DRINKS
Internet-Draft
Intended status: Standards Track
Expires: October 24, 2013

M.M. Marrache
Jerusalem College of Technology
D.S. Schwartz
XConnect
S.A. Ali
NeuStar

April 22, 2013

Session Peering Provisioning (SPP) Protocol over REST
draft-marrache-drinks-spp-protocol-rest-02

Abstract

The Session Peering Provisioning Framework (SPPF) is a framework that exists to enable the provisioning of session establishment data into Session Data Registries or SIP Service Provider data stores. This SPP Protocol implementation follows the REST architectural principles over HTTP to allow efficient provisioning of session establishment data. The benefits include inter alia better performances under high loads through the use of HTTP caches and proxies and less coupling between clients and servers. This document describes the specification of a protocol for transporting SPPF structures over HTTP(s) following REST architectural principles.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 24, 2013.

Table of Contents

1. Introduction

TBD

2. Terminology

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

3. Protocol Architecture

The following figure illustrates the technical architecture of the RESTful SPP Protocol:

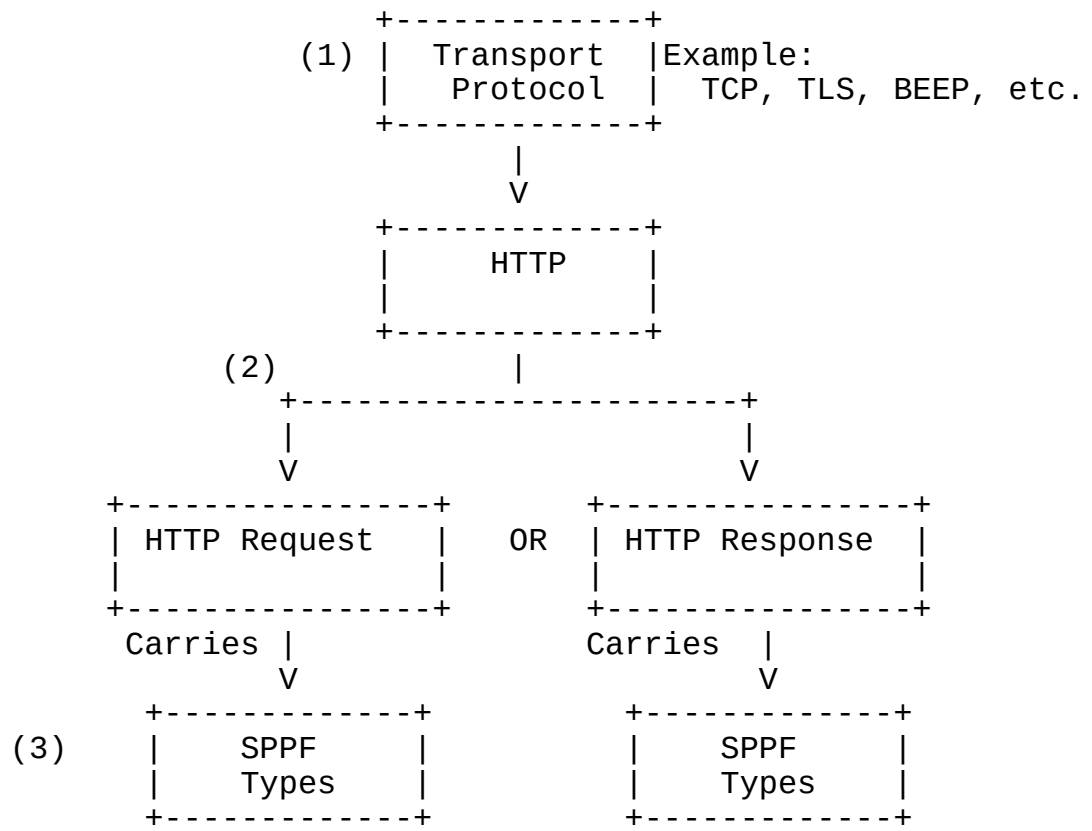


Figure 1: Layering and Technical Architecture of the RESTful SPP Protocol

RESTful SPP Protocol is supported by different technologies accross multiple layers as follows:

Layer 3: This is the data layer in which are defined the SPPF objects transported by the protocol between the involved components. These objects are defined in [I-D.draft-ietf-drinks-spp-framework].

Layer 2: The application protocol layer uses HTTP to allow clients perform the operations defined in the framework document. These operations are mostly provisioning operations. A client initiates an operation by sending an HTTP request to a server. Then, an HTTP response indicating the results of the operation is sent back by the server to the client. SPPF objects defined in the layer above are eventually carried by these HTTP messages.

Layer 1: The transport protocol layer represents the communication mechanism between the client and server. SPPF can be layered over any transport protocol that provides a set of basic requirements defined in the "Transport Protocol Requirements" section. But this document specifies the required mechanism.

SPPF is a request/reply framework that allows a client application to submit provisioning data and query requests to a server. The SPPF data structures are designed to be protocol agnostic. Concerns regarding encryption, non-repudiation, and authentication are beyond the scope of this document. For more details, please refer to the "Transport Protocol Requirements" section in the framework document.

4. Architectural Principles

4.1. Use of HTTP

HTTP(s) is the application protocol used by RESTful web services. HTTP 1.1 includes the "persistent connection" feature, which allows multiple HTTP request/response pairs to be transported across a single HTTP connection. This is an important performance optimization feature, particularly when the connection is an HTTPS connection where the relatively time consuming SSL handshake has occurred. Persistent connections SHOULD be used for the SPPF HTTP connections.

HTTP 1.1 [RFC2616] or higher SHOULD be used.

4.2. SPPF Objects as Resources

As mentioned in the previous section, the application protocol used by this protocol implementation is HTTP. Since HTTP has been conceived to operate on resources exposed on the web, the SPPF objects need to be exposed as resources. The SPPF objects then become available to clients for performing operations defined in the framework document.

Each resource exposed on the web is identified by a Uniform Resource Identifier (URI). Therefore, a URI is defined for each SPPF object. In order to be able to identify uniquely an SPPF object, the corresponding URI must include the attributes of a candidate key for this SPPF object. The attributes that form the key of each SPPF object are specified in the framework document. These attributes are included in the URI as path parameters.

4.2.1. Base URI

In the next sections, the concept of base URI will be used. It is the root URI where the RESTful service is located. All the URI defined by the following sections are relative to the base URI.

4.2.2. Resources URI

In the following sub-sections, for each type of resource, two URI are defined: the URI that identifies the resource type and the URI that uniquely identifies an instance of this resource type. In order to provide a URI for each SPPF object, a URI template is defined for each one of them. The URI templates defined in the following sub-sections are relative to the base URI defined in the Base URI section.

Each URI template defined in the following sub-sections starts with the version. It allows maintaining multiple versions of the same interface. The client specifies the version of the interface to use through the URI.

4.2.2.1. Destination Group

As mentioned in the framework document, a destination group is uniquely identified by the following attributes: the registrant and the destination group’s name. Therefore, the destination group resources are identified by the following URI template:

/${version}/rant/{rant}/DG/{name}

where:

o rant: registrant organization of the destination group.

- name: destination group's name.

The corresponding resource type URI is:

/${version}/rant/{rant}/DG

4.2.2.2. Telephone Number

As mentioned in the framework document, a telephone number (TN) is uniquely identified by the following attributes: the registrant and the telephone number. Therefore, the telephone number resources are identified by the following URI template:

/${version}/rant/{rant}/TN/{tn}

where:

- rant: registrant organization of the telephone number.
- tn: telephone number.

The corresponding resource type URI is:

/${version}/rant/{rant}/TN

4.2.2.3. Telephone Number Prefix

As mentioned in the framework document, a telephone number prefix (TNP) is uniquely identified by the following attributes: the registrant and the telephone number prefix. Therefore, the telephone number prefix resources are identified by the following URI template:

/${version}/rant/{rant}/TNP/{prefix}

where:

- rant: registrant organization of the telephone number prefix.
- prefix: telephone number prefix.

The corresponding resource type URI is:

/${version}/rant/{rant}/TNP

4.2.2.4. Telephone Number Range

As mentioned in the framework document, a telephone number range (TNR) is uniquely identified by the following attributes: the

registrant, the telephone number that starts the range and the telephone number that ends the range. Therefore, the telephone number range resources are identified by the following URI template:

/${version}/rant/{rant}/TNR/start/{startTn}/end/{endTn}

where:

- rant: registrant organization of the telephone number range.
- startTn: first telephone number of the range.
- endTn: last telephone number of the range.

The corresponding resource type URI is:

/${version}/rant/{rant}/TNR

4.2.2.5. Routing Number

As mentioned in the framework document, a routing number is uniquely identified by the following attributes: the registrant and the routing number. Therefore, the routing number resources are identified by the following URI template:

/${version}/rant/{rant}/RN/{rn}

where:

- rant: registrant organization of the routing number.
- rn: routing number.

The corresponding resource type URI is:

/${version}/rant/{rant}/RN

4.2.2.6. URI Public Identifier

As mentioned in the framework document, a public identifier URI is uniquely identified by the following attributes: the registrant and the URI. Therefore, the public identifier URI resources are identified by the following URI template:

/${version}/rant/{rant}/URI/{uri}

where:

- rant: registrant organization of the public identifier URI.
- uri: URI.

The corresponding resource type URI is:

/${version}/rant/{rant}/URI

4.2.2.7. SED Group

As mentioned in the framework document, a SED Group is uniquely identified by the following attributes: the registrant and the SED Group's name. Therefore, the SED Group resources are identified by the following URI template:

/${version}/rant/{rant}/SG/{name}

where:

- rant: registrant organization of the SED Group.
- name: SED Group's name.

The corresponding resource type URI is:

/${version}/rant/{rant}/SG

4.2.2.8. SED Record

As mentioned in the framework document, a SED Record is uniquely identified by the following attributes: the registrant and the SED Record's name. Therefore, the SED Record resources are identified by the following URI template:

/${version}/rant/{rant}/SR/{name}

where:

- rant: registrant organization of the SED Record.
- name: SED Record's name.

Unlike public identifiers types, there is no need to define one URI template for each subtype of SED Record (e.g. NAPTR) since a SED Record instance is identified by attributes that are defined at the SED Record level(i.e. rant and name).

The corresponding resource type URI is:

/${version}/rant/{rant}/SR

4.2.2.9. Egress Route

As mentioned in the framework document, an Egress route is uniquely identified by the following attributes: the registrant and the Egress route's name. Therefore, the Egress route resources are identified by the following URI template:

/${version}/rant/{rant}/ER/{name}

where:

- rant: registrant organization of the Egress route.
- name: Egress route's name.

The corresponding resource type URI is:

/${version}/rant/{rant}/ER

4.2.2.10. SED Group Offer

As mentioned in the framework document, a SED Group Offer is uniquely identified by the following attributes: the offering registrant (i.e. the registrant of the offered SED Group), the name of the offered SED Group and the organization to which the SED Group is offered. Therefore, the SED Group Offer resources are identified by the following URI template:

/${version}/rant/{rant}/SG/{sedGrpName}/offer/{offeredTo}

where:

- rant: offering registrant organization.
- sedGrpName: offered SED Group's name. This parameter along with the rant parameter uniquely identifies the offered SED Group.
- offeredTo: organization to which the SED Group is offered.

The corresponding resource type URI is:

/${version}/rant/{rant}/SG/{sedGrpName}/offer

4.2.2.11. SED Group Offer Acceptance

As defined by the framework document, a SED Group Offer may be accepted by the organization to which the SED Group has been offered. The acceptance of a SED Group Offer is performed through the use of an accept resource. The accept resources are identified by the following URI template:

/${version}/rant/{rant}/SG/{sedGrpName}/accept/{offeredTo}

where the parameters are defined as for the SED Group Offer resources.

4.2.2.12. SED Group Offer Rejection

As defined by the framework document, a SED Group Offer may be rejected by the organization to which the SED Group has been offered. The rejection of a SED Group Offer is performed through the use of a reject resource. The reject resources are identified by the following URI template:

/${version}/rant/{rant}/SG/{sedGrpName}/reject/{offeredTo}

where the parameters are defined as for the SED Group Offer resources.

4.2.2.13. Server Status

The Server Status is exposed as a singleton resource. Therefore, a single URI is need to identify this resource:

/${version}/ServerStatus

This resource contains information about the server as described later.

4.2.3. Resources Representations

For some operations defined by SPPF, resource representations may be present in the HTTP messages. When this is the case, the resource representation is carried in the HTTP message’s body. A resource may have many available representations where each one may use a specific format (e.g. XML, JSON).

Therefore, HTTP messages that carry resource representations MUST have their Content-Type HTTP header set to the appropriate media type.

4.3. HTTP methods and operations mapping

Most operations exposed by this protocol implementation are regular CRUD operations on resources. As mentioned earlier, an operation on a resource is initiated by a client when he sends an HTTP request that targets the URI that identifies the resource. In order to indicate the desired operation to perform on a given resource, a client selects one of the following HTTP methods:

4.3.1. GET

A client uses the HTTP GET method to retrieve a representation of a resource. The URI present in the HTTP request MUST be a full URI that identifies the particular resource to retrieve. In order to specify which representation formats are accepted, a client SHOULD include a Accept-Type header.

This HTTP method may be used for all the resources defined in the "Resources URI" section.

4.3.2. POST

A client uses the HTTP POST method to create a resource.

The nature of the URI present in an HTTP POST request depends on which type of resource the request targets.

When an HTTP POST request targets a resource of the following types

Destination Group, URI, TN, TNP, TNR, RN, SED Group, SED Record, Egress Route, SED Group Offer

the URI present in the request MUST be the corresponding resource type URI as defined in the "Resources URI" section. An HTTP POST request targetting a resource of the types above MUST carry a representation of the resource in its entity. The representation format of the resource MUST be specified using the Content-Type header.

When an HTTP POST request targets a resource of the following types

SED Group Offer Acceptance, SED Group Offer Rejection

the URI present in the request MUST be the URI that identifies the particular SED Group Offer Acceptance or SED Group Offer Rejection. This URI identifies the particular SED Group Offer to accept or reject. However, in this case, the entity of the HTTP request MUST be empty since all the information required to perform the acceptance /rejection operation is present in the URI.

4.3.3. PUT

The primary purpose of the HTTP PUT method is to allow a client to update the resource identified by the targetted URI. However, in some cases, the HTTP PUT method may also be used to create a resource.

Regardless the nature of the desired operation (i.e. update or create), the URI present in an HTTP PUT request MUST always be a full URI that identifies the particular resource targetted by the operation. Also, an HTTP PUT request MUST carry a representation of the resource to update or create in its entity. The representation format of the resource MUST be specified using the Content-Type header.

If the URI present in the HTTP PUT request corresponds to an existing resource, the server will replace the current resource representation by the representation carried in the request’s entity. Otherwise, the server will create the resource based on the representation carried in the request’s entity.

This HTTP method may be used for the following resources (defined in the "Resources URI" section): Destination Group, URI, TN, TNP, TNR, RN, SED Group, SED Record, Egress Route, SED Group Offer.

4.3.4. DELETE

A client uses the HTTP DELETE method to delete a resource. The URI present in the HTTP request MUST be a full URI that identifies the particular resource to delete.

This HTTP method may be used for the following resources (defined in the "Resources URI" section): Destination Group, URI, TN, TNP, TNR, RN, SED Group, SED Record, Egress Route, SED Group Offer.

5. Authentication and Session Management

To achieve integrity and privacy, conforming SPP Protocol Clients and Servers MUST support HTTP over TLS [RFC5246] as the secure transport mechanism. This combination of HTTP and TLS is referred to as HTTPS. And to accomplish authentication, conforming SPPF Clients and Servers MUST use HTTP Digest Authentication as defined in [RFC2617]. As a result, the communication session is established through the initial HTTP connection setup, the digest authentication, and the TLS handshake. When the HTTP connection is broken down, the communication session ends.

6. Operation Request and Response Structures

An SPPF client interacts with an SPPF server by using one of the supported transport mechanisms to send one or more requests to the server and receive corresponding replies from the server. The basic set of operations that an SPPF client can submit to an SPPF server and the semantics of those operations are defined in the "Framework Operations" section of the framework document. The following sub-sections describe how these operations should be performed in the context of this protocol implementation.

6.1. Add Operation Structure

In order to add an object to the registry, an authorized entity sends an add request to the registry. This request consists of an HTTP POST request on the URI that identifies the type of the resource to add, or an HTTP PUT request on the URI that identifies the resource to add. Since the format of the HTTP PUT request and response for resource creation is the same as for resource update, this format is not defined in this section (see the "Update Operation Structure" section). The representation of the resource to add is carried in the request's entity. After the operation is performed, the registry sends back an HTTP response to the client indicating if the request has been performed successfully, and if not, the reason of the failure. The following sub-sections describe the expected format of the HTTP requests and responses.

6.1.1. Add Request

The format of an HTTP POST request used to add an SPPF object to the registry is as follows:

```
POST ${ResourceTypeURI} HTTP/1.1
.....
[ClientTransId: ${ClientTransId}]
Content-Type: ...
Content-Length: ...

${ResourceRepresentation}
```

The data elements within the HTTP POST request are described as follows:

- ResourceTypeURI: The URI that identifies the type of the resource to add as defined in the "Resources URI" section.

- ClientTransId: An optional HTTP header representing a client-generated transaction ID that, within the context of the SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.

- ResourceRepresentation: HTTP request's entity that consists of the representation of the resource to add. The representation format MUST match the value of the Content-Type header.

6.1.2. Add Response

The format of an HTTP response to an HTTP POST request is as follows:

```
HTTP/1.1 ${StatusCode}
....
[ClientTransId: ${ClientTransId}]
ServerTransId: ${ServerTransId}
Content-Length: 0
```

The data elements within the HTTP response are described as follows:

- StatusCode: One of the available HTTP status codes indicating the result of the request. See Response Codes and Messages section.

- ClientTransId: An HTTP header representing the client transaction ID of the corresponding HTTP request, if provided. This value is simply an echo of the client transaction ID that SPPF client passed into the SPPF request. When included in the request, the SPPF server MUST return it in the corresponding response message.

- ServerTransId: A mandatory HTTP header representing the server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPF server.

6.2. Update Operation Structure

In order to update an object present in the registry, an authorized entity sends an update request to the registry. This request consists of an HTTP PUT request on the URI that identifies the resource to update. The new representation of the resource is carried in the request's entity. After the operation is performed, the registry sends back an HTTP response to the client indicating if

the request has been performed successfully, and if not, the reason of the failure. The following sub-sections describe the expected format of the HTTP requests and responses.

As mentioned by the previous section, the HTTP PUT request may also be used to create a resource. The format of the HTTP request and response is as defined in this section.

6.2.1. Update Request

The format of an HTTP PUT request used to update an SPPF object present in the registry is as follows:

```
PUT ${ResourceURI} HTTP/1.1
.....
[ClientTransId: ${ClientTransId}]
Content-Type: ...
Content-Length: ...

${ResourceRepresentation}
```

The data elements within the HTTP PUT request are described as follows:

- ResourceURI: The URI that identifies the resource to update as defined in the "Resources URI" section.
- ClientTransId: An optional HTTP header representing a client-generated transaction ID that, within the context of the SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.
- ResourceRepresentation: HTTP request’s entity that consists of the new representation of the resource. The representation format MUST match the value of the Content-Type header.

6.2.2. Update Response

The format of an HTTP response to an HTTP PUT request is as follows:

```
HTTP/1.1 ${StatusCode}
```

```
....
[ClientTransId: ${ClientTransId}]
ServerTransId: ${ServerTransId}
Content-Length: 0
```

The data elements within the HTTP response are described as follows:

- StatusCode: One of the available HTTP status codes indicating the result of the request. See Response Codes and Messages section.
- ClientTransId: An HTTP header representing the client transaction ID of the corresponding HTTP request, if provided. This value is simply an echo of the client transaction ID that SPPF client passed into the SPPF request. When included in the request, the SPPF server MUST return it in the corresponding response message.
- ServerTransId: A mandatory HTTP header representing the server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPF server.

6.3. Delete Operation Structure

In order to remove an object from the registry, an authorized entity sends a delete request to the registry. This request consists of an HTTP DELETE request on the URI that identifies the resource to delete. The request's entity SHOULD be empty since the resource to delete is uniquely identified by the URI included in the request. If an entity is present in the request, the registry MUST ignore it. After the operation is performed, the registry sends back an HTTP response to the client indicating if the request has been performed successfully, and if not, the reason of the failure. The following sub-sections describe the expected format of the HTTP requests and responses.

6.3.1. Delete Request

The format of an HTTP DELETE request used to delete an SPPF object from the registry is as follows:

```
DELETE ${ResourceURI} HTTP/1.1
.....
[ClientTransId: ${ClientTransId}]
Content-Length: 0
```

The data elements within the HTTP DELETE request are described as follows:

- ResourceURI: The URI that identifies the resource to delete as defined in the "Resources URI" section.

- ClientTransId: An optional query parameter representing a client-generated transaction ID that, within the context of the SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.

6.3.2. Delete Response

The format of an HTTP response to a delete request is as follows:

```
HTTP/1.1 ${StatusCode}
....
[ClientTransId: ${ClientTransId}]
ServerTransId: ${ServerTransId}
Content-Length: 0
```

The data elements within the HTTP response are described as follows:

- StatusCode: One of the available HTTP status codes indicating the result of the request. See Response Codes and Messages section.

- ClientTransId: An HTTP header representing the client transaction ID of the corresponding HTTP request, if provided. This value is simply an echo of the client transaction ID that SPPF client passed into the SPPF request. When included in the request, the SPPF server MUST return it in the corresponding response message.

- ServerTransId: An HTTP header representing the server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPF server.

6.4. Accept Operation Structure

In SPPF, a SED Group Offer can be accepted or rejected by, or on behalf of, the organization to whom the SED Group has been offered (refer "Framework Data Model Objects" section of the framework document for a description of the SED Group Offer object). The

Accept operation is used to accept such SED Group Offers by, or on behalf of, the organization. This request consists of an HTTP POST request on the URI that identifies the accept resource that corresponds to the concerned SED Group Offer. After the operation is performed, the registry sends back an HTTP response to the client indicating if the request has been performed successfully, and if not, the reason of the failure. The following sub-sections describe the expected format of the HTTP requests and responses.

6.4.1. Accept Request Structure

The format of an HTTP POST request used to accept a SED Group Offer is as follows:

```
POST /${version}/rant/{rant}/SG/{sedGrpName}/accept/{offeredTo} HTTP/1.1
.....
[ClientTransId: ${ClientTransId}]
Content-Length: 0
```

The data elements within the HTTP POST request are described as follows:

- rant: The identifier of the registrant organization that offered the SED Group.
- sedGrpName: The name of the SED Group offered by the registrant organization.
- offeredTo: The identifier of the organization to whom the SED Group has been offered.
- ClientTransId: An optional HTTP header representing a client-generated transaction ID that, within the context of the SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.

6.4.2. Accept Response

The format of an HTTP response to an Accept request is as follows:

```
HTTP/1.1 ${StatusCode}
```

```
....
[ClientTransId: ${ClientTransId}]
ServerTransId: ${ServerTransId}
Content-Length: 0
```

The data elements within the HTTP response are described as follows:

- StatusCode: One of the available HTTP status codes indicating the result of the request. See Response Codes and Messages section.
- ClientTransId: An HTTP header representing the client transaction ID of the corresponding HTTP request, if provided. This value is simply an echo of the client transaction ID that SPPF client passed into the SPPF request. When included in the request, the SPPF server MUST return it in the corresponding response message.
- ServerTransId: A header parameter representing the server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPF server.

6.5. Reject Operation Structure

In SPPF, a SED Group Offer can be accepted or rejected by, or on behalf of, the organization to whom the SED Group has been offered (refer "Framework Data Model Objects" section of the framework document for a description of the SED Group Offer object). The Reject operation is used to reject such SED Group Offers by, or on behalf of, the organization. This request consists of an HTTP POST request on the URI that identifies the reject resource for the concerned SED Group Offer. After the operation is performed, the registry sends back an HTTP response to the client indicating if the request has been performed successfully, and if not, the reason of the failure. The following sub-sections describe the expected format of the HTTP requests and responses.

6.5.1. Reject Request

The format of an HTTP POST request used to reject a SED Group Offer is as follows:

```
POST /${version}/rant/{rant}/SG/{sedGrpName}/reject/{offeredTo} HTTP/1.1
.....
[ClientTransId: ${ClientTransId}]
Content-Length: 0
```

The data elements within the HTTP POST request are described as follows:

- rant: The identifier of the registrant organization that offered the SED Group.
- sedGrpName: The name of the SED Group offered by the registrant organization.
- offeredTo: The identifier of the organization to whom the SED Group has been offered.
- ClientTransId: An optional HTTP header representing a client-generated transaction ID that, within the context of the SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.

6.5.2. Reject Response

The format of an HTTP response to a reject request is as follows:

```
HTTP/1.1 ${StatusCode}
....
[ClientTransId: ${ClientTransId}]
ServerTransId: ${ServerTransId}
Content-Length: 0
```

The data elements within the HTTP response are described as follows:

- StatusCode: One of the available HTTP status codes indicating the result of the request. See Response Codes and Messages section.
- ClientTransId: An HTTP header representing the client transaction ID of the corresponding HTTP request, if provided. This value is simply an echo of the client transaction ID that SPPF client

passed into the SPPF request. When included in the request, the SPPF server MUST return it in the corresponding response message.

- ServerTransId: A header parameter representing the server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPF server.

6.6. Get Operation Structure

In order to query the details of an object from the Registry, an authorized entity sends a get request to the registry. This request consists of an HTTP GET request on the URI that identifies the queried resource. After the operation is performed, the registry sends back an HTTP response to the client indicating if the request has been performed successfully, and if not, the reason of the failure. Moreover, if the queried object is found in the registry, the HTTP response's entity contains the representation of the result object. The following sub-sections describe the expected format of the HTTP requests and responses.

6.6.1. Get Request

The format of an HTTP GET request used to get an SPPF object is as follows:

```
GET ${ResourceURI} HTTP/1.1
.....
[Accept-Type: ${AcceptType}]
Content-Length: 0
```

The data elements within the HTTP GET request are described as follows:

- ResourceURI: The URI that identifies the resource to retrieve as defined in the "Resources URI" section.

- AcceptType: If an Accept-Type header is present in the request, it consists of the representation formats accepted by the client.

6.6.2. Get Response

The format of an HTTP response to a get request is as follows:

```
HTTP/1.1 ${StatusCode}
```

```
....
Content-Type: ...
Content-Length: ...

${ResourceRepresentation}
```

The data elements within the HTTP response are described as follows:

- StatusCode: One of the available HTTP status codes indicating the result of the request. See Response Codes and Messages section.

- ResourceRepresentation: HTTP response's entity that consists of the representation of the queried resource. The representation format MUST match the value of the Content-Type header.

7. Response Codes and Messages

HTTP provides a set of status codes that are used to indicate an overall result of the request to the client. This protocol implementation uses the status codes defined in [RFC2616].

7.1. 200 OK

When returned in response to an HTTP GET request, this status code indicates that the get operation performed successfully.

When returned in response to an HTTP PUT request, this status code indicates that the resource targetted by the URI present in the request has been updated.

When returned in response to an HTTP DELETE request, this status code indicates that the resource targetted by the URI present in the request has been deleted.

When returned in response to an HTTP POST request used to accept or reject a SED Group Offer, this status code indicates that the SED Group Offer resource targetted by the URI present in the request has been accepted or rejected.

7.2. 201 Created

When returned in response to an HTTP POST request used to create a resource, this status code indicates that the resource targetted by the URI present in the request has been created.

7.3. 400 Bad Request

When returned in response to any HTTP request, this status code indicates that the HTTP request received by the server is invalid.

7.4. 401 Unauthorized

When returned in response to any HTTP request, this status code indicates that authentication is required and has failed or has not yet been provided.

7.5. 403 Forbidden

When returned in response to any HTTP request, this status code indicates that the client is authenticated but not authorized to perform the desired operation.

7.6. 404 Not Found

When returned in response to an HTTP GET or DELETE request, this status code indicates that the URI present in the HTTP request targets a nonexistent resource.

When returned in response to an HTTP POST request used to accept or reject a SED Group Offer, this status code indicates that the SED Group Offer resource targetted by the URI present in the request does not exist.

7.7. 405 Method Not Allowed

When returned in response to any HTTP request, this status code indicates that the HTTP method present in the request is not allowed to be used for the resource identified by the given URI.

7.8. 415 Unsupported Media Type

When returned in response to an HTTP POST or PUT request, this status code indicates that the Content-Type header has a value corresponding to a media type not supported by the server.

7.9. 500 Internal Server Error

When returned in response to any HTTP request, this status code indicates that an unexpected internal system or server error happened.

7.10. 503 Service Unavailable

When returned in response to any HTTP request, this status code indicates that the server is temporarily unable to process incoming HTTP requests.

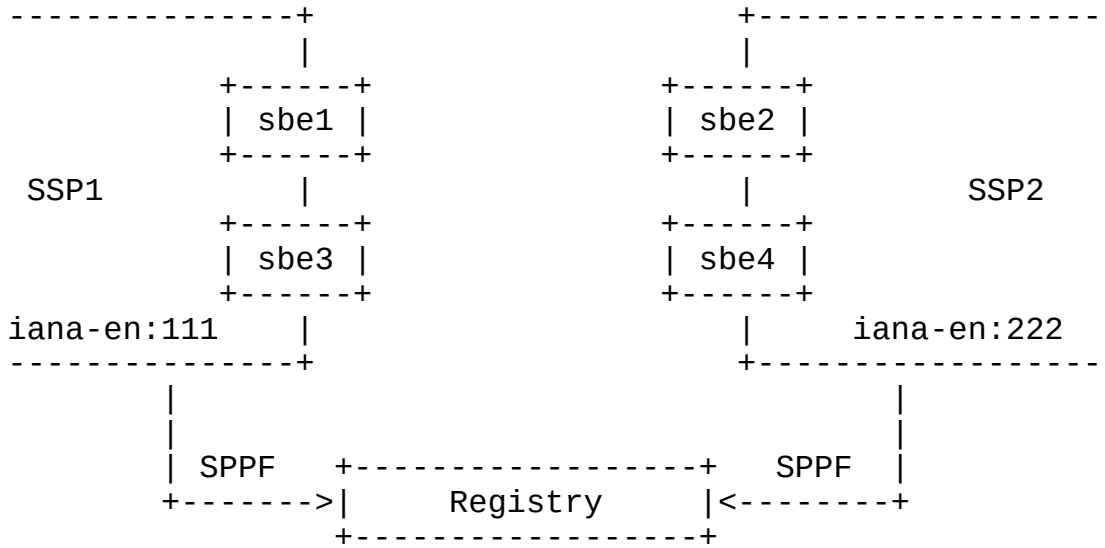
8. Protocol Operations

Refer the "Framework Operations" section of the framework document for a description of all SPPF operations, and any necessary semantics that MUST be adhered to in order to conform with the SPPF specification.

9. SPP Protocol over SOAP Examples

This section shows examples of HTTP message exchange between two SIP Service Providers (SSP) and a registry. The messages in this section are HTTP requests/responses that may include XML content representing the SPPF objects defined in the framework document. This section relies on the XML data structures defined in the base SPPF specification [I-D.draft-ietf-drinks-spp-framework]. So refer to that document to understand XML object types embedded in these example messages.

In this sample use case scenario, SSP1 and SSP2 provision resource data in the registry and use SPPF constructs to selectively share the SED groups. In the figure below, SSP2 has two ingress SBE instances that are associated with the public identities that SSP2 has the retail relationship with. Also, the two SBE instances for SSP1 are used to show how to use SPPF to associate route preferences for the destination ingress routes and exercise greater control on outbound traffic to the peer's ingress SBEs.



9.1. Add Destination Group

SSP2 adds a destination group to the registry for use later using a POST request. SSP2 sets a unique transaction identifier 'txn_1479' for tracking purposes through the ClientTransId header field. It also sets the Content-Type header field to application/xml since it provides an XML representation of the destination group in the HTTP entity. The name of the destination group is DEST_GRP_SSP2_1.

```
POST /v1.0/rant/iana-en:222/DG HTTP/1.1
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<DestGroup xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
</DestGroup>
```

The registry processes the request and returns a 201 Created response confirming successful creation of the named destination group. Also, besides returning a unique server transaction identifier (through the ServerTransId header field), Registry also returns the matching client transaction identifier from the request message back to the SPPF client. The response also includes the Location header field indicating the URI of the created destination group.

```
HTTP/1.1 201 Created
............
ClientTransId: txn_1479
ServerTransId: tx_12345
Location: ${base_uri}/v1.0/rant/iana-en:222/DG/DEST_GRP_SSP2_1
Content-Length: 0
```

9.2. Update Destination Group

SSP2 updates the destination group previously created (i.e. DEST_GRP_SSP2_1). In this case, no information about the destination

group can be modified since the only information it contains is its key, and the key can't be modified.

```
PUT /v1.0/rant/iana-en:222/DG/DEST_GRP_SSP2_1 HTTP/1.1
...........
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<DestGroup xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
</DestGroup>
```

The registry processes the request and returns a 200 OK response confirming successful update of the named destination group.

```
HTTP/1.1 200 OK
...........
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.3. Add SED Records

SSP2 adds a SED record in the form of ingress route to the registry. In this example, the SED record is a NAPTR record. Note that the NAPTR recorder is added in a disabled state (i.e. isInSvc is set to false).

```
POST /v1.0/rant/iana-en:222/SR HTTP/1.1
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<NAPTR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
```

```
    xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd"
    xmlns="urn:ietf:params:xml:ns:sppf:rest:1">
     <sppfb:rant>iana-en:222</sppfb:rant>
     <sppfb:rar>iana-en:223</sppfb:rar>
     <sppfb:sedName>SED_SSP2_SBE2</sppfb:sedName>
     <sppfb:isInSvc>false</sppfb:isInSvc>
     <sppfb:order>10</sppfb:order>
     <sppfb:flags>u</sppfb:flags>
     <sppfb:svcs>E2U+sip</sppfb:svcs>
     <sppfb:regx>
       <sppfb:ere>^(.*)$</sppfb:ere>
       <sppfb:repl>sip:\1@sbe2.ssp2.example.com</sppfb:repl>
     </sppfb:regx>
   </NAPTR>
```

The registry returns a success response.

```
HTTP/1.1 201 Created
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Location: ${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE2
Content-Length: 0
```

9.4. Update SED Records

SSP2 updates the SED record previsously added to the registry by enabling it (i.e. setting isInSvc to true).

```
PUT /v1.0/rant/iana-en:222/SR/SED_SSP2_SBE2 HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<NAPTR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd"
 xmlns="urn:ietf:params:xml:ns:sppf:rest:1">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:sedName>SED_SSP2_SBE2</sppfb:sedName>
```

```
    <sppfb:isInSvc>true</sppfb:isInSvc>
    <sppfb:order>10</sppfb:order>
    <sppfb:flags>u</sppfb:flags>
    <sppfb:svcs>E2U+sip</sppfb:svcs>
    <sppfb:regx>
      <sppfb:ere>^(.*)$</sppfb:ere>
      <sppfb:repl>sip:\1@sbe2.ssp2.example.com</sppfb:repl>
    </sppfb:regx>
  </NAPTR>
```

The registry returns a success response.

```
HTTP/1.1 200 OK
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.5. Add SED Records -- URIType

SSP2 adds another SED record to the registry and makes use of URIType.

```
POST /v1.0/rant/iana-en:222/SR HTTP/1.1
............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<URI xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:sedName>SED_SSP2_SBE4</sppfb:sedName>
  <sppfb:isInSvc>true</sppfb:isInSvc>
  <sppfb:ere>^(.*)$</sppfb:ere>
  <sppfb:uri>sip:\1;npdi@sbe4.ssp2.example.com</sppfb:uri>
</URI>
```

The registry returns a success response.

```
HTTP/1.1 201 Created
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Location: ${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE4
Content-Length: 0
```

9.6. Add SED Group

SSP2 creates the grouping of SED records (e.g. ingress routes) and chooses higher precedence for SED_SSP2_SBE2 by setting a lower number for the "priority" attribute, a protocol agnostic precedence indicator. The SED Group is added with a disabled state.

```
POST /v1.0/rant/iana-en:222/SG HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<SedGrp xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:sedGrpName>SED_GRP_SSP2_1</sppfb:sedGrpName>
  <sppfb:sedRecRef>
    <sppfb:sedKey xsi:type="ObjKeyType">
      <ref>${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE2</ref>
    </sppfb:sedKey>
    <sppfb:priority>80</sppfb:priority>
  </sppfb:sedRecRef>
  <sppfb:sedRecRef>
    <sppfb:sedKey xsi:type="ObjKeyType">
      <ref>${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE4</ref>
    </sppfb:sedKey>
    <sppfb:priority>100</sppfb:priority>
  </sppfb:sedRecRef>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:isInSvc>false</sppfb:isInSvc>
  <sppfb:priority>10</sppfb:priority>
```

```
   </SedGrp>
```

The registry returns a success response.

```
   HTTP/1.1 201 Created
   ............
   ClientTransId: txn_1479
   ServerTransId: tx_12345
   Location: ${base_uri}/v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1
   Content-Length: 0
```

9.7. Update SED Group

SSP2 enables the previously created SED Group by setting its isInSvc field to true.

```
   PUT /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1 HTTP/1.1
   ............
   ClientTransId: txn_1479
   Content-Type: application/xml
   Content-Length: ...

   <SedGrp xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
     xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
     <sppfb:rant>iana-en:222</sppfb:rant>
     <sppfb:rar>iana-en:223</sppfb:rar>
     <sppfb:sedGrpName>SED_GRP_SSP2_1</sppfb:sedGrpName>
     <sppfb:sedRecRef>
       <sppfb:sedKey xsi:type="ObjKeyType">
         <ref>${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE2</ref>
       </sppfb:sedKey>
       <sppfb:priority>80</sppfb:priority>
     </sppfb:sedRecRef>
     <sppfb:sedRecRef>
       <sppfb:sedKey xsi:type="ObjKeyType">
         <ref>${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE4</ref>
       </sppfb:sedKey>
       <sppfb:priority>100</sppfb:priority>
     </sppfb:sedRecRef>
     <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
```

```
    <sppfb:isInSvc>true</sppfb:isInSvc>
    <sppfb:priority>10</sppfb:priority>
  </SedGrp>
```

The registry returns a success response.

```
HTTP/1.1 200 OK
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.8. Add Public Identity -- Successful COR claim

SSP2 activates a TN public identity by associating it with a valid destination group. Further, SSP2 puts forth a claim that it is the carrier-of-record for the TN.

```
POST /v1.0/rant/iana-en:222/TN HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<TN xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:tn>+12025556666</sppfb:tn>
  <sppfb:corInfo>
    <sppfb:corClaim>true</sppfb:corClaim>
  </sppfb:corInfo>
</TN>
```

Assuming that the registry has access to TN authority data and it performs the required checks to verify that SSP2 is in fact the service provider of record for the given TN, the request is processed

successfully. In order to get the COR claim status, SSP2 will have to perform a GET on the created public identity.

```
HTTP/1.1 201 Created
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Location: ${base_uri}/v1.0/rant/iana-en:222/TN/+12025556666
Content-Length: ...
```

9.9. Update Public Identity

SSP2 updates the previously created TN public identity.

```
PUT /v1.0/rant/iana-en:222/TN/+12025556666 HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<TN xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:tn>+12025556666</sppfb:tn>
  <sppfb:corInfo>
    <sppfb:corClaim>true</sppfb:corClaim>
  </sppfb:corInfo>
</TN>
```

The registry returns a success response.

```
HTTP/1.1 200 OK
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: ...
```

```
<CORInfo xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:corClaim>true</sppfb:corClaim>
  <sppfb:cor>true</sppfb:cor>
  <sppfb:corDate>2010-05-30T09:30:11Z</sppfb:corDate>
</CORInfo>
```

9.10. Add LRN

If another entity that SSP2 shares session establishment information (e.g. routes) with has access to Number Portability data, it may choose to perform route lookups by routing number. Therefore, SSP2 associates a routing number to a destination group in order to facilitate ingress route discovery.

```
POST /v1.0/rant/iana-en:222/RN HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<RN xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:rn>2025550000</sppfb:rn>
</RN>
```

Registry completes the request successfully and returns a favorable response to the SPPF client.

```
HTTP/1.1 201 Created
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Location: ${base_uri}/v1.0/rant/iana-en:222/RN/2025550000
Content-Length: 0
```

9.11. Update LRN

SSP2 updates the previously created routing number.

```
PUT /v1.0/rant/iana-en:222/RN/2025550000 HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<RN xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:rn>2025550000</sppfb:rn>
</RN>
```

Registry completes the request successfully and returns a favorable response to the SPPF client.

```
HTTP/1.1 200 OK
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.12. Add TN Range

Next, SSP2 activates a block of ten thousand TNs and associates it to destination group DEST_GRP_SSP2_1.

```
POST /v1.0/rant/iana-en:222/TNR HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<TNR xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
```

```
     xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
     <sppfb:rant>iana-en:222</sppfb:rant>
     <sppfb:rar>iana-en:223</sppfb:rar>
     <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
     <sppfb:range>
       <sppfb:startRange>+12026660000</sppfb:startRange>
       <sppfb:endRange>+12026669999</sppfb:endRange>
     </sppfb:range>
   </TNR>
```

Registry completes the request successfully and returns a favorable response.

```
   HTTP/1.1 201 Created
   ............
   ClientTransId: txn_1479
   ServerTransId: tx_12345
   Location: ${base_uri}/v1.0/rant/iana-en:222/TNR/start/+12026660000/end/+1
2026669999
   Content-Length: 0
```

9.13. Update TN Range

SSP2 updates the previously created block of TNs.

```
   PUT /v1.0/rant/iana-en:222/TNR/start/+12026660000/end/+12026669999 HTTP/1
.1
   ............
   ClientTransId: txn_1479
   Content-Type: application/xml
   Content-Length: ...

   <TNR xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
     xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
     <sppfb:rant>iana-en:222</sppfb:rant>
     <sppfb:rar>iana-en:223</sppfb:rar>
     <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
     <sppfb:range>
       <sppfb:startRange>+12026660000</sppfb:startRange>
       <sppfb:endRange>+12026669999</sppfb:endRange>
```

```
    </sppfb:range>
  </TNR>
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.14. Add TN Prefix

Next, SSP2 activates a block of ten thousand TNs using the TNPType structure and identifying a TN prefix.

```
POST /v1.0/rant/iana-en:222/TNP HTTP/1.1
............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<TNP xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:tnPrefix>+1202777</sppfb:tnPrefix>
</TNP>
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 201 Created
............
ClientTransId: txn_1479
```

```
ServerTransId: tx_12345
Location: ${base_uri}/v1.0/rant/iana-en:222/TNP/+1202777
Content-Length: 0
```

9.15. Update TN Prefix

SSP2 updates the previously created TN prefix.

```
PUT /v1.0/rant/iana-en:222/TNP/+1202777 HTTP/1.1
............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<TNP xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:tnPrefix>+1202777</sppfb:tnPrefix>
</TNP>
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.16. Enable Peering -- SED Group Offer

In order for SSP1 to complete session establishment for a destination TN where the target subscriber has a retail relationship with SSP2, it first requires an asynchronous bi-directional handshake to show mutual consent. To start the process, SSP2 initiates the peering handshake by offering SSP1 access to its SED group.

```
   POST /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1/offer HTTP/1.1
   ............
   ClientTransId: txn_1479
   Content-Type: application/xml
   Content-Length: ...

   <SedGrpOffer xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
     xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
     <sppfb:rant>iana-en:222</sppfb:rant>
     <sppfb:rar>iana-en:223</sppfb:rar>
     <sppfb:sedGrpOfferKey xsi:type="SedGrpOfferKeyType">
       <sgRef>${base_uri}/v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1</sgRef>
       <offeredTo>iana-en:111</offeredTo>
     </sppfb:sedGrpOfferKey>
     <sppfb:status>offered</sppfb:status>
     <sppfb:offerDateTime>2006-05-04T18:13:51.0Z</sppfb:offerDateTime>
   </SedGrpOffer>
```

Registry completes the request successfully and confirms that the SSP1 will now have the opportunity to weigh in on the offer and either accept or reject it. The registry may employ out-of-band notification mechanisms for quicker updates to SSP1 so they can act faster, though this topic is beyond the scope of this document.

```
   HTTP/1.1 201 Created
   ............
   ClientTransId: txn_1479
   ServerTransId: tx_12345
   Location: ${base_uri}/v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1/offer/iana-
en:111
   Content-Length: 0
```

9.17. Enable Peering -- SED Group Offer Accept

SSP1 responds to the offer from SSP2 and agrees to have visibility to SSP2 session establishment information (e.g. ingress routes).

```
   POST /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1/accept/iana-en:111 HTTP/1.1
   ............
   ClientTransId: txn_1479
   Content-Length: 0
```

Registry confirms that the request has been processed successfully. From this point forward, if SSP1 looks up a public identity through the query resolution server, where the public identity is part of the destination group by way of "SED_GRP_SSP2_1" session establishment data association, SSP2 ingress SBE information will be shared with SSP1.

```
HTTP/1.1 200 OK
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.18. Remove Peering -- SED Group Offer Reject

SSP1 had earlier accepted to have visibility to SSP2 session establishment data. SSP1 now decides to no longer maintain this visibility and hence rejects the SED Group Offer.

```
POST /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1/reject/iana-en:111 HTTP/1.1
.............
ClientTransId: txn_1479
Content-Length: 0
```

Registry confirms that the request has been processed successfully. From this point forward, if SSP1 looks up a public identity through the query resolution server, where the public identity is part of the destination group by way of "SED_GRP_SSP2_1" session establishment data association, SSP2 ingress SBE information will NOT be shared with SSP1 and hence SSP2 ingress SBE will NOT be returned in the query response.

```
HTTP/1.1 200 OK
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.19. Add Egress Route

SSP1 wants to prioritize all outbound traffic to the ingress routes associated with the "SED_GRP_SSP2_1" SED Group record, through "sbe1.ssp1.example.com".

```
POST /v1.0/rant/iana-en:111/ER HTTP/1.1
............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<EgrRte xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:111</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:egrRteName>EGR_RTE_01</sppfb:egrRteName>
  <sppfb:pref>50</sppfb:pref>
  <sppfb:regxRewriteRule>
    <sppfb:ere>^(.*@)(.*)$</sppfb:ere>
    <sppfb:repl>\1\2?route=sbe1.ssp1.example.com</sppfb:repl>
  </sppfb:regxRewriteRule>
  <sppfb:ingrSedGrp xsi:type="ObjKeyType">
    <ref>${base_uri}/v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1</ref>
  </sppfb:ingrSedGrp>
</EgrRte>
```

Since peering has already been established, the request to add the egress route has been successfully completed.

```
HTTP/1.1 201 Created
............
ClientTransId: txn_1479
ServerTransId: tx_12345
Location: ${base_uri}/v1.0/rant/iana-en:111/ER/EGR_RTE_01
Content-Length: 0
```

9.20. Update Egress Route

SSP1 wants to modify the priority of the previously created egress route (i.e. EGR_RTE_01).

```
PUT /v1.0/rant/iana-en:111/ER/EGR_RTE_01 HTTP/1.1
.............
ClientTransId: txn_1479
Content-Type: application/xml
Content-Length: ...

<EgrRte xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:111</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:egrRteName>EGR_RTE_01</sppfb:egrRteName>
  <sppfb:pref>40</sppfb:pref>
  <sppfb:regxRewriteRule>
    <sppfb:ere>^(.*@)(.*)$</sppfb:ere>
    <sppfb:repl>\1\2?route=sbe1.ssp1.example.com</sppfb:repl>
  </sppfb:regxRewriteRule>
  <sppfb:ingrSedGrp xsi:type="ObjKeyType">
    <ref>${base_uri}/v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1</ref>
  </sppfb:ingrSedGrp>
</EgrRte>
```

Since peering has already been established, the request to update the egress route has been successfully completed.

```
HTTP/1.1 200 OK
.............
ClientTransId: txn_1479
ServerTransId: tx_12345
Content-Length: 0
```

9.21. Get Destination Group

SSP2 sends an HTTP GET request to fetch the last provisioned record for destination group DEST_GRP_SSP2_1.

```
GET /v1.0/rant/iana-en:222/DG/DEST_GRP_SSP2_1 HTTP/1.1
```

```
.............
Accept-Type: application/xml
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
.............
Content-Type: application/xml
Content-Length: ...

<DestGroup xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:cDate>2012-10-22T09:30:10Z</sppfb:cDate>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
</DestGroup>
```

9.22. Get Public Identity

SSP2 obtains the last provisioned record associated with a given TN.

```
GET /v1.0/rant/iana-en:222/TN/+12025556666 HTTP/1.1
.............
Accept-Type: application/xml
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
.............
Content-Type: application/xml
Content-Length: ...
```

```
<TN xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
  <sppfb:tn>+12025556666</sppfb:tn>
  <sppfb:corInfo>
    <sppfb:corClaim>true</sppfb:corClaim>
    <sppfb:cor>true</sppfb:cor>
    <sppfb:corDate>2010-05-30T09:30:10Z</sppfb:corDate>
  </sppfb:corInfo>
</TN>
```

9.23. Get SED Group Request

SSP2 obtains the last provisioned record for the SED group SED_GRP_SSP2_1.

```
GET /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1 HTTP/1.1
............
Accept-Type: application/xml
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
............
Content-Type: application/xml
Content-Length: ...

<SedGrp xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:cDate>2012-10-22T09:30:10Z</sppfb:cDate>
  <sppfb:sedGrpName>SED_GRP_SSP2_1</sppfb:sedGrpName>
  <sppfb:sedRecRef>
```

```
      <sppfb:sedKey xsi:type="ObjKeyType">
        <ref>${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE2</ref>
      </sppfb:sedKey>
      <sppfb:priority>80</sppfb:priority>
    </sppfb:sedRecRef>
    <sppfb:sedRecRef>
      <sppfb:sedKey xsi:type="ObjKeyType">
        <ref>${base_uri}/v1.0/rant/iana-en:222/SR/SED_SSP2_SBE4</ref>
      </sppfb:sedKey>
      <sppfb:priority>100</sppfb:priority>
    </sppfb:sedRecRef>
    <sppfb:dgName>DEST_GRP_SSP2_1</sppfb:dgName>
    <sppfb:peeringOrg>iana-en:111</sppfb:peeringOrg>
    <sppfb:isInSvc>true</sppfb:isInSvc>
    <sppfb:priority>10</sppfb:priority>
  </SedGrp>
```

9.24. Get SED Group Offers Request

SSP2 fetches the last provisioned SED group offer to the <peeringOrg> SSP1.

```
GET /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1/offer/FD182737 HTTP/1.1
.............
Accept-Type: application/xml
Content-Length: 0
```

Registry processes the request successfully and returns a favorable response. In this example, the offer has been accepted by SSP1.

```
HTTP/1.1 200 OK
.............
Content-Type: application/xml
Content-Length: ...

<SedGrpOffer xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:222</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:cDate>2012-10-22T09:30:10Z</sppfb:cDate>
```

```
    <sppfb:sedGrpOfferKey xsi:type="SedGrpOfferKeyType">
      <sgRef>${base_uri}/v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1</sgRef>
      <offeredTo>iana-en:111</offeredTo>
    </sppfb:sedGrpOfferKey>
    <sppfb:status>accepted</sppfb:status>
    <sppfb:offerDateTime>2006-05-04T18:13:51.0Z</sppfb:offerDateTime>
    <sppfb:acceptDateTime>2006-07-08T11:12:46.0Z</sppfb:acceptDateTime>
  </SedGrpOffer>
```

9.25. Get Egress Route

SSP2 wants to verify the last provisioned record for the egress route called EGR_RTE_01.

```
GET /v1.0/rant/iana-en:111/ER/EGR_RTE_01 HTTP/1.1
............
Accept-Type: application/xml
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
............
Content-Type: application/xml
Content-Length: ...

<EgrRte xmlns="urn:ietf:params:xml:ns:sppf:rest:1"
  xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ietf:params:xml:ns:sppf:rest:1 sppr.xsd">
  <sppfb:rant>iana-en:111</sppfb:rant>
  <sppfb:rar>iana-en:223</sppfb:rar>
  <sppfb:egrRteName>EGR_RTE_01</sppfb:egrRteName>
  <sppfb:pref>40</sppfb:pref>
  <sppfb:regxRewriteRule>
    <sppfb:ere>^(.*@)(.*)$</sppfb:ere>
    <sppfb:repl>\1\2?route=sbe1.ssp1.example.com</sppfb:repl>
  </sppfb:regxRewriteRule>
  <sppfb:ingrSedGrp xsi:type="ObjKeyType">
    <ref>${base_uri}/v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1</ref>
  </sppfb:ingrSedGrp>
```

```
</EgrRte>
```

9.26. Delete Destination Group

SSP2 initiates a request to delete the destination group DEST_GRP_SSP2_1.

```
DELETE /v1.0/rant/iana-en:222/DG/DEST_GRP_SSP2_1 HTTP/1.1
.............
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
.............
ServerTransId: tx_12345
Content-Length: 0
```

9.27. Delete Public Identity

SSP2 chooses to de-activate the TN and remove it from the registry.

```
DELETE /v1.0/rant/iana-en:222/TN/+12025556666 HTTP/1.1
.............
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
.............
ServerTransId: tx_12345
Content-Length: 0
```

9.28. Delete SED Group Request

SSP2 removes the SED group called SED_GRP_SSP2_1.

```
DELETE /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1 HTTP/1.1
.............
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
.............
ServerTransId: tx_12345
Content-Length: 0
```

9.29. Delete SED Group Offers Request

SSP2 no longer wants to share SED group SED_GRP_SSP2_1 with SSP1.

```
DELETE /v1.0/rant/iana-en:222/SG/SED_GRP_SSP2_1/offer/FD182737 HTTP/1.1
.............
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response. Restoring this resource sharing will require a new SED group offer from SSP2 to SSP1 followed by a successful SED group accept request from SSP1.

```
HTTP/1.1 200 OK
.............
ServerTransId: tx_12345
Content-Length: 0
```

9.30. Delete Egress Route

SSP1 decides to remove the egress route with the label EGR_RTE_01.

```
DELETE /v1.0/rant/iana-en:111/ER/EGR_RTE_01 HTTP/1.1
.............
Content-Length: 0
```

Registry completes the request successfully and returns a favorable response.

```
HTTP/1.1 200 OK
.............
ServerTransId: tx_12345
Content-Length: 0
```

10. Security Considerations

RESTful SPP Protocol is used to query and update session peering data and addresses, so the ability to access this protocol should be limited to users and systems that are authorized to query and update this data. Because this data is sent in both directions, it may not be sufficient for just the client or user to be authenticated with the server. The identity of the server should also be authenticated by the client, which is often accomplished using the TLS certificate exchange and validation described in [RFC2818]. SPP Protocol messages may include sensitive information, routing data, lists of resolvable addresses, etc. So when used in a production setting and across non-secure networks, SPP Protocol should only be used over communications channels that provide strong encryption for data privacy.

10.1. Integrity, Privacy, and Authentication

The RESTful SPP Protocol relies on an underlying secure transport for integrity and privacy. Such transports are expected to include TLS/ HTTPS. In addition to the application level authentication imposed by an SPPF server, there are a number of options for authentication within the transport layer and the messaging envelope. These include TLS client certificates and HTTP Digest Access Authentication headers.

At a minimum, all conforming RESTful SPP Protocol implementations MUST support HTTPS.

10.2. Vulnerabilities

The above protocols may have various vulnerabilities, and these may be inherited by the RESTful SPP Protocol. RESTful SPP Protocol itself may have vulnerabilities because an authorization model is not explicitly specified in the current specification.

Sections 5 and 10.1 describe requirements for HTTPS support using TLS. Non-anonymous TLS servers can optionally request a certificate from a TLS client; that option is not a requirement for this protocol. This presents a denial of service risk in which unauthenticated clients can consume server CPU resources by creating TLS sessions. The risk is increased if the server supports client-initiated renegotiation. This risk can be mitigated by disabling client-initiated renegotiation on the server and by ensuring that other means (such as firewall access control lists) are used to restrict unauthenticated client access to servers.

In conjunction with the above, it is important that REST SPP Protocol implementations implement an authorization model that considers the source of each query or update request and determines whether it is reasonable to authorize that source to perform that specific query or update.

10.3. Deployment Environment Specifics

Some deployments of REST SPP Protocol could choose to use transports without encryption. This presents vulnerabilities but could be selected for deployments involving closed networks or debugging scenarios. However, the vulnerabilities of such a deployment could be a lack of integrity and privacy of the data transported in this type of deployment.

11. Acknowledgements

TBD

12. References

12.1. Normative References

[I-D.draft-ietf-drinks-spp-framework]
Cartwright, K.C., Bhatia, V.B., Ali, S.A., and D.S. Schwartz, "Session Peering Provisioning Framework ", draft-ietf-drinks-spp-framework-02 (work in progress), July 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P.M., Hostetler, J.L., Lawrence, S.D., Leach, P.J., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

12.2. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RoyFielding] Fielding, R.T., "Architectural Styles and the Design of Network-based Software Architectures", University of California , 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[W3C.REC-xml-20081126] Sperberg-McQueen, C., Yergeau, F., Bray, T., Maler, E., and J. Paoli, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

Authors' Addresses

Mickael Marrache
Jerusalem College of Technology
Havaad Haleumi St. 21
Jerusalem 91160
Israel

Email: mickaelmarrache@gmail.com

David Schwartz
XConnect
316 Regents Park Road
London N3 2XJ
United Kingdom

Email: dschwartz@xconnect.net

Syed Wasim Ali
NeuStar
46000 Center Oak Plaza
Sterling, VA 20166
USA

Email: syed.ali@neustar.biz