is based on HTTP version 1.1

6.1. HTTP Methods

The workflow defined in this document leverages the HTTP GET method and its corresponding response(s) to request for and subsequently obtain the service provider capability set document. The HTTP POST method and its corresponding response(s) is also used for client authentication.

To ensure the smooth operation of this workflow, this draft enforces certain controls (not to be confused with HTTP controls as defined in [RFC7231] on the HTTP client and server. These controls as they relate to formatting, operational guidelines, security concerns and more, are detailed in subsequent sections of this draft.

6.2. Integrity and Confidentiality

Peering requests and responses are defined over HTTP. However, due to the sensitive nature of information transmitted between client and server, it is required to secure HTTP using Transport Layer Security. The enterprise edge element and capability server MUST be compliant to [RFC7235]. The enterprise edge element and capability server MUST support the use of the https uri scheme as defined in [RFC7230].

6.3. Authenticated Client Identity

The configuration workflow and corresponding YANG model described in this draft allow for smooth IP peering between enterprise and SIP service provider networks by encoding the essential session and media characteristics. It is NOT RECOMMENDED to encode information that is sensitive in nature. It is only required for the client (enterprise edge element) to authenticate the SIP service provider. If however,

there is a need for the SIP service provider to authenticate the enterprise edge element, client authentication mechanisms based on OAuth 2.0 token are RECOMMENDED. The specifics of how the client obtains such tokens is outside the scope of this draft. Section 11 provides an example of how clients may be authenticated using OAuth 2.0 tokens.

6.4. Encoding the Request

The edge element in the enterprise network generates a HTTPS GET request such that the request-target is obtained using the procedure outlined in section 6.6 The MIME types for the capability set document defined in this draft are "application/peering-info+json" and "application/peering-info+xml". Accordingly, the Accept header field value MUST be restricted only to these MIME types. It is possible that the edge element supports responses formatted in both JSON and XML. In such situations, the edge element might generate a HTTPS GET request such that the Accept header field includes both MIME types along with the corresponding "qvalue" for each MIME type. For implementations that require client authentication, the bearer access token acquired by the client (see section 6.3) MUST be presented to the capability server to the obtain the capability set document. The bearer token is presented in the "Authorization" header field of the GET request as specified in Section 2.1 of [RFC6750].

The generated HTTPS GET request MUST NOT use the "Expect" and "Range" header fields. The requests also MUST NOT use any conditional request.

6.5. Generating the Response

Capability servers include the capability set documents in the body of a successful response. Capability set documents MUST be formatted in XML or JSON. For requests that are incorrectly formatted, the capability server SHOULD generate a "400 Bad Request" response. If the client (enterprise edge element) includes any other MIME types in Accept header field other than "application/peering-info+json" or "application/peering-info+xml", the capability set MUST reject the request with a "406 Not Acceptable" response.

The capability server can respond to client requests with redirect responses, specifically, the server can respond with the following redirect responses:

1. 301 Moved Temporarily

2. 302 Found

3. 307 Temporary Redirect

The server SHOULD include the Location header field in such responses.

For requests that carry an invalid bearer token in the Authorization header field, the capability server SHOULD respond with a HTTP 401 status code.

6.6. Identifying the Request Target

HTTPS GET requests from enterprise edge elements MUST carry a valid request-target. The enterprise edge element might obtain the URL of the resource hosted on the capability server in one of two ways:

1. Manual Configuration
2. Using the WebFinger Protocol

The complete HTTPS URLs to be used when authenticating the enterprise edge element (optional) and obtaining the SIP service provider capability set can be obtained from the SIP service provider beforehand and entered into the edge element manually via some interface - for example, a CLI or GUI.

However, if the resource URL is unknown to the administrator (and by extension of that to the edge element), the WebFinger protocol [RFC7033] may be leveraged. From the perspective of this draft, three link relation types (rel) are defined, namely:

1. Capability Server: The base URL of the capability server hosting the capability set document.
2. Authorization Endpoint: The URL of the authorization endpoint to be used for OAuth 2.0
3. Token Endpoint: The URL of the token endpoint to be used for OAuth 2.0

The corresponding link relation type values are as follows:

1. Capability Server: "<http://sipserviceprovider/capserver">
2. Authorization Endpoint: "<http://sipserviceprovider/auth">
3. Token Endpoint: "<http://sipserviceprovider/token">

If an enterprise edge element attempts to discover the URL of the endpoints hosted in the ssp1.example.com domain, it issues the following request (line wraps are for display purposes only)

```
              GET /.well-known/webfinger?
                resource=http%3A%2F%2Fssp1.example.com
                &rel=http%3A%2f%2fsipserviceprovider%2fcapserver
                &rel=http%3A%2f%2fsipserviceprovider%2auth
                &rel= http%3A%2f%2fsipserviceprovider%2token
                HTTP/1.1
              Host: ssp1.example.com
```

The response to the above request might be as follows:

```
             HTTP/1.1 200 OK
             Access-Control-Allow-Origin: *
             Content-Type: application/jrd+json
             {
               "subject" : "http://ssp1.example.com",
               "links" :
               [
                 {
                   "rel" : "http://sipserviceprovider/capserver",
                   "href" : "https://capserver.ssp1.com"
                 },
                 {
                   "rel" : "http://sipserviceprovider/auth",
                   "href" : "https://ssp1.com/authorize"
                 },
                 {
                   "rel" : "http://sipserviceprovider/token",
                   "href" : "https://ssp1.com/token"
                 }
               ]
             }
```

SIP service providers MUST support the "https" URI and "acct" URI [link] schemes in WebFinger queries. The "acct" URI scheme might be used in the WebFinger query, if the enterprise adminsitrator is provided with a username by the SIP service provider. The SIP service provider might provide a unique username for each SIP trunk purchased by the enterprise network or a single username that is applicable to all the trunks purchased by the enterprise network. This draft does not require SIP service providers or enterprise networks to favor one URI scheme over the other; rather, the choice of which scheme to use is left to the discretion of the SIP service provider. The security considerations of using the "acct" URI is provided in section 5 of [RFC7565].

The base URL of the capability server returned in the WebFinger response SHOULD not contain a path or query component. Once the base

URL is obtained, the enterprise edge element builds on the base URL to identify the capability set document on the capability server. The general format for identifying resources on a capability server is as follows:

```
<scheme> /<baseURL>/<path>?<query>
```

scheme: Is always HTTPS in the context of this draft.

baseURL: The base URL of the capability server discovered as a result of the WebFinger query.

path: The path expression identifying the capability set document on the capability server. The path expression MUST be set to the static string of "capdoc".

query: A query string identifying a specific representation of the capability set document. The format of the query string is in the form of a "name=value" pair. This draft defines the following optional query parameter:

trunkid: A parameter uniquely identifying the SIP trunk for which the capability set document is sought.

The "trunkid" is useful in situations in which an enterprise SIP network has multiple SIP trunks with the SIP service provider, such that parameters for such trunks vary, perhaps because of the geographical distribution of these sites. The value of the "trunkid" parameter is generated by the SIP service provider and communicated to the enterprise SIP network by some out-of-band mechanism, for example, it may be provided in an email to the enterprise administrator after the trunk is purchased. It is RECOMMENDED that SIP service providers generate unique trunk identifiers across enterprise networks.

It is RECOMMENDED that SIP service provider networks support the "trunkid" query parameter. SIP service providers expose varying capability sets to different enterprise SIP telephony networks. Using the "trunkid" query parameter not only helps the SIP service provider capability server to uniquely identify the trunk/enterprise/edge element for which the capability set document is being request, but also, it is helpful in generating unique URL strings for capability set documents. These unique URL strings are helpful in HTTP content caching.

7. State Deltas

Given that the service provider capability set is largely expected to remain static, the work needed to implement an asynchronous push mechanism to encode minor changes in the capability set document (state deltas) is not commensurate with the benefits. Rather, enterprise edge elements can poll capability servers at pre-defined intervals to obtain the full capability set document. It is RECOMMENDED that capability servers are polled every 24 hours.

8. Encoding the Service Provider Capability Set

In the context of this draft, the capability set of a service provider refers collectively to a set of characteristics which when communicated to an enterprise network, provides it with sufficient information to directly peer with the service provider network. The capability set document is not designed to encode extremely granular details of all features, services, and protocol extensions that are supported by the service provider network. For example, it is sufficient to encode that the service provider uses T.38 relay for faxing, it is not required to know the value of the "T38FaxFillBitRemoval" parameter.

The parameters within the capability set document represent a wide array of characteristics, such that these characteristics collectively disseminate sufficient information to enable direct IP peering between enterprise and service provider networks. The various parameters represented in the capability set are chosen based on existing practises and common problem sets typically seen between enterprise and service provider SIP networks.

9. Data Model for Capability Set

This section defines a YANG module for encoding the service provider capability set. Section 9.1 provides the tree diagram, which is followed by a description of the various nodes within the module defined in this draft.

9.1. Tree Diagram

This section provides a tree diagram [RFC8340] for the "ietf-capability-set" module. The interpretation of the symbols appearing in the tree diagram is as follows:

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

o Abbreviations before data node names: "rw" means configuration (read-write), and "ro" means state data (read-only).

o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and 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.

The data model for the peering capability document has the following structure:

```
           module: ietf-sip-auto-peering
             +--rw peering-info
                +--rw variant           string
                +--rw transport-info
                |  +--rw transport?        enumeration
                |  +--rw registrar*        host-port
                |  +--rw registrarRealm?   string
                |  +--rw callControl*      host-port
                |  +--rw dns*              inet:ip-address
                |  +--rw outboundProxy?    host-port
                +--rw call-specs
                |  +--rw earlyMedia?         boolean
                |  +--rw signalingForking?   boolean
                |  +--rw supportedMethods?   string
                +--rw media
                |  +--rw mediaTypeAudio
                |  |  +--rw mediaFormat*   string
                |  +--rw fax
                |  |  +--rw protocol*   enumeration
                |  +--rw rtp
                |  |  +--rw RTPTrigger?     boolean
                |  |  +--rw symmetricRTP?   boolean
                |  +--rw rtcp
                |     +--rw symmetricRTCP?   boolean
                |     +--rw RTCPfeedback?    boolean
                +--rw dtmf
                |  +--rw payloadNumber?   int8
                |  +--rw iteration?       boolean
                +--rw security
                |  +--rw signaling
                |  |  +--rw type?      string
                |  |  +--rw version?   string
                |  +--rw mediaSecurity
                |     +--rw keyManagement?   string
                +--rw extensions?       string
```

9.2. YANG Model

This section defines the YANG module for the peering capability set document. It imports modules (ietf-yang-types and ietf-inet-types) from [RFC6991].

```
    module ietf-sip-auto-peering {
      namespace "urn:ietf:params:xml:ns:ietf-sip-auto-peering";
      prefix "peering";

      description
      "Data model for transmitting peering parameters from SP to Enterprise";

      revision 2019-05-06 {
        description "Initial revision of peering-response doc.";
      }

      import ietf-inet-types {
        prefix "inet";
      }

      typedef ipv4-address-port {
        type string {
          pattern '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
          +  '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])'
          + ':^()([1-9]|[1-5]?[0-9]{2,4}|6[1-4][0-9]{3}|65[1-4][0-9]{2}|
655[1-2][0-9]|6553[1-5])$';
        }
        description "The ipv4-address-port type represents an IPv4 address in
        dotted-quad notation followed by a port number.";
      }

      typedef ipv6-address-port {
        type string {
          pattern '((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}'
          + '((([0-9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|'
          + '(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}'
          + '(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))'
          + ':^()([1-9]|[1-5]?[0-9]{2,4}|6[1-4][0-9]{3}|65[1-4][0-9]{2}|
655[1-2][0-9]|6553[1-5])$';
          pattern
          '(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|'
          + '((([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?)'
          + ':^()([1-9]|[1-5]?[0-9]{2,4}|6[1-4][0-9]{3}|65[1-4][0-9]{2}|
655[1-2][0-9]|6553[1-5])$';
        }
          description
          "The ipv6-address type represents an IPv6 address in full,
          mixed, shortened, and shortened-mixed notation followed by a port
number.";
      }
      typedef ip-address-port {
        type union {
          type ipv4-address-port;
          type ipv6-address-port;
        }
        description
        "The ip-address-port type represents an IP address:port number
        and is IP version neutral.";
      }

      typedef domain-name-port {
        type string {
          pattern
          '((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.)*'
          + '([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.?)'
          + '|\.'
          + ':^()([1-9]|[1-5]?[0-9]{2,4}|6[1-4][0-9]{3}|65[1-4][0-9]{2}|
655[1-2][0-9]|6553[1-5])$';
          length "1..258";
        }
        description
        "The domain-name-port type represents a DNS domain name followed by a
port number.
        The name SHOULD be fully qualified whenever possible.";
      }

      typedef host-port {
        type union {
          type ip-address-port;
          type domain-name-port;
        }
        description
        "The host type represents either an IP address or a DNS
        domain name followed by a port number.";
      }

      container peering-info {
        leaf variant {
          type string;
          mandatory true;
          description "Variant of peering-response document";
        }

        container transport-info {
          leaf transport {
            type enumeration {
              enum "TCP";
              enum "TLS";
```

```
               enum "UDP";
               enum "TCP;TLS";
```

```
              enum "TCP;TLS;UDP";
              enum "TCP;UDP";
            }
            description "Transport Protocol(s) used in SIP communication";
          }

          leaf-list registrar {
            type host-port;
            max-elements 3;
            description "List of service provider registrar servers";
          }

          leaf registrarRealm {
            type string;
            description "Realm for REGISTER requests carrying credentials";
          }

          leaf-list callControl {
            type host-port;
            max-elements 3;
            description "List of service provider call control servers";
          }

          leaf-list dns {
            type inet:ip-address;
            max-elements 2;
            description "IP address of the DNS Server(s) hosted by the service
provider";
          }

          leaf outboundProxy {
            type host-port;
            description "SIP Outbound Proxy";
          }
        }

        container call-specs {
          leaf earlyMedia {
            type boolean;
            description "Flag indicating whether the service provider is
expected
            to deliver early media.";
          }

          leaf signalingForking {
            type boolean;
            description "Flag indicating whether the service provider is
capable
```

```
            of forking incoming calls ";
          }
          leaf supportedMethods {
            type string;
            description "Leaf/Leaf List indicating the different SIP methods
            support by the service provider.";
          }
        }

        container media {
          container mediaTypeAudio {
            leaf-list mediaFormat {
              type string;
              description "Leaf List indicating the audio media formats
supported.";
            }
          }

          container fax {
            leaf-list protocol {
              type enumeration {
                enum "pass-through";
                enum "t38";
              }
              max-elements 2;
              description "Leaf List indicating the different fax protocols
              supported by the service provider.";
            }
          }

          container rtp {
            leaf RTPTrigger {
              type boolean;
              description "Flag indicating whether the service provider expects
to

              receive the first media packet.";
            }

            leaf symmetricRTP {
              type boolean;
              description "Flag indicating whether the service provider expects
              symmetric RTP defined in [@RFC4961]";
            }
          }

          container rtcp {
            leaf symmetricRTCP {
              type boolean;
              description " Flag indicating whether the service provider
expects
```

```
          symmetric RTP defined in [@RFC4961].";
        }
```

```
            leaf RTCPfeedback {
              type boolean;
              description "Flag Indicating support for RTP profile extension
for
              RTCP-based feedback, as defined in [@RFC4585]";
            }
          }
        }

        container dtmf {
          leaf payloadNumber {
            type int8 {
              range "96..127";
            }
            description "Leaf that indicates the payload number(s) supported by
            the service provider for DTMF relay via Named-Telephony-Events";
          }

          leaf iteration {
            type boolean;
            description "Flag identifying whether the service provider supports
            NTE DTMF relay using the procedures of [@RFC2833] or [@RFC4733] .";
          }
        }

        container security {
          container signaling {
            leaf type {
              type string {
                pattern "TLS";
              }
              description "Type of signaling security supported.";
            }

            leaf version {
              type string {
                pattern "([1-9]\.[0-9])(;[1-9]\.[0-9])?|(NULL)";
              }
              description "Indicates TLS version for SIP signaling";
            }
          }

          container mediaSecurity {
            leaf keyManagement {
              type string {
                pattern "(SDES(;DTLS-SRTP,version=[1-9]\.[0-9](,[1-9]\.
[0-9])?)?)|(DTLS-SRTP,version=[1-9]\.[0-9](,[1-9]\.[0-9])?)|(NULL)";
              }
              description "Leaf that identifies the key management methods
              supported by the service provider for SRTP.";
```

```
            }
          }
        }

        leaf extensions {
          type string;
          description "Lists the various SIP extensions supported by SP";
        }
      }
    }
```

9.3. Node Definitions

This sub-sections provides the definition and encoding rules of the various nodes of the YANG module defined in section 9.2

o *capability-set*: This node serves as a container for all the other nodes in the YANG module; the capability-set node is akin to the root element of an XML schema.

o *variant*: This node identifies the version number of the capability set document. This draft defines the parameters for variant 1.0; future specifications might define a richer parameter set, in which case the variant can be changed to 2.0, 3.0 and so on. Future extensions to the capability set document MUST also ensure that the corresponding YANG module is defined.

o *transport-info*: The transport-info node is a container that encapsulates transport characteristics of SIP sessions between enterprise and service provider networks.

o *transport*: A leaf node that enumerates the different Transport Layer protocols supported by the SIP service provider. Valid transport layer protocols include: UDP, TCP, TLS or a combination of them (with the exception of TLS and UDP).

o *registrar*: A leaf-list that specifies the transport address of one or more registrar servers in the service provider network. The transport address of the registrar can be provided using a combination of a valid IP address and port number, or a subdomain of the SIP service provider network, or the fully qualified domain name (FQDN) of the SIP service provider network. If the transport address of a registrar is specified using either a subdomain or a fully qualified domain name, the DNS element needs to be populated with one or more valid DNS server IP addresses.

o *callControl*: A leaf-list that specifies the transport address of the call server(s) in the service provider network. The

enterprise network MUST use an applicable transport protocol in conjunction with the call control server(s) transport address when transmitting call setup requests. The transport address of a call server(s) within the service provider network can be specified using a combination of a valid IP address and port number, or a subdomain of the SIP service provider network, or a fully qualified domain name of the SIP service provider network. If the transport address of a call control server(s) is specified using either a subdomain or a fully qualified domain name, the DNS element MUST be populated with one or more valid DNS server IP addresses. The transport address specified in this element can also serve as the target for non-call requests such as SIP OPTIONS.

o *dns*: A leaf list that encodes the IP address of one or more DNS servers hosted by the SIP service provider. If the enterprise network is unaware of the IP address, port number, and transport protocol of servers within the service provider network (for example, the registrar and call control server), it MUST use DNS NAPTR and SRV. Alternatively, if the enterprise network has the fully qualified domain name of the SIP service provider network, it MUST use DNS to resolve the said FQDN to an IP address. The dns element encodes the IP address of one or more DNS servers hosted in the service provider network. If however, either the registrar or callControl elements or both are populated with a valid IP address and port pair, the dns element MUST be set to the quadruple octet of 0.0.0.0.

o *outboundProxy*: A leaf list that specifies the transport address of one or more outbound proxies. The transport address can be specified by using a combination of an IP address and a port number, a subdomain of the SIP service provider network, or a fully qualified domain name and port number of the SIP service provider network. If the outbound-proxy sub-element is populated with a valid transport address, it represents the default destination for all outbound SIP requests and therefore, the registrar and callControl elements MUST be populated with the quadruple octet of 0.0.0.0.

o *call-specs*: A container that encapsulates information about call specifications, restrictions and additional handling criteria for SIP calls between the enterprise and service provider network.

o *earlyMedia*: A leaf that specifies whether the service provider network is expected to deliver in-band announcements/tones before call connect. The P-Early-Media header field can be used to indicate pre-connect delivery of tones and announcements on a per-call basis. However, given that signalling and media could

traverse a large number of intermediaries with varying capabilities (in terms of handling of the P-Early-Media header field) within the enterprise, such devices can be appropriately configured for media cut through if it is known before-hand that early media is expected for some or all of the outbound calls. This element is a Boolean type, where a value of 1/true signifies that the service provider is capable of early media. A value of 0/false signifies that the service provider is not expected to generate early media.

- *signalingForking*: A leaf that specifies whether outbound call requests from the enterprise might be forked on the service provider network leading to multiple early dialogs. This information would be useful to the enterprise network in appropriately handling multiple early dialogs reliably and in enforcing local policy. This element is a Boolen type, where a value of 1/true signifies that the service provider network can potentially fork outbound call requests from the enterprise. A value of 0/false indicates that the service provider will not fork outbound call requests.

- *supportedMethods*: A leaf node that specifies the various SIP methods supported by the SIP service provider. The list of supported methods help to appropriately configuration various devices within the enterprise network. For example, if the service provider enumerates support for the OPTIONS method, the enterprise network could periodically send OPTIONS requests as a keep-alive mechanism.

- *media*: A container that is used to collectively encapsulate the characteristics of UDP-based audio streams. A future extension to this draft may extend the media container to describe other media types. The media container is also used to encapsulate basic information about Real-Time Transport Protocol (RTP) and Real-Time Transport Control Protocol (RTCP) from the perspective of the service provider network.

- *mediaTypeAudio*: A container for the mediaFormat leaf-list. This container collectively encapsulates the various audio media formats supported by the SIP service provider.

- *mediaFormat*: A leaf-list encoding the various audio media formats supported by the SIP service provider. The relative ordering of different media format leaf nodes from left to right indicates preference from the perspective of the service provider. Each mediaFormat node begins with the encoding name of the media format, which is the same encoding name as used in the "RTP/AVP" and "RTP/SAVP" profiles. The encoding name is followed by

required and optional parameters for the given media format as specified when the media format is registered [RFC4855]. Given that the parameters of media formats can vary from one communication session to another, for example, across two separate communication sessions, the packetization time (ptime) used for the PCMU media format might vary from 10 to 30 ms, the parameters included in the format element MUST be the ones that are expected to be invariant from the perspective of the service provider. Providing information about supported media formats and their respective parameters, allows enterprise networks to configure the media plane characteristics of various devices such as endpoints and middleboxes. The encoding name, one or more required parameters, one or more optional parameters are all separated by a semicolon. The formatting of a given media format parameter, MUST follow the formatting rules as specified for that media format.

o *fax*: A container that encapsulates the fax protocol(s) supported by the SIP service provider. The fax container encloses a leaf-list (named protocol) that enumerates whether the service provider supports t38 relay, protocol-based fax passthrough or both. The relative ordering of leaf nodes within the leaf lists indicates preference.

o *rtp*: A container that encapsulates generic characteristics of RTP sessions between the enterprise and service provider network. This node is a container for the "RTPTrigger" and "SymmetricRTP" leaf nodes.

o *RTPTrigger*: A leaf node indicating whether the SIP service provider network always expects the enterprise network to send the first RTP packet for an established communication session. This information is useful in scenarios such as "hairpinned" calls, in which the caller and callee are on the service provider network and because of sub-optimal media routing, an enterprise device such as an SBC is retained in the media path. Based on the encoding of this node, it is possible to configure enterprise devices such as SBCs to start streaming media (possibly filled with silence payloads) toward the address:port tuples provided by caller and callee. This node is a Boolean type. A value of 1/true indicates that the service provider expects the enterprise network to send the first RTP packet, whereas a value of 0/false indicates that the service provider network does not require the enterprise network to send the first media packet. While the practise of preserving the enterprise network in a hairpinned call flow is fairly common, it is RECOMMENDED that SIP service providers avoid this practise. In the context of a hairpinned call, the enterprise device retained in the call flow can easily eavesdrop on the conversation between the offnet parties.

- *symmetricRTP*: A leaf node indicating whether the SIP service provider expects the enterprise network to use symmetric RTP as defined in [RFC4961]. Uncovering this expectation is useful in scenarios where "latching" [RFC3762] is implemented in the service provider network. This node is a Boolean type, a value of 1/true indicates that the service provider expects the enterprise network to use symmetric RTP, whereas a value of 0/false indicates that the enterprise network can use asymmetric RTP.

- *rtcp*: A container that encapsulates generic characteristics of RTCP sessions between the enterprise and service provider network. This node is a container for the "RTCPFeedback" and "SymmetricRTCP" leaf nodes.

- *RTCPFeedback*: A leaf node that indicates whether the SIP service provider supports the RTP profile extension for RTCP-based feedback [RFC4585]. Media sessions spanning enterprise and service provider networks, are rarely made to flow directly between the caller and callee, rather, it is often the case that media traffic flows through network intermediaries such as SBCs. As a result, RTCP traffic from the service provider network is intercepted by these intermediaries, which in turn can either pass across RTCP traffic unmodified or modify RTCP traffic before it is forwarded to the endpoint in the enterprise network. Modification of RTCP traffic would be required, for example, if the intermediary has performed media payload transformation operations such as transcoding or transrating. In a similar vein, for the RTCP-based feedback mechanism as defined in [RFC4585] to be truly effective, intermediaries MUST ensure that feedback messages are passed reliably and with the correct formatting to enterprise endpoints. This might require additional configuration and considerations that need to be dealt with at the time of provisioning the intermediary device. This node is a Boolean type, a value of 1/true indicates that the service provider supports the RTP profile extension for RTP-based feedback and a value of 0/false indicates that the service provider does not support the RTP profile extension for RTP-based feedback.

- *symmetricRTCP*: A leaf node indicating whether the SIP service provider expects the enterprise network to use symmetric RTCP as defined in [RFC4961]. This node is a Boolean type, a value of 1 indicates that the service provider expects symmetric RTCP reports, whereas a value of 0 indicates that the enterprise can use asymmetric RTCP.

- *dtmf*: A container that describes the various aspects of DTMF relay via RTP Named Telephony Events. The dtmf container allows

SIP service providers to specify two facets of DTMF relay via Named Telephony Events:

1. The payload type number using the payloadNumber leaf node.
2. Support for [RFC2833] or [RFC4733] using the iteration leaf node.

In the context of named telephony events, senders and receivers may negotiate asymmetric payload type numbers. For example, the sender might advertise payload type number 97 and the receiver might advertise payload type number 101. In such instances, it is either required for middleboxes to interwork payload type numbers or allow the endpoints to send and receive asymmetric payload numbers. The behaviour of middleboxes in this context is largely dependent on endpoint capabilities or on service provider constraints. Therefore, the payloadNumber leaf node can be used to determine middlebox configuration before-hand.

[RFC4733]iterates over [RFC2833] by introducing certain changes in the way NTE events are transmitted. SIP service providers can indicate support for [RFC4733] by setting the iteration flag to 1 or indicating support for [RFC2833] by setting the iteration flag to 0.

- *security*: A container that encapsulates characteristics about encrypting signalling streams between the enterprise and SIP service provider networks.

- *signaling*: A container that encapsulates the type of security protocol for the SIP communication between the enterprise SBC and the service provider.

- *type*: A leaf node that specifies the protocol used for protecting SIP signalling messages between the enterprise and service provider network. The value of the type leaf node is only defined for Transport Layer Security (TLS). Accordingly, if TLS is allowed for SIP sessions between the enterprise and service provider network, the type leaf node is set to the string "tls".

- *version*: A leaf node that specifies the version(s) of TLS supported in decimal format. If multiple versions of TLS are supported, they MUST be separated by semi-colons. If the service provide does not support TLS for protecting SIP sessions, the signalling element is set to the string "NULL".

- *mediaSecurity*: A container that describes the various characteristics of securing media streams between enterprise and service provider networks.

- *keyManagement*: A leaf node that specifies the key management method used by the service provider. Possible values of this node include: "SDES" and "DTLS-SRTP". A value of "SDES" signifies that the SIP service provider uses the methods defined in [RFC4568] for the purpose of key management. A value of "DTLS-SRTP" signifies that the SIP service provider uses the methods defined in [RFC5764] for the purpose of key management. If the value of this leaf node is set to "DTLS-SRTP", the various versions of DTLS supported by the SIP service provider MUST be encoded as per the formatting rules of Section 9.2. If the service provider does not support media security, the keyManagement node MUST be set to "NULL".

- *extensions*: A leaf node that is a semicolon separated list of all possible SIP option tags supported by the service provider network. These extensions MUST be referenced using name registered under IANA. If the service provider network does not support any extensions to baseline SIP, the extensions node MUST be set to "NULL".

9.4. Extending the Capability Set

There are situations in which equipment manufactures or service providers would benefit from extending the YANG module defined in this draft. For example, service providers could extend the YANG module to include information that further simplifies direct IP peering. Such information could include: trunk group identifiers, direct-inward-dial (DID) number ranges allocated to the enterprise, customer/enterprise account numbers, service provider support numbers, among others.

Extension of the module can be achieved by importing the module defined in this draft. An example is provided below:

Consider a new YANG module "vendorA" specified for VendorA's enterprise SBC. The "vendorA-config" YANG module is configured as follows:

```
    module vendorA-config {
      namespace "urn:ietf:params:xml:ns:yang:vendorA-config";
      prefix "vendorA";

      description
      "Data model for configuring VendorA Enterprise SBC";

      revision 2020-05-06 {
      description "Initial revision of VendorA Enterprise SBC configuration
data model";
      }

      import ietf-peering {
        prefix "peering";
      }

      augment "/peering:peering-info" {
        container vendorAConfig {
          leaf vendorAConfigParam1 {
            type int32;
            description "vendorA configuration parameter 1 (SBC Device ID)";
          }

          leaf vendorAConfigParam2 {
            type string;
              description "vendorA configuration parameter 2 (SBC Device
name)";
          }
          description "Container for vendorA SBC configuration";
        }
      }
    }
```

In the example above, a custom module named "vendorA-config" uses the "augment" statement as defined in Section 4.2.8 of [RFC7950] to extend the module defined in this draft.

10. Example Capability Set Document Encoding

This section provides examples of how capability set documents that leverage the YANG module defined in this document can be encoded over JSON or XML.

10.1. JSON Capability Set Document

```
    {
      "peering-info:variant": "1.0",
      "transport-info": {
        "transport": "TCP;TLS;UDP",
        "registrar": ["registrar1.voip.example.com:5060",
"registrar2.voip.example.com:5060"],
        "registrarRealm": "voip.example.com",
        "callControl": ["callServer1.voip.example.com:5060",
"192.168.12.25:5065"],
        "dns": [8.8.8.8, 208.67.222.222],
        "outboundProxy": "0.0.0.0"
      },
      "call-specs": {
        "earlyMedia": "true",
        "signalingForking": "false",
        "supportedMethods":
"INVITE;OPTIONS;BYE;CANCEL;ACK;PRACK;SUBSCRIBE;NOTIFY;REGISTER"
      },
      "media": {
        "mediaTypeAudio": {
          "mediaFormat":
["PCMU;rate=8000;ptime=20","G729;rate=8000;annexb=yes","G722;rate=8000;bitrate=56k,
64k"]
        },
        "fax": {
          "protocol": ["pass-through", "t38"]
        },
        "rtp": {
          "RTPTrigger": "false",
          "symmetricRTP": "true"
        },
        "rtcp": {
          "symmetricRTCP": "true",
          "RTCPFeedback": "true"
        }
      },
      "dtmf": {
        "payloadNumber": "101",
        "iteration": "0"
      },
      "security": {
        "signaling": {
          "type": "TLS",
          "version": "1.0;1.2"
        },
        "mediaSecurity": {
          "keyManagement": "SDES;DTLS-SRTP,version=1.2"
        }
      },
      "extensions": "timer;rel100;gin;path"
    }
```

10.2. XML Capability Set Document

```
    <peering-info xmlns="urn:ietf:params:xml:ns:yang:ietf-peering"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:yang:ietf-peering ietf-
peering.xsd">
      <variant>1.0</variant>
      <transport-info>
        <transport>TCP;TLS;UDP</transport>
        <registrar>registrar1.voip.example.com:5060</registrar>
        <registrar>registrar2.voip.example.com:5060</registrar>
        <registrarRealm>voip.example.com</registrarRealm>
        <callControl>callServer1.voip.example.com:5060</callControl>
        <callControl>192.168.12.25:5065</callControl>
        <dns>8.8.8.8</dns>
        <dns>208.67.222.222</dns>
        <outboundProxy>0.0.0.0</outboundProxy>
      </transport-info>
      <call-specs>
        <earlyMedia>true</earlyMedia>
        <signalingForking>false</signalingForking>

<supportedMethods>INVITE;OPTIONS;BYE;CANCEL;ACK;PRACK;SUBSCRIBE;NOTIFY;REGISTER</
supportedMethods>
      </call-specs>
      <media>
        <mediaTypeAudio>
          <mediaFormat>PCMU;rate=8000;ptime=20</mediaFormat>
          <mediaFormat> G729;rate=8000;annexb=yes</mediaFormat>
          <mediaFormat>G722;rate=8000;bitrate=56k,64k</mediaFormat>
        </mediaTypeAudio>
        <fax>
          <protocol>pass-through</protocol>
          <protocol>t38</protocol>
        </fax>
        <rtp>
          <RTPTrigger>true</RTPTrigger>
          <symmetricRTP>true</symmetricRTP>
        </rtp>
        <rtcp>
          <symmetricRTCP>true</symmetricRTCP>
          <RTCPFeedback>true</RTCPFeedback>
        </rtcp>
      </media>
      <dtmf>
        <payloadNumber>101</payloadNumber>
        <iteration>0</iteration>
      </dtmf>
      <security>
        <signaling>
          <type>TLS</type>
          <version>1.0;1.2</version>
        </signaling>
        <mediaSecurity>
          <keyManagement>SDES;DTLS-SRTP,version=1.2</keyManagement>
          </mediaSecurity>
        </security>
      <extensions>timer;rel100;gin;path</extensions>
    </peering-response>
```

11. Example Exchange

This section depicts an example of the configuration flow that ultimately results in the enterprise edge element obtaining the capability set document from the SIP service provider.

Assuming the enterprise edge element isn't pre-configured with the request target for the capability set document and is required to authenticate with the SIP service provider capability server, the following sequence of events are put into motion to obtain the capability set document:

The enterprise edge element generates a WebFinger query to discover endpoints hosted in the ssp1.example.com domain (line wraps are for display purposes only)

```
    GET /.well-known/webfinger?
      resource=http%3A%2F%2Fssp1.example.com
      &rel=http%3A%2f%2fsipserviceprovider%2fcapserver
      &rel=http%3A%2f%2fsipserviceprovider%2auth
      &rel= http%3A%2f%2fsipserviceprovider%2token
      HTTP/1.1
    Host: ssp1.example.com
```

The resulting WebFinger response, contains the URLs of the capability server, the OAuth 2.0 authorization endpoint and OAuth 2.0 token endpoint.

```
           HTTP/1.1 200 OK
           Access-Control-Allow-Origin: *
           Content-Type: application/jrd+json
           {
             "subject" : "http://ssp1.example.com",
             "links" :
             [
               {
                 "rel" : "http://sipserviceprovider/capserver",
                 "href" : "https://capserver.ssp1.com"
               },
               {
                 "rel" : "http://sipserviceprovider/auth",
                 "href" : "https://ssp1.com/authorize"
               },
               {
                 "rel" : "http://sipserviceprovider/token",
                 "href" : "https://ssp1.com/token"
               },
             ]
           }
```

The endpoint URLs returned in the WebFinger response are stored by the edge element and referenced when required. Then, the administrator logs into the GUI of the edge element and initiates the download of the service provider capability set (perhaps by clicking on a button). This triggers the edge element to redirect the administrator to the OAuth 2.0 authorization endpoint (discovered via WebFinger). Once the administrator is authenticated and provides authorization, flow is redirected to the callback URL of the edge element application. The edge element then contacts the OAuth 2.0 token endpoint (discovered via WebFinger) to authenticate itself and obtain access and refresh tokens. Accordingly, the edge element mints a JWT bearer token to authenticate itself with the token endpoint and obtain an access and refresh token. Below is an example of client authentication using a JWT during the presentation of an authorization code grant for an access token request (line wraps are for display purposes only).

```
POST /token  HTTP/1.1
Host: ssp1.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=n0esc3NRze7LTCu7iYzS6a5acc3f0ogp4&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJhbGciOiJSUzI1NiIsImtpZCI6IjIyIn0.
eyJpc3Mi[...omitted for brevity...].
cC4hiUPo[...omitted for brevity...]
```

If the request is acceptable to the token endpoint, an access token and a refresh token is provided in the response. For example:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"sF_9.B5f-4.1JqM",
  "token_type":"Bearer",
  "expires_in":86400,
  "refresh_token":"hGzv3JOkF0XG5Qx2TlKWIA"
}
```

The obtained bearer access token can subsequently be used to obtain the capability set document for the capability server. The edge element generates a HTTPS GET request with the bearer token included in the Authorization header field.

```
GET //capdoc?trunkid=trunkent1456 HTTP/1.1
Host: capserver.ssp1.com
Authorization: Bearer mF_9.B5f-4.1JqM
Accept:application/peering-info+xml
```

The capability set document is obtained in the body of the response and is encoded in XML.

```
HTTP/1.1 200 OK
Content-Type: application/peering-info+xml
Content-Length: nnn

<peering-info>
...
</peering-info>
```

12. Security Considerations

Capability set documents have a significant bearing on the quality of the peering relationship between an enterprise and service provider network. These documents can be modified by an attacker to drastically impact the quality of communication sessions between enterprise and service provider networks. Additionally, capability set documents contain parameters that may be considered sensitive from the perspective of the SIP service provider. For example, the YANG model defined in this document might be extended by SIP service providers to include account sensitive information such as the username and password to used when generating an MD5 response to 401/407 SIP challenges.

To ensure the problems discussed in the previous paragraph are accounted for, the following considerations MUST be taken into account:

o Integrity and Confidentiality

Request and responses for the capability set documents are defined over HTTP. However, due to the sensitive nature of information transmitted between client and server, it is required to secure HTTP using Transport Layer Security. The enterprise edge element and capability server MUST be compliant to [RFC7235]. The enterprise edge element and capability server MUST support the use of the https uri scheme as defined in [RFC7230].

o Authenticated Client Identity

While this draft does not enforce client authentication, there are situations in which client need to authenticated by SIP service providers before they are provided capability set documents. In such situations client MUST be authenticated using the procedures outlined in section 6.3 of this draft.

o The "trunkid" parameter

It is RECOMMENDED that enterprise edge elements use the "trunkid" parameter in query strings when requesting for the capability set documents. The value of "trunkid" parameter is generated by the SIP service provider and provided to the administrator via some out-of-band mechanism. SIP service providers MUST ensure that value of the "trunkid" parameters does not inadvertently communicate sensitive information to an attacker such as a username or password credential.

In addition to the considerations listed above, all the security considerations that are part of the WebFinger and OAuth 2.0 specifications are applicable to this draft.

13. Acknowledgments

TBD

14. References

14.1. 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>.

[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>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

14.2. Informative References

[RFC2833] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, DOI 10.17487/RFC2833, May 2000, <https://www.rfc-editor.org/info/rfc2833>.

[RFC3762] Levin, O., "Telephone Number Mapping (ENUM) Service Registration for H.323", RFC 3762, DOI 10.17487/RFC3762, April 2004, <https://www.rfc-editor.org/info/rfc3762>.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <https://www.rfc-editor.org/info/rfc4568>.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <https://www.rfc-editor.org/info/rfc4585>.

[RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals", RFC 4733, DOI 10.17487/RFC4733, December 2006, <https://www.rfc-editor.org/info/rfc4733>.

[RFC4855] Casner, S., "Media Type Registration of RTP Payload Formats", RFC 4855, DOI 10.17487/RFC4855, February 2007, <https://www.rfc-editor.org/info/rfc4855>.

[RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, DOI 10.17487/RFC4961, July 2007, <https://www.rfc-editor.org/info/rfc4961>.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <https://www.rfc-editor.org/info/rfc5764>.

[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>.

[RFC6665] Roach, A., "SIP-Specific Event Notification", RFC 6665, DOI 10.17487/RFC6665, July 2012, <https://www.rfc-editor.org/info/rfc6665>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, DOI 10.17487/RFC7033, September 2013, <https://www.rfc-editor.org/info/rfc7033>.

[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>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[RFC7565] Saint-Andre, P., "The 'acct' URI Scheme", RFC 7565, DOI 10.17487/RFC7565, May 2015, <https://www.rfc-editor.org/info/rfc7565>.

[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>.

[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>.

Authors' Addresses

Kaustubh Inamdar
Cisco Systems

Email: kinamdar@cisco.com

Sreekanth Narayanan
Cisco Systems

Email: sreenara@cisco.com

Cullen Jennings
Cisco Systems

Email: fluffy@iii.ca