Sliceable Network Management API

Version 1.01

September 2012

Table of Contents

Tables

1 Introduction

This document intends to describe for application developers a virtual resource management service as defined by this sliceable network management API. It documents in detail a RESTful API to configure various functional components which constitute the operation of a sliceable virtual layer 2. At a minimum, this virtual layer is able to learn MAC addresses and perform aging in the standard manner to provide forwarding actions for message routing to a closed set of users.

2 Scope

This document specifies an interface to configure interconnections between virtual resources to give users a granular control over routing decisions to either restrict or allow traffic. This API as a prerequisite it specifies that an emulated layer 2 is created. It does not specify a limit on the number of emulated layers 2 that may be configured but scaling issues might arise from the number of ports and end point attachments that can be supported on a particular emulated layer 2.

This API does not support state management of resources hence no mechanism exists to subscribe to receive events from managed resources.

The API does not support any guards preventing access to objects created by other clients. It could be that higher level APIs provide the necessary authentication but this is outside the scope of this API.

3 Overview

The API interrogates a stack of resources in a hierarchically way. Resources at a lower level can not be accessed unless resources at the higher level are accessible. Such dependencies should be well understood because side effects might occur. For example deletion of a network can result in all other dependent instances disappearing as well.
Users interact with this API through its supported addressing model and are unaware of the underlying network infrastructure that are using. This network infrastructure is a virtual layer 2 designed within the Trema framework supplying a distributed topology arrangement. The API does a simple mapping between its emulated network and its attachment points based on information received from incoming logical/physical ports. Routes are calculated based on configured logical domains rather than the multiple VMs it represents, therefore extending network reachability. The virtual layer 2 imposes no restriction on overlapping MAC addresses and VLAN ids. By virtualizing all layer 2 functions the API distributes resource management such that you only need a single configuration point.

4 Definitions

For the purpose of this document the following terms and definitions apply.

4.1 Sliceable API

It is a conceptual term used to describe the system that this API is targeted for. The services that this API provides can be categorized as Infrastructure as a Service (IaaS) in the cloud computing delivery model.

4.2 Tenant

It represents a top level entity defining a large pool of network resources. A tenant may span to cover one or more network entities. It isolates traffic from multiple tenants over a shared physical infrastructure essential within a virtualized data center environment.

4.3 Network

Models the relationship of a virtual resource pool (ports/attachments) that this entity may have. It represents a concept that groups resources together sharing certain characteristics defining their unique behavior within the network.

4.4 Port

Mapping of port attributes to this logical entity to enable network connectivity.

4.5 Attachment

An entity that as its name suggests can either be attached to a network or port entity objects.

This entity represents a mapping to a value that exposes a control attribute to the entity being attached (network/port).

4.6 Entity/Element

Terms used interchangeably referring to objects that this API manipulates.

4.7 Container

A simple grouping of objects of certain type with similar characteristics.

5 Types of resources in Sliceable API

Resource type	**Description**
Tenant	A container of one or more objects uniquely addressable by its tenant identifier.
Network	A container of one or more objects uniquely addressable by its network identifier.
Port	A container of one or more objects addressable by its logical port identifier. A port container may also have attachment container objects.
Attachment	An entity that represents a binding of key/value pairs that can be attached either to network or port. Attachment also can be viewed as a container accessible through its attachment identifier.

Resource access applies to all synchronous operations regarding getting, setting and enumerating values. Subclauses below define a mechanism for acquiring JSON based representations of entities using this RESTful API. A URI syntax may indicate the type of entity being enumerated or a particular selector object.

6 Tenant

A tenant entity refers to logical grouping of one or more network entities aggregated to represent the tenant's features as a whole. An organization registered as a tenant should be able to manage all the services that this API provides.

6.1 Create a tenant object

Create a tenant object with its contextual information id, description. Tenant is the top-most managed object.

The following HTTP POST creates a tenant object at the specified URI.

Resource:

POST /tenants

The POST request includes two name/value pairs, id and description. The server would create a tenant object even if both attributes id/description not supplied.

Table 1 Request message body – Create a tenant object.

Field name	**Type**	**Description**	**Requirement**
id	JSON String	A unique locally assigned textual name of the tenant.	Optional
description	JSON String	A textual string containing information about the tenant.	Optional

Example: POST /tenants

```
POST /tenants HTTP/1.1
Content-type: application/json
{
      "id" : "tenant-id",
      "description" : "our organization"
}
```

6.1.1 Response status

The HTTP status codes returned from creating a tenant object are shown in Table 2.

Table 2 HTTP status codes – Create a tenant object.

HTTP status code	**Description**
202 Created	Operation succeeded. A new tenant object is created.
422 Bad Request	Invalid parameters or field names or attempt to create an existing tenant object.
500 Internal Server Error	Indicates an internal processing error. The server is unable to create a tenant object.

6.2 List multiple tenant objects

The following HTTP GET lists one or more tenant representation objects at the specified request URI. The message is targeted to return an array of tenant objects.

Resource:

GET /tenants

Example: Perform a GET to the tenant container URI.

```
GET /tenants HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      { "id" : "tenant-1", "description" : "abc organization-warehouse" },
      { "id" : "tenant-2", "description" : "abc organization-data-management" }
]
```

6.2.1 Response headers

The following response headers expected as shown in Table 3.

Table 3 Response headers – List multiple tenant objects.

Header	**Type**	**Requirement**
Content-Type	application/json	If present a JSON array of zero, one or more tenant objects. An empty array indicates that there are no tenant objects.
Content-Type	text/plain	If present an implementation specific error message.

6.2.2 Response status

The HTTP status codes returned from listing multiple tenant objects are shown in Table 4.

Table 4 HTTP status codes – List multiple tenant objects.

HTTP status code	**Description**
200 OK/Success	One or more tenant object found and returned.
500 Internal Server Error	Failed to locate and return details of any tenant.

6.3 List a tenant object

The following HTTP GET lists an existing tenant object at the specified URI.

Resource:

GET /tenants/<tenant-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

Example: Perform a GET to the tenant object URI.

```
GET /tenants/tenant-1 HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      "description" : "abc organization-warehouse"
]
```

6.3.1 Response headers

The following response headers expected as shown in Table 5.

Table 5 Response headers – List a tenant object.

Header	**Type**	**Requirement**
Content-Type	application/json	Optional. If present the referenced tenant object instance if configured.
Content-Type	text/plain	Optional. If present the referenced tenant object instance is not found.

6.3.2 Response message body

The response message body for listing a specific tenant object is shown in Table 6.

Table 6 Response message body – List a tenant object.

Field name	**Type**	**Description**
description	JSON String	A readable description of this tenant object instance.

6.3.3 Response status

The HTTP status codes returned from listing a tenant object are shown in Table 7.

Table 7 HTTP status codes – List a tenant object.

HTTP status code	**Description**
200 OK/Success	A tenant object found and returned.
404 Not Found	Unable to locate the specified tenant.
500 Internal Server Error	An internal processing error. The server is unable to locate the tenant object.

6.4 Delete a tenant object

It is an operation to delete a tenant resource in its entirety.

The following HTTP DELETE deletes a tenant object at the specified URI. Any dependent entities visible to this tenant object (networks, ports, attachments) would also be deleted.

Resource:

DELETE /tenants/<tenant-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

6.4.1 Response status

The HTTP status codes returned from deleting a tenant object are shown in Table 8.

Table 8 HTTP status codes – Delete a tenant object.

HTTP status code	**Description**
202 Accepted	A tenant object successfully deleted.
404 Not Found	A tenant object already deleted or is not found.
500 Internal Server Error	An internal processing error. The server is unable to delete the tenant object.

6.5 Modify a tenant object

Request that the tenant identified by the request URI be updated with the enclosed representation. If a tenant exists at the request URI the description should be modified.

Resource:

PUT /tenants/<tenant-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

The PUT request includes a name/value pair of description.

Table 9 Request message body – Modify a tenant object.

Field name	**Type**	**Description**
description	JSON String	A textual string containing information about the tenant.

Example: PUT to the tenant object URI to alter its description.

```
PUT /tenants/tenant-1 HTTP/1.1
Content-type: application/json
{
      "description" : "another organization"
}
```

The following shows the server's response.

```
HTTP/1.1 202 Accepted
```

6.5.1 Response status

The HTTP status codes returned from modifying a tenant object are shown in Table 10.

Table 10 HTTP status codes – Modifying a tenant object.

HTTP status code	**Description**
202 Accepted	A tenant object is modified.
404 Not Found	An attempt is made to modify a non-existing tenant object.
422 Unprocessable Entity	Failed to parse the modification parameters.
500 Internal Server Error	An internal processing error. The server is unable to modify the tenant object.

7 Network

Within a tenant a network represents a sliceable virtual layer 2. It defines the semantics of a logical boundary that allows ports and other attachment objects bound to it implement a particular function. A network is identified by a unique id, auto-generated if not given that remains constant within the life of the entity.

7.1 Create a network object

A create operation creates a network object with its initial representation. If the target network object already exists the create operation would fail returning an error.

This creates and initializes a sliceable virtual layer 2 without any functionality for connectivity yet.

The following HTTP POST creates a network object at the specified URI.

Resource:

POST /tenants/<tenant-id>/networks

Where:

<tenant-id> : the identifier of a tenant object instance.

The POST request message body includes two name/value pairs, id and description.

Table 11 Request message body – Create a network object.

Field name	**Type**	**Description**	**Requirement**
id	JSON String	Any textual description as long as it is unique which is used to identify the object. If the request doesn't include the id the server creates and returns one in response.	Optional
description	JSON String	A name that this network is known as.	Optional

If a user attempts to create a network with unknown tenant-id, a tenant with the tenant-id would be created. This function may be supported in this version of the Sliceable API.

Example: Create a network object which name is "net-1" in "tenant-1".

```
POST /tenants/tenant-1/networks HTTP/1.1
Content-type: application/json
{
      "id" : "net-1",
      "description" : "marketing department"
}
```

7.1.1 Response status

The HTTP status codes returned from creating a network object are shown in Table 12.

Table 12 HTTP status codes – Create a network object.

HTTP status code	**Description**
202 Created	Operation succeeded. A new network object is created.
422 Bad Request	Invalid parameters or field names or attempt to create an existing network object.
500 Internal Server Error	An internal processing error. The server is unable to create a network object.

7.2 List multiple network objects

The following HTTP GET lists one or more network objects at the specified URI. The message is targeted to return an array of network objects. If an object can not be retrieved due to conflicting locking or simultaneous access or any other conflict an error response is returned. If the request results in no data being found an empty response is returned (an empty array with no values).

Resource:

GET /tenants/<tenant-id>/networks

Where:

<tenant-id> : the identifier of a tenant object instance.

Example: Perform a GET to the network container URI.

```
GET /tenants/tenant-1/networks HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      { "id" : "net-1", "description" : "marketing department" },
      { "id" : "net-2", "description" : "development department" }
]
```

7.2.1 Response headers

The following response headers expected as shown in Table 13.

Table 13 Response headers – List multiple network objects.

Header	Type	Requirement
Content-Type	application/json	If present a JSON array of zero, one or more network objects.
Content-Type	text/plain	If present an implementation specific error message.

7.2.2 Response status

The HTTP status codes returned from listing multiple network objects are shown in Table 14.

Table 14 HTTP status codes – List multiple network objects.

HTTP status code	Description
200 OK/Success	One or more network object is found.
404 Not Found	Unable to retrieve one or more network object because specified tenant-id is not found.
500 Internal Server Error	An internal processing error. The server is unable to locate one or more network object.

7.3 List a network object

The following HTTP GET lists an existing network object at the specified URI.

Resource:

GET /tenants/<tenant-id>/networks/<net-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

Example: Perform a GET to the network object URI.

```
GET /tenants/tenant-1/networks/net-2 HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "bindings" : [],
      "description" : "development department"
}
```

7.3.1 Response headers

The following response headers expected as shown in Table 15.

Table 15 Response headers – List a network object.

Header	**Type**	**Requirement**
Content-Type	application/json	Optional. If present the referenced network object instance.
Content-Type	text/plain	Optional. If present the referenced network object instance is not found.

7.3.2 Response message body

The response message body for listing a specific network object is shown in Table 16.

Table 16 Response message body – List a network object.

Field name	Type	Description
bindings	JSON Array	Any port or attachment object associated with this network instance. If there are no bindings an empty array is returned.
description	JSON String	A readable description of this network object instance.

7.3.3 Response status

The HTTP status codes returned from listing a network object are shown in Table 17.

Table 17 HTTP status codes – List a network object.

HTTP status code	Description
200 OK/Success	A network object found and returned.
404 Not Found	Unable to locate a network object because of incorrect tenant-id or network-id.
500 Internal Server Error	An internal processing error. A server is unable to locate the network object.

7.4 *Delete a network object*

It is an operation to delete a network resource in its entirety. The delete operation specifies a precondition to match in order for the operation to be carried out. This does not mean that the operation would succeed other errors may prevent and cause the delete operation to fail.

The following HTTP DELETE deletes a network object at the specified URI. Any dependent entities visible to this network object (ports, attachments) would also be deleted.

Resource:

DELETE /tenants/<tenant-id>/networks/<net-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

7.4.1 Response status

The HTTP status codes returned from deleting a network object are shown in Table 18.

Table 18 HTTP status codes – Delete a network object.

HTTP status code	**Description**
202 Accepted	A network object successfully deleted.
404 Not Found	A network object already deleted or is not found.
500 Internal Server Error	An internal processing error. The server is unable to delete the network object.

7.5 Modify a network object

Ability to modify an existing network object. A successful PUT operation would modify the network object modifiable properties and leave the other properties unchanged. If the targeting PUT network representation and the actual saved representation do not differ the object is left intact. A modification to a network object should not disturb the preservation order of other objects. A client user must query again to display the updated results.

Resource:

PUT /tenants/<tenant-id>/networks/<net-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

The PUT request includes a name/value pair of description.

Table 19 Request message body – Modify a network object.

Field name	**Type**	**Description**
description	JSON String	A name that this network is known as.

Example: PUT to the network object URI to alter its description.

```
PUT /tenants/tenant-1/networks/net-1 HTTP/1.1
Content-type: application/json
{
      "description" : "another department"
}
```

The following shows the server's response.

```
HTTP/1.1 202 Accepted
```

7.5.1 Response status

The HTTP status codes returned from modifying a network object are shown in Table 20.

Table 20 HTTP status codes – Modify a network object.

HTTP status code	**Description**
202 Accepted	A network object is modified.
404 Not Found	An attempt is made to modify a non-existing network object.
422 Unprocessable Entity	Failed to parse the modification parameters.
500 Internal Server Error	An internal processing error. The server is unable to modify the network object.

8 Port

A port describes a set of properties a virtual resource must have in order to connect to the network. This membership declaration allows user traffic from a virtual resource to be propagated among other registered resources within the same network even if physically separated. Physically a port object could be any OpenFlow aware switch hooked up to this network. There is no restriction on the number of ports to be added to a network. A port attachment further limits the extent of the network by establishing boundaries dictating which specific MAC addresses are forwarded.

A ports URI allows access to specific port objects by referencing their ids. A client user can create inquire about the configuration or delete port object instances.

8.1 Create a port object

Create a port by defining a number of key/values attributes. The created port should be accessible either by the ports container or by its unique id.

Resource:

POST /tenants/<tenant-id>/networks/<net-id>/ports

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

The POST request contains the following name/value pairs.

Table 21 Request message body – Create a port object.

Field name	**Type**	**Description**	**Requirement**
id	JSON String	Any textual description as long as it is unique within the scope of this network object instance. If the request doesn't include the id the server creates and returns one in response. Although optional client users may prefer to set this to create a more human-readable identifier.	Optional
datapath_id	JSON String	Identifies a unique instance of any OpenFlow aware switch. It should be specified in hexadecimal format prefixed with “0x”.	Mandatory
port	JSON Number	Specifies the physical port number that any OpenFlow aware switch attached and frames are received.	Mandatory
vid	JSON Number	Assign a VLAN identifier to a port. Set this value to 65535 to skip VLAN setting. The VLAN tags let you logically separate traffic on a port into multiple channels.	Mandatory

Example:The following HTTP POST creates a port object at the specified URI.

```
POST /tenants/tenant-1/networks/net-1/ports HTTP/1.1
Content-type: application/json
{
      "id" : "port-1",
      "datapath_id" : "0x0000000000001234",
      "port" : 1,
      "vid" : 1024
}
```

On successful creation the following HTTP response is returned.

```
HTTP/1.1 202 Accepted
```

8.1.1 Response status

The HTTP status codes returned from creating a port object are shown in Table 22.

Table 22 HTPP status codes– Create a port object.

HTTP status code	**Description**
202 Accepted	The referenced port object successfully added.
422 Unprocessable Entity	Attempt to create an existing port object.
404 Not Found	Reference to non-existing network object is found.
500 Internal Server Error	An internal processing error. The server is unable to create a port object.

8.2 *List multiple port objects*

It is a query to return all port objects belonging to a specific network. The following HTTP GET lists one or more port objects at the specified URI. If the request results in no data being found an empty response is returned (an empty array with no values).

To find out what ports are available for a network.

Resource:

GET /tenants/<tenant-id>/networks/<net-id>/ports

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

Example: Perform a GET to the port container at the network's URI.

```
GET /tenants/tenant-1/networks/net-1/ports HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
	{
		"vid" : 1024,
		"datapath_id" : "0x0000000000001234",
		"id" : "port-1",
		"port" : 1
	}
]
```

8.2.1 Response headers

The following response headers expected as shown in Table 23.

Table 23 Response headers – List multiple port objects.

Header	Type	Requirement
Content-Type	application/json	If present a JSON array of zero one or more JSON port objects.
Content-Type	text/plain	If present an implementation specific error message.

8.2.2 Response message body

The response message body for enumerating multiple port objects of a JSON port object is shown in Table 24.

Table 24 Response message body – List multiple port objects.

Field name	Type	Description
vid	JSON Number	The VLAN id attached to a port.
datapath_id	JSON String	A unique identifier of an OpenFlow switch.
id	JSON String	A name this port is known as.
port	JSON Number	An index into a list of physical ports.

8.2.3 Response status

The HTTP status codes returned from listing port objects shown in Table 25.

Table 25 HTTP status codes – List multiple port objects.

HTTP status code	Description
200 OK/Success	One or more port object returned.
404 Not Found	Unable to locate one or more port object using the specified tenant-id or network-id.
500 Internal Server Error	An internal processing error. The server is unable to locate one or more port objects.

8.3 List a port object

The following HTTP GET reads the attributes of an existing port object at the specified URL.

Resource:

GET /tenants/<tenant-id>/networks/<net-id>/ports/<port-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<port-id> : the identifier of a port object instance.

Example: Perform a GET to the port object URI.

```
GET /tenants/tenant-1/networks/net-1/ports/port-1 HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "config" : { "vid" : 1024, "datapath_id" : "0x0000000000001234", "port" : 1 }
}
```

8.3.1 Response headers

The following response headers expected as shown in Table 26.

Table 26 Response headers – List a port object.

Header	**Type**	**Requirement**
Content-Type	application/json	If present a JSON port object.
Content-Type	text/plain	If present an implementation specific error message.

8.3.2 Response message body

The response message body for listing a particular port object is shown in Table 27.

Table 27 Response message body – List a port object.

Field name	**Type**	**Description**
config	JSON object	An object encapsulating the port instance with the following attributes: • vid • datapath_id • port

8.3.3 Response status

The HTTP status codes returned from listing a port object are shown in Table 28.

Table 28 HTTP status codes– List a port object.

HTTP status code	**Description**
200 OK/Success	A port object successfully located.
404 Not Found	A port object was not found at the specified URI.
500 Internal Server Error	An internal processing error. The server is unable to locate a port object.

8.4 Delete a port object

The following HTTP DELETE deletes an existing port object at the specified URI.

Resource:

DELETE /tenants/<tenant-id>/networks/<net-id>/ports/<port-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<port-id> : the identifier of a port object instance.

Example: Perform a DELETE to the port object URI.

```
DELETE /tenants/tenant-1/networks/net-1/ports/port-1 HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 202 Accepted
```

This API does not define any addressing schemes or techniques for batched port delete operations. An attempt to delete all ports should fail with a 405 (Method not Allowed) error.

8.4.1 Response status

The HTTP status codes returned from deleting a port object are shown in Table 29.

Table 29 Response status codes – Delete a port object.

HTTP status code	**Description**
202 Accepted	A port object successfully deleted.
404 Not Found	A port object is already deleted or not found.
500 Internal Server Error	An internal processing error. The server is unable to delete the port object.

8.5 Create an attachment object to a port object

To create a new attachment you use a HTTP POST operation.

Resource:

POST /tenants/<tenant-id>/networks/<net-id>/ports/<port-id>/attachments

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<port-id> : the identifier of a port object instance.

The POST request contains the following name/value pairs.

Table 30 Request message body – Create an attachment object to a port object.

Field name	**Type**	**Description**	**Requirement**
id	JSON string	Specifies an identifier which is a user defined string that uniquely addresses this attachment.	Mandatory
mac	JSON string	A 6-octet MAC address value used as a unique identifier for the network/port that contains this attachment. Determines frame forwarding behavior on MAC addresses.	Mandatory

Example: The following HTTP POST creates an attachment object at the specified URI.

```
POST /tenants/tenant-1/networks/net-1/ports/port-1/attachments HTTP/1.1
Content-type: application/json
{
      "id" : "attachment-1",
      "mac" : "11:22:33:44:55:66"
}
```

This request attaches a particular object (attachment-1) with MAC address (11:22:33:44:55:66) to a specify port (port-1).

On successful creation the following HTTP response is returned.

```
HTTP/1.1 202 Accepted
```

8.5.1 Response status

The HTTP status codes returned from creating an attachment object are shown in Table 31.

Table 31 HTTP status – Create an attachment object to a port object.

HTTP status	Description
202 Accepted	An attachment object successfully created.
422 Unprocessable Entity	Attempt to create an existing attachment object or invalid parameters specified or reference to non-existing port object.
404 Not Found	Reference to non-existing network object is found.
500 Internal Server Error	An internal processing error. The server is unable to add an attachment object.

8.6 *Read multiple port attachment objects*

To return a list of all port attachment objects perform a HTTP GET on the specified URI.

Resource:

GET /tenants/<tenant-id>/networks/<net-id>/ports/<port-id>/attachments

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<port-id> : the identifier of a port object instance.

Example: Perform a GET to the attachments container URI.

```
GET /tenants/tenant-1/networks/net-1/ports/port-3/attachments HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
	{
		"id" : "attachment-1",
		"mac" : "11:22:33:44:55:66"
	}
]
```

8.6.1 Response headers

The following response headers expected as shown in Table 32.

Table 32 Response headers – List multiple port attachment objects.

Header	**Type**	**Requirement**
Content-Type	application/json	If present a JSON array of zero one, or more JSON attachment objects.
Content-Type	text/plain	If present an implementation specific error message.

8.6.2 Response status

The HTTP status codes returned from listing multiple port attachment objects are shown in Table 33.

Table 33 HTTP status codes – List multiple port attachment objects.

HTTP status code	**Description**
200 OK/Success	Returns one or more port attachment objects.
404 Not Found	Failed to return one or more port attachment object either because an invalid tenant-id or network-id specified.
500 Internal Server Error	An internal processing error. The server is unable to locate one or more port attachment objects.

8.7 List a port attachment object

To return a specific port attachment object perform a HTTP GET on the specified URI.

Resource:

GET /tenants/<tenant-id>/networks/<net-id>/ports/<port-id>/attachments/<attachment-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<port-id> : the identifier of a port object instance.

<attachment-id> : the identifier of the port attachment object instance.

Example: Perform a GET to the attachments container URI using a specific attachment-id.

```
GET /tenants/tenant-1/networks/net-1/ports/port-3/attachments/host3-mac HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      {
            "id" : "host3-mac",
            "mac" : "00:00:00:00:00:03"
      }
]
```

8.7.1 Response headers

The following response headers expected as shown in Table 34.

Table 34 Response headers – List a port attachment object.

Header	**Type**	**Requirement**
Content-Type	application/json	If present a JSON object instance of a port attachment object.
Content-Type	text/plain	If present an implementation specific error message.

8.7.2 Response status

The HTTP status codes returned when querying a port attachment object shown in Table 35.

Table 35 HTTP status codes – List a port attachment object.

HTTP Status	**Description**
200 OK/Success	A port attachment object found and returned.
404 Not Found	Unable to locate a port attachment object using either the tenant-id, network-id, port-id, attachment-id.
500 Internal Server Error	An internal processing error. The server is unable to list a port attachment object.

8.8 Delete a port attachment object

The following HTTP DELETE deletes an existing attachment object at the specified network and port URI.

Resource:

DELETE /tenants/<tenant-id>/networks/<net-id>/ports/<port-id>/attachments/<attachment-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<port-id> : the identifier of a port object instance.

<attachment-id> : the identifier of the port attachment object instance.

Example: Perform a DELETE to the port attachment object URI.

```
DELETE /tenants/tenant-1/networks/net-1/ports/port-3/attachments/123 HTTP/1.1
```

The following shows the server's response:

```
HTTP/1.1 202 Accepted
```

8.8.1 Response status

The HTTP status codes returned from deleting a port attachment object shown in Table 36.

Table 36 Response status codes – Delete a port attachment object.

HTTP status code	**Description**
202 Accepted	A port attachment object successfully deleted.
404 Not Found	A port attachment object is not found.
500 Internal Server Error	An internal processing error. The server is unable to delete a port attachment object.

9 Network Attachment

A network attachment object represents a passive uniquely identifiable resource that remains unreachable until such time that joins the network by initiating its own transmission. All operations and data semantics on the network attachment object are the same as the port attachment object. Different level of indirection achieves information independence between the two entities. All network attachment devices must be OpenFlow aware devices.

9.1 Create an attachment object to a network object

To create as new attachment object to a network object you use the HTTP POST operation. An attachment to a network can restrict or permit access across multiple networks.

Resource:

POST /tenants/<tenant-id>/networks/<net-id>/attachments

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

The POST request contains the following name/values pairs.

Table 37 Request message body – Create an attachment object to a network object.

Field	**Type**	**Description**	**Requirement**
id	JSON String	Any textual description as long as it remains unique within the network instance.	Mandatory
mac	JSON String	A 6-octet Ethernet MAC address of the attachment.	Mandatory

Example: The following HTTP POST creates an attachment object at the specified URI.

```
POST /tenants/tenant-1/networks/net-1/attachments HTTP/1.1
Content-type: application/json
{
      "id" : "test-attachment",
      "mac" : "00:00:00:00:00:03"
}
```

On successful creation the following HTTP response is returned.

```
HTTP/1.1 202 Accepted
```

9.1.1 Response status

The HTTP status codes returned from creating an attachment object to a network object are shown in Table 38.

Table 38 HTTP status codes – Create an attachment object to a network object.

HTTP status code	**Description**
202 Created	Indicates that no error occurred and the server was able to create an attachment object to a network object in the attachments container.
404 Not Found	Unable to locate specified parameters tenant-id or network-id.
422 Unprocessable Entity	An attempt to create an existing attachment object to a network object.
500 Internal Server Error	An internal processing error. The server is unable to create an attachment object to a network object.

9.2 List multiple network attachment objects

To read a group of network attachment objects perform a HTTP GET on the specified URI.

Resource:

GET /tenants/<tenant-id>/networks/<net-id>/attachments

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

Example: Perform a GET to the network attachments container URI.

```
GET /tenants/tenant-1/networks/net-1/attachments HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
	{
		"id" : "test-attachment",
		"mac" : "00:00:00:00:00:01"
	},
	{
		"id" : "another-test-attachment",
		"mac" : "00:00:00:00:00:02"
	}
]
```

9.2.1 Response headers

The following response headers expected as shown in Table 39.

Table 39 Response headers – List multiple network attachment objects.

Header	Type	Requirement
Content-Type	application/json	If present a JSON array of zero one, or more JSON attachment objects.
Content-Type	text/plain	If present an implementation specific error message.

9.2.2 Response status

The HTTP status codes returned from listing multiple network attachment objects are shown in Table 40.

Table 40 HTTP status codes – List multiple network attachment objects.

HTTP status code	**Description**
200 OK/Success	No error occurred. One or more network attachment objects found and returned.
404 Not Found	Indicates that one of the specified parameters not found.
500 Internal Server Error	An internal processing error. The server is unable to list one or more network attachment objects.

9.3 List a network attachment object

Allows a user client to select and retrieve a network attachment instance by its identifier.

Resource:

GET /tenants/<tenant-id>/networks/<net-id>/attachments/<attachment-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<attachment-id> : the identifier of the network attachment object instance.

Example: Perform a GET to the specified URI.

```
GET /tenants/tenant-1/networks/net-1/attachments/test-attachment HTTP/1.1
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "id" : " test-attachment",
      "mac" : "00:00:00:00:00:01"
}
```

9.3.1 Response status

The HTTP status codes returned from listing a network attachment object shown in Table 41.

Table 41 HTTP status codes – List a network attachment object.

HTTP status code	Description
200 OK/Success	A network attachment object is found.
404 Not Found	A network attachment object is not found.
500 Interval Server Error	An internal processing error. The server is unable to list a network attachment object.

9.4 Delete a network attachment object

The following HTTP DELETE deletes an existing network attachment object at the specified URI.

Resource:

DELETE /tenants/<tenant-id>/networks/<net-id>/attachments/<attachment-id>

Where:

<tenant-id> : the identifier of a tenant object instance.

<net-id> : the identifier of a network object instance.

<attachment-id> : the identifier of the network attachment object instance.

Example: Perform a DELETE to the network attachment object URI.

```
DELETE /tenants/tenant-1/networks/net-1/attachments/test-attachment HTTP/1.1
```

The following shows the server's response:

```
HTTP/1.1 202 Accepted
```

9.4.1 Response status

The HTTP status codes returned from deleting a network attachment object are shown in Table 42.

Table 42 HTTP status codes– Delete a network attachment object.

HTTP status code	**Description**
202 Accepted	A network attachment object successfully deleted.
404 Not Found	A network attachment object is not found.
500 Internal Server Error	An internal processing error. The server is unable to delete the network attachment object.

Version	**Description**	**Date**
1.0	Initial version	April/27/2012
1.01	Specified in hexadecimal format prefixed with "0x".	September/28/2012