Workgroup: ASAP
Published: 24 October 2021
Intended Status: Standards Track
Expires: 27 April 2022
Authors: K. Inamdar  S. Narayanan  C. Jennings
Unaffiliated  Cisco Systems  Cisco Systems

# Automatic Peering for SIP Trunks

## 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

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 27 April 2022.

**Table of Contents**


1. Introduction
2. Overview of Operations
   2.1. Reference Architecture
   2.2. Configuration Workflow
   2.3. Transport
3. Conventions and Terminology
4. HTTP Transport
   4.1. HTTP Methods
   4.2. Integrity and Confidentiality
   4.3. Authenticated Client Identity
   4.4. Encoding the Request
   4.5. Identifying the Request Target
   4.6. Generating the response
5. State Deltas
6. Encoding the Service Provider Capability Set
7. Data Model for Capability Set
   7.1. Tree Diagram
   7.2. YANG Model
   7.3. Node Definitions
   7.4. Extending the Capability Set
8. Processing the Capability Set Response
9. Examples
   9.1. JSON Capability Set Document
   9.2. Example Exchange
10. Security Considerations
11. Acknowledgments
12. Normative References

Authors' Addresses


## 1. Introduction

The deployment of a Session Initiation Protocol [RFC3261] (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 [SIP-Connect-TR] 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. However, this workflow complements the SIP Connect technical recommendations, in that both endeavours aim to promote/ achieve interop between the enterprise and service provider.

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. Overview of Operations

This section details the configuration workflow proposed by this draft.

### 2.1. 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 [RFC7092]. 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 HTTPS [RFC7231] server.

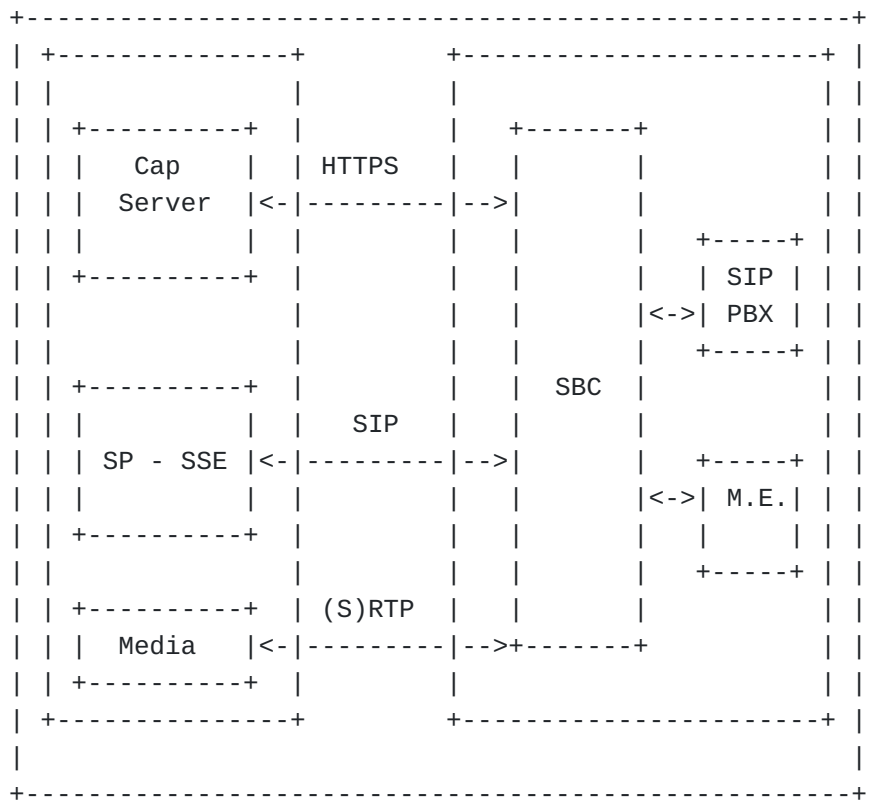


Figure 1: Reference Architecture

This draft makes use of the following terminology:

*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.

*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) [RFC7092] such as a Session Border Controller (SBC).

*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 HTTPS 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.

*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: The term capability set (or capability set document) refers collectively to a set of characteristics within the service provider network, which when communicated to the enterprise network, provides the enterprise network the information required to interconnect with the service provider network. The various parameters that constitute the capability set relate to characteristics that are specific to signalling, media, transport and security. Certain aspects of interconnecting with service providers are out of scope of the capability set. For example, the access technology used to interconnect with service provider networks.

## 2.2. 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 barrier to adoption for SIP service providers and enterprise networks as equipment manufacturers would have to first add support for such a SIP extension. A HTTPS-based approach would be relatively easier to adopt as most edge devices deployed in enterprise networks today already support HTTPS; from the perspective of service provider networks, all that is required is for them to deploy HTTPS servers that function as capability servers. Additionally, most SIP service providers require enterprise networks to register with them (using a SIP REGISTER message) before any other SIP methods that initiate subscriptions (SIP SUBSCRIBE) or calls (SIP INVITE) are processed. As a result, a SIP-based framework to obtain a capability set would require operational changes on the part of service provider networks.

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 centralised 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.

### 2.3. Transport

To solicit the capability set of a SIP service provider, the edge element in an 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 normalise 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 interconnection.

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 4.

There could be situations wherein an enterprise telephony network interconnects with its SIP service provider such that traffic between the two networks traverses an intermediary SIP service provider network. This could be a result of interconnect agreements between the terminating and transit SIP service provider networks. In such situations, the capability set provided to the enterprise network by its SIP service provider must account for the characteristics of the transit SIP service provider network from a signalling and media perspective. For example, if the terminating SIP service provider network supports the G.729 codec and the transit SIP service provider network does not, G.729 must not be advertised in the capability set. As another example, if the transit SIP service provider network doesn't support a SIP extension, for instance, the SIP extension for Reliable Provisional Responses as defined in RFC 3262, the terminating SIP service provider network must not advertise support for this extension in the capability set provided to the enterprise network. How a terminating SIP service provider obtains the characteristics of the intermediary SIP service provider network is out of the scope of this document; however, one method could be for the terminating SIP service provider to obtain the characteristics of the intermediary SIP service provider by leveraging the YANG model introduced in this document.

## 3. 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 [BCP-14]

## 4. HTTP Transport

This section describes the use of HTTPS as a transport protocol for the peering workflow. This workflow is based on HTTP version 1.1, and as such is compatible with any future version of HTTP that is backward compatible with HTTP 1.1.

### 4.1. HTTP Methods

The workflow defined in this document leverages the HTTPS GET method and its corresponding response(s) to request for and subsequently obtain the service provider capability set document.

### 4.2. Integrity and Confidentiality

Peering requests and responses are defined over HTTPS. However, due to the sensitive nature of information transmitted between client and server, it is required to secure HTTP using Transport Layer Security [RFC5246]. The enterprise edge element and capability server MUST be compliant with [RFC7235]. The enterprise edge element and capability server MUST support the use of the HTTPS uri scheme as defined in [RFC7230].

### 4.3. Authenticated Client Identity

HTTP usually adopts asymmetric methods of authentication. For example, clients typically use certificate based authentication to verify the server they are talking to, whereas, servers typically use methods such as HTTP digest authentication or OAuth2.0 to authenticate clients. Though OAuth2.0 is not an authentication protocol, it nonetheless allows for client authentication to be carried out with the use of OAuth tokens.

Figure 2 elucidates the use of this grant type.

In the context of the SIP Auto Peer framework, OAuth2.0 MUST be used to carry out client authentication. Enterprise edge elements that obtain the capability set document from SIP service providers could have differing capabilities in terms of adhering to a specific OAuth2.0 authorisation grant flow. For example, an SBC that is configured and managed through a CLI and that does not have the ability to launch a web-browser wouldn't be able to obtain an authorisation code and subsequently an access token. Alternatively, an SBC that is configured and managed via a GUI could redirect an administrator to an appropriate OAuth2.0 authorisation server to obtain an authorisation grant and subsequently an access token. In order to ensure that OAuth2.0-based client authentication can be carried out irrespective of enterprise edge element capabilities, this draft requires that the Resource Owner Password Credentials grant type be supported.

Using the resource owner password credentials grant type requires the existence of a trust relationship between the resource owner(in this context, the administrator/enterprise network) and the client(in this context, an edge element such as an SBC). In SIP trunking deployments between enterprise and service provider networks, such a trust relationship between the administrator/ resource owner/enterprise network and the client(edge element) already exists, as SIP trunk registration (and refreshing registrations) require credentials - typically a username and password, that are configured on the edge element by the administrator.

The use of the resource owner credential grant type in the context of the SIP Auto Peer framework, provides two advantages:

1. It enables OAuth2.0-based client authentication even in deployments in wherein the edge element is not capable of launching a web-browser to set in motion the authorisation code grant flow of OAuth2.0

2. For situations in which a refresh token is not provided by the authorisation endpoint, human/administrator involvement is not required to obtain fresh tokens once an existing token expires. Figure 2 provides a high-level diagrammatic illustration of how OAuth2.0-based client authentication is achieved using resource owner credentials in the context of SIP Auto Peer.

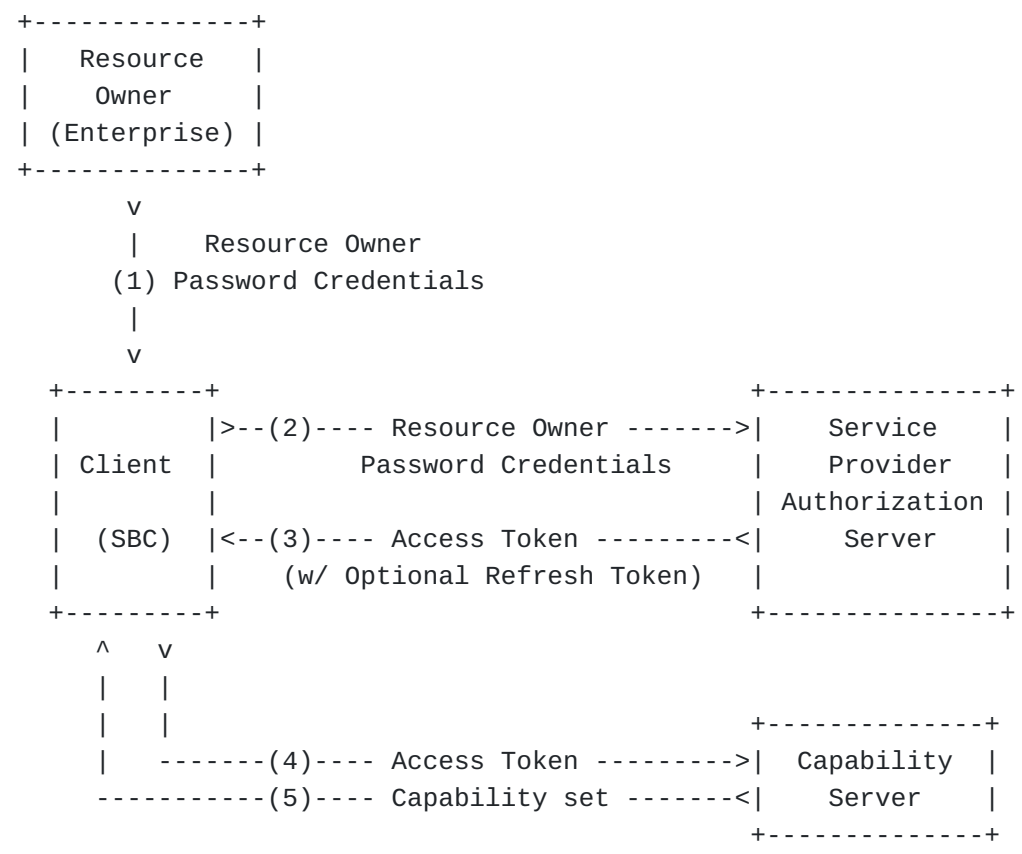


Figure 2: Client Authentication Mechanism

The flow illustrated in Figure 2 includes the following steps:

1. The enterprise SBC stores the enterprise credentials required to authenticate with the authorization server located in the service provider network. These credentials may be passed to the enterprise from the service provider in an out-of-band fashion such as an email or a self-management service provided by the service provider to the enterprise.

2. The enterprise SBC contacts the service provider authorization server to obtain an access token using the credentials acquired in Step 1.

3. The service provider authorization server ratifies the credentials and grants the access token to the enterprise SBC. The server could also provide a refresh token to the SBC to regenerate the access token in the future.

4. The enterprise SBC then contacts the capability server located in the service provider network with an HTTPS GET request along with the access token to retrieve the capability set document.

5. The capability server checks for a valid access token and returns the capability set document to the enterprise SBC.

### 4.4. 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.

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

### 4.5. 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. Discovery 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.

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=capabilitySet
    HTTP/1.1
Host: ssp1.example.com


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

Once the target URI is obtained by an enterprise telephony network, the URI may be dereferenced to obtain a unique capability set document that is specific to that given enterprise telephony network. The ITSP may use credentials to determine the identity of the enterprise telephony network and provide the appropriate capability set document.

### 4.6. 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 must 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.

## 5. 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.

## 6. 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.

## 7. 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.

### 7.1. 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:

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

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

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

* Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

* 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 revision
     |  +--rw notBefore?   string
     |  +--rw location?    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 callerId
     |  |  +--rw e164Format?        boolean
     |  |  +--rw preferredMethod?   string
     |  +--rw numRange
     |     +--rw numRangeType?   string
     |     +--rw count?          int32
     |     +--rw value*          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 certLocation?              string
     |  +--rw secureTelephonyIdentity
     |     +--rw STIRCompliance?   boolean
     |     +--rw certDelegation?   boolean
```

```
|     +--rw ACMEDirectory?    string
+--rw extensions?       string
```

### 7.2. 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 revision {
      leaf notBefore {
          type string;
          description "Time and date of activation of new
        capability set";
      }

      leaf location {
```

```
        type string;
        description "Location of the new version of
        capability set 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 callerId {
    leaf e164Format {
      type boolean;
      description "Flag indicating whether enterprise must
      format caller information into E.164";
    }

    leaf preferredMethod {
      type string;
      description "Field that instructs enterprise regarding
      which SIP header it must populate to communicate caller
      information.";
    }
  }

  container numRange {
    leaf numRangeType {
      type string;
      description "String indicating whether the DID number
      range is passed by value or by reference";
    }

    leaf count {
      when "../numRangeType = 'range' or
      ../numRangeType = 'block'";
      type int32;
      description "Number of DID numbers present in the number
```

```
      range.";
    }

    leaf-list value {
      type string;
      description "Value of the DID number range or URL being
      passed as reference.";
    }

  }
}

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 certLocation {
    type string;
    description "Location of the service provider certificate
    chain for SIP over TLS.";
  }

  container secureTelephonyIdentity {
    leaf STIRCompliance {
      type boolean;
      description "Indicates whether the SIP service provider
      is STIR compliant.";
    }

    leaf certDelegation {
      type boolean;
      description "Indicates whether a SIP service provider is
      willing to delegate authority to the enterprise network
      over its allocated number range(s)";
    }

    leaf ACMEDirectory {
      when "../certDelegation = 1
                  or ../certDelegation = 'true'";
      type string;
      description "Directory object URL, when de-referenced,
      provides a collection of field name-value pairs to
      kickstart ACME.";
    }
  }
}

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

```
    }
}
```

### 7.3. Node Definitions

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

**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 json document.

**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 must 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.

**revision**: The revision node is a container that encapsulates information regarding the availability of a new version of the capability set document for the enterprise.

**notBefore**: The notBefore node indicates the date and time at which the new capabilities go live in the service provider network.

**location**: This leaf node value provides the URL of the new revision of the capability set document

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

**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).

**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 must be populated with one or more valid DNS server IP addresses.

**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.

**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

**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

**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.

**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 that MAY lead 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 Boolean 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.

**callerId**: This is a container that encodes the preferences of SIP Service Providers in terms calling number presentation by the enterprise network. Certain ITSPs require that the calling number be formatted in E.164, whereas others place no such restrictions. Additionally, some ITSPs require that the calling number be included in a specific SIP header field, for example, the P-Asserted-ID header field or the P-Asserted-ID field, whereas others place no restrictions on the specific SIP header field used to convey the calling number.

**e164Format**: A leaf node that indicates whether the service provider requires enterprise to normalize the caller number into the E.164 format while communicating caller details. This node is of the boolean type. A value of 'true' or '1' mandates the enterprise format caller numbers into the E.164 format, while a 'false' or '0' leaves the formatting of the caller number up to the enterprise.

**preferredMethod**: A leaf node that instructs the enterprise regarding which SIP header to populate the caller information into when communicating caller ID information. This node will contain the name of the SIP Header.

**numRange**: Is a container that specifies the Direct Inward Dial (DID) number range allocated to the enterprise network by the SIP service provider. The DID number range allocated by the service provider to the enterprise network might be a contiguous or a non-contiguous block. The number range allocated to an enterprise can be communicated as a value or as a reference. For large enterprise networks, the size of the DID range might run into several hundred

numbers. For situations in which the enterprise is allocated a large DID number range or a non-contiguous number range it is RECOMMENDED that the SIP service provider communicate this information by reference, that is, through a URL. The enterprise network is required to de-reference this URL in order to obtain the DID number range allocated by the SIP service provider. The numRange container can be used more than once. Refer to the example provided in Section 10.1.

**numRangeType**: A leaf node that indicates whether the DID range is communicated by value or by reference. It can have a value of 'range', 'block' or 'reference'.

**count**: A leaf node that indicates the size of the DID number range. The number range may be contiguous or non-contiguous. This leaf node MUST NOT be included when using the 'reference' numRangeType value.

**value**: A leaf-list that encapsulates the DID number range allocated to the enterprise. If the numRangeType value is set to 'range' or 'block', this is the list of numbers allocated to the enterprise. If the numRangeType value is set to 'reference', this is the URL of the resource containing the DID number range. To ensure ease of parsing, it is RECOMMENDED that the resource contain a number range formatted as if it were being passed as a block or range.

**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. As of the date of writing this draft, video media streams aren't exchanged between enterprise and service provider SIP networks.

**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.

**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.

**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.

**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" [RFC7362] 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 should be separated by semi-colons. If the service provider 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".

**certLocation:**: If the enterprise network is required to exchange SIP traffic over TLS with the SIP service provider, and if the SIP service provider is capable of accepting TLS connections from the enterprise network, it may be required for the SIP service provider certificates to be pre-installed on the enterprise edge element. In such situations, the certLocation leaf node is populated with a URL, which when dereferenced, provides a single PEM encoded file that contains all certificates in the chain of trust. This is an optional leaf node.

**secureTelephonyIdentity**: A container that is used to collectively encapsulate Secure Telephony Identity Revisited (STIR) characteristics.

**STIRCompliance**: A leaf node that indicates whether the SIP service provider is STIR compliant. This node is a Boolean type, a value 1/ true indicates that the SIP service provider is STIR compliant. A value of 0/false indicates that the SIP service provider is not STIR compliant. A SIP service provider being STIR compliant has implications for inbound and outbound calls, from the perspective of the enterprise network.

For inbound calls received from a STIR compliant SIP service provider, the enterprise edge element can be configured to appropriately handle calls based on their "attestation value". For example, calls with an attestation value of "A" (Full Attestation) are allowed to go through, while calls with an attestation value of "C" (Gateway Attestation) may be flagged for administrative analysis.

For outgoing calls placed to a STIR compliant SIP service provider, the enterprise edge element must ensure that the calling number populated in SIP From header field (or in trusted environments, the P-Asserted-Identity header field), is as per what the service provider expects. This is so that the Authentication Service running in the SIP service provider network can determine if it is authoritative for the calling number presented by the enterprise network.

**certDelegation**: A leaf node value that indicates whether a SIP service provider that allocates one or more number ranges to an enterprise network, is willing to delegate authority to the enterprise network over that number range(s). This node is a Boolean

type, a value of 1/true indicates that the SIP service provider is willing to delegate authority to the enterprise network over one or more number ranges. A value of 0/false indicates that the SIP service provider is not willing to delegate authority to the enterprise network over one or more number ranges. This leaf node MUST only be included in the capability set if the value of the STIRCompliance leaf node is set to 1/true. In order to obtain delegate certificates, the enterprise network must be made aware of the scope of delegation - the number or number range(s) over which the SIP service provider is willing to delegate authority. This information is included in the numRange container.

**ACMEDirectory**: For delegate certificates that are obtained by the enterprise network using Automatic Certificate Management Environment (ACME), this leaf node value provides the URL of the directory object [ACME]. The directory object URL, when de-referenced, provides a collection of field name-value pairs. Certain field name-value pairs provided in the response are used to bootstrap the process the obtaining delegate certificates. This leaf node MUST only be included in the capability set if the value of the certDelegation leaf node is set to 1/true.

**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".

### 7.4. 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.

## 8. Processing the Capability Set Response

This section provides a non-normative description of the procedures that could be carried out by the enterprise network after obtaining the SIP service provider capability set. On obtaining the capability set, the enterprise edge element can parse the various fields within the capability set and generate configuration blocks. For example, the configuration required to successfully register a SIP trunk with the SIP registrar hosted in the service provider network, the configuration required to ensure that fax calls are handled appropriately, the configuration required to advertise only audio codecs supported by the SIP service provider, among many other

configuration blocks. A configuration block generated for an almost identical SIP service provider capability set document is likely going to differ drastically from one vendor to the next.

Enterprise edge elements are usually capable of normalising mismatches in the signalling and media planes between the enterprise and service provider SIP networks. As a result, most, if not all of the configuration blocks required to enable successful SIP service provider peering might need to be added on the edge element. In situations wherein configuration blocks need to be distributed across multiple devices, some mechanism, that is out of scope of this document might be used to communicate the specific fields of capacity set and their corresponding value. Alternatively, a human administrator could go through the capability set document and configure the edge element (and if required, other devices in the enterprise network appropriately.

## 9. Examples

This section provides examples of how capability set documents that leverage the YANG module defined in this document can be encoded over JSON as well as the exchange of messages between the enterprise edge element and the service provider to acquire the capability set document.

### 9.1. JSON Capability Set Document

```
{
    "peering-info": {
        "variant": "1.0",
        "revision": {
            "notBefore": "2021-10-16T00:00:00.00000Z",
            "location":
            "https://capserver.ssp1.com/capserver/capdoc.json",
        },
        "transport-info": {
            "transport": "TCP;TLS;UDP",
            "registrar": ["registrar1.voip.example.com:5060",
             "registrar2.voip.example.com:5060"],
            "registerRealm": "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",
            "callerId": {
                "e164Format": "true",
                "preferredMethod": "P-Asserted-Identity"
            },
            "numRange": {
                "type": "range",
                "count": "20",
                "value": "19725455000"
            },
            "numRange": {
                "type": "block",
                "count": "2",
                "value": ["19725455000", "19725455001"]
            }
        },
        "media": {
            "mediaTypeAudio": {
                "mediaFormat": ["PCMU;rate=8000;ptime=20",
                 "G729;rate=8000;annexb=yes",
                 "G722;rate=8000;bitrate=56k,64k"]
            },
            "fax": {
                "protocol": ["t38", "pass-through"]
            },
            "rtp": {
                "RTPTrigger": "true",
```

```
                "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"
            },
            "certLocation":
              "https://sipserviceprovider.com/certificateList.pem",
            "secureTelephonyIdentity": {
                "STIRCompliance": "true",
                "certDelegation": "true",
                "ACMEDirectory":
                    "https://sipserviceprovider.com/acme.html"
            }
        },
        "extensions": "timer;rel100;gin;path"
    }
}
```

### 9.2. Example Exchange

This section is an informational example depicting 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 has been pre-configured with the request target for the capability set document or has dynamically found the request target, the edge element generates a HTTPS GET request. This request can be challenged by the service provider to authenticate the enterprise.

```
GET /capdoc?trunkid=trunkent1456 HTTP/1.1
Host: capserver.ssp1.com
Accept:application/peering-info+json
```

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

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

{
    "peering-info": ...
}
```

## 10. Security Considerations

[TBD]

## 11. Acknowledgments

[TBD]

## 12. Normative References


**[ACME]** "Automatic Certificate Management Environment", <https://datatracker.ietf.org/doc/html/draft-ietf-acme-acme-18#section-7.1.1>.

**[BCP-14]** "Key words for use in RFCs to Indicate Requirement Levels", <https://www.rfc-editor.org/info/bcp14>.

**[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>.

**[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>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[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>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[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>.

[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>.

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

**[RFC6749]** Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

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

**[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>.

**[RFC7092]** Kaplan, H. and V. Pascual, "A Taxonomy of Session Initiation Protocol (SIP) Back-to-Back User Agents", RFC 7092, DOI 10.17487/RFC7092, December 2013, <https://www.rfc-editor.org/info/rfc7092>.

**[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>.

**[RFC7362]** Ivov, E., Kaplan, H., and D. Wing, "Latching: Hosted NAT Traversal (HNT) for Media in Real-Time Communication", RFC 7362, DOI 10.17487/RFC7362, September 2014, <https://www.rfc-editor.org/info/rfc7362>.

**[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>.

**[RFC8446]** Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

**[SIP-Connect-TR]** "SIP Connect Technical Recommendation", <https://www.sipforum.org/download/sipconnect-technical-recommendation-version-2-0/?wpdmdl=2818>.

## Authors' Addresses

Kaustubh Inamdar
Unaffiliated

Email: kaustubh.ietf@gmail.com

Sreekanth Narayanan
Cisco Systems

Email: sreenara@cisco.com

Cullen Jennings
Cisco Systems

Email: fluffy@iii.ca