OpenStack Networking API in CEE in BSP Deployment

Cloud Execution Environment

INTERWORK DESCRIPTION

7/155 19-CRA 119 1862/5-V3 Uen A

Contents

1 Introduction

This document describes the use of the Application Programming Interface (API) for networking in the Cloud Execution Environment (CEE) on Blade Server Platform (BSP). The API is based on the OpenStack networking component (Neutron).

1.1 API Version

The CEE Networking API is based on OpenStack Networking API v2.

1.2 Document References

This section contains the official OpenStack API reference.

1.2.1 API Design Base Reference

For the description of the API operations and extensions of networking, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

This is a stored copy of the OpenStack API Reference document version that was the base for the development of this version of CEE.

Note: It is possible that the date on the front page differs from the revision date in this document. The front page shows the date on which the document was generated.

2 Prerequisites

The following sections list the prerequisites.

2.1 CEE Networking Configurations

CEE networking can be configured during the installation of the CEE on BSP. For details about the configurations, refer to the document *Configuration File Guide*.

Table 1 Neutron Configurations

Name	Neutron in CEE, Compared to Standard OpenStack Neutron
`ericsson_user_spec`	• Does not use L3 agent and floating IPs. • Uses several extensions that are explained in this document. • Can use additional features if properly configured by the user at installation of CEE • Uses a user specific ML2 driver to provide L2 connectivity for tenant networks. • Does not use L3 service plugin.
`ericsson_cmx`	• Does not use L3 agent and floating IPs. • Uses several extensions that are explained in this document. • Uses a ML2 driver to manage CMX switches in BSP configuration to provide L2 connectivity for tenant networks. • Does not use L3 service plugin.

2.2 Segmentation IDs

A segmentation ID is implemented as a VLAN tag in the virtual and physical switches. The range must be defined during the installation of the CEE. CMX Virtual Routers to CEE region connectivity also requires a range Virtual LANs (VLANs) to be specified at installation of CEE. The cloud administrator has to use the latter range when using the Neutron provider extension, as described in the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in *OpenStack API Complete Reference*. The two ranges must not overlap.

Table 2 VLAN Allocation Example

Name	Range	Description
Default	130-3999	Used by Neutron for tenant separation
Provider	50-129	Used for Neutron to CMX Virtual Routers connectivity

Note: The table is an example, it does not mandate specific or default values. The values must be configured before the CEE region is installed. The configuration of values is described in the *Configuration File Guide*.

Each Neutron network requires one segmentation ID. Segmentation IDs have to be unique, they cannot be used on several Neutron networks. Segmentation IDs on Neutron networks are static and cannot be changed after creation of the Neutron network.

3 Supported Operations

The following sections contain information about the API operations and API extensions in CEE.

3.1 Basic OpenStack Operations

For the detailed description of basic Networking API operations, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

3.1.1 Limitations

For the CEE-specific limitations and recommendations, see Section 6 on page 15.

3.2 OpenStack Extensions

This section contains information about which Networking API extensions are supported.

- Agent Management Extension

  The following agents are supported:

  – `neutron-dhcp-agent`

  – `neutron-openvswitch-agent`

- Agent Schedulers

  The following agents are supported:

  – DHCP Agent Scheduler

  – Device Agent Scheduler

- List Extensions

  Note: Supported, but the reply contains unsupported operations.

- Networks multiple provider extension (networks)

- Quotas extension (quotas)

4 Deviations

This chapter describes the deviations between vanilla OpenStack networking and CEE networking.

4.1 Partly Supported Operations

The following operations are partly supported in CEE Networking:

Port binding extended attributes (ports)
Only the `host_id` attribute is supported.

External networks (external-net)
Support for dynamic creation of external networks. It allows the addition of external networks towards additional BGWs in runtime.

Operations for subnets
Only IPv4 is supported.

Network provider extended attributes (networks)
Only the `vlan` network type is supported.

Administrative State Down (ports)
Administration state `down` is not supported for trunk ports and their subports.
If the administration state of a trunk port is set to `down` at the CIC, it will not take effect to the trunk port and its subports at the relevant Compute node, although Neutron will show that the trunk port and its subports are administratively down.

4.2 Added Operations

The following extension is added:

- Trunkport Extension

  The Neutron Trunkport extension allows Virtual Machines (VMs) to send VLAN tagged traffic to Neutron networks. In this way, each VLAN tag is associated with a Neutron network.

4.3 Not Supported Operations

The following operations are not supported in CEE Networking:

- Configurable external gateway modes
- Extra DHCP options (`extra-dhcp-opt`)
- The ExtraRoute Extension
- Firewall as a Service (FwaaS)
- Load Balancer as a Service (LBaaS)
- Metering labels and rules
- Security groups and rules (`security-groups`)
- Virtual Private Network as a Service (VPNaaS)

5 Ericsson Extensions

This section describes the added CEE API extensions in detail.

5.1 Trunkport Extension

Note: The trunkport extension is deprecated and will be removed in the next CEE release.

The Neutron Trunkport extension allows VMs to send VLAN tagged traffic to Neutron networks in a way that each VLAN tag is associated with a Neutron network.

5.1.1 Topology

A trunkport can be a simple trunkport or a subport.

A trunkport is a port that is connected to the VMs (representing a Network Interface Card - NIC).

Trunkports function much like a normal Neutron port.

For outgoing traffic from a VM, untagged traffic is forwarded to the network to which the trunkport is connected.

Incoming traffic from trunkports is untagged when entering the VM.

A subport is a port that is on a trunkport, and connected to a Neutron network.

For outgoing traffic from a VM, the VLAN tag is stripped off before the traffic is forwarded to the network connected to a subport. The tag from the VM is stripped before entering the associated Neutron network. When entering the associated Neutron network, the tag associated with the network is added. If the network is VxLAN based, a tunnel + key, associated with the network, is used to create the proper header.

For incoming traffic to a VM, from the network connected to a subport, the VLAN tag is added before the traffic is forwarded to a VM.

The topology of the trunkport concept is shown in Figure 1.

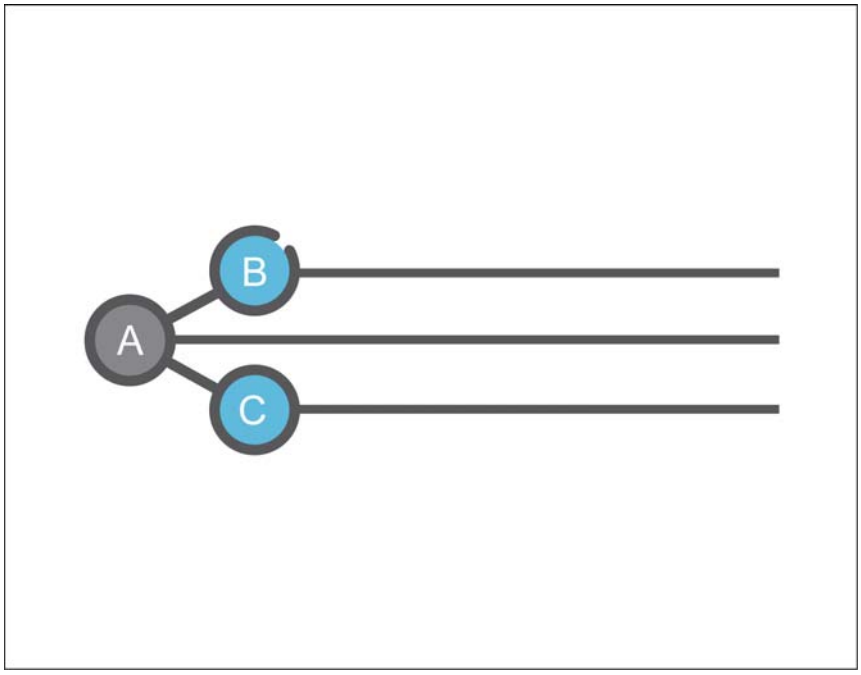

Figure 1 Trunkport Topology

A	Trunkport
B	Subport
C	Subport

5.1.2 Concepts

The `trunkport`-prefixed extended attributes for the **port** resource are shown in Table 3.

Table 3 Trunkport Attributes

Attribute	Type	Required	CRUD[(1)]	Default Value	Validation Constraints	Notes
trunkport:type	String	N/A	CR	None	Trunkport or subport	Defines whether a port is a trunkport or a subport.
trunkport:parent_id	uuid-str	N/A	CR	None	Valid `port id` of a trunkport	For subports, the id of the trunkport to which the subport is connected.
					Unset	For trunkports, unset.

Table 3 Trunkport Attributes

Attribute	Type	Required	CRUD(1)	Default Value	Validation Constraints	Notes
trunkport:vid	Integer	N/A	CR	None	1-4094	For subports, segmentation ID that is used to access the given subport from the trunkport.
					Unset	For trunkports, unset.

(1) C : Use the attribute in create operations.
R: This attribute is returned in response to show and list operations.
U: You can update the value of this attribute.
D: You can delete the value of this attribute.

5.1.3 Trunkport API Operations

This section discusses the operations for setting and retrieving the trunkport port extension attributes for port objects.

5.1.3.1 List Ports

Table 4 List Ports

Verb	URI	Description
GET	/ports	Returns a list of ports with their attributes.

Normal response code: 200

Error response code: Unauthorized (401)

This operation returns all the ports defined in Neutron to which the given user has access.

```
{
"ports": [
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"extra_dhcp_opts": [{"opt_value": "testfile.1",⇒
 "opt_name": "bootfile-name"},
```

```
{"opt_value": "123.123.123.45", "opt_name":⇒
 "server-ip-address"}, {"opt_value": "123.
123.123.123", "opt_name": "tftp-server"}],
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:52:92:3a",
"fixed_ips": [{"subnet_id": "99a8aea3-b9da-409d-a5e5-⇒
f45338ceb4d3"
, "ip_address": "172.24.4.228"}],
"id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "trunk",
"trunkport:parent_id": "",
"trunkport:vid": ""
},
{
"status": "ACTIVE",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"extra_dhcp_opts": [],
"binding:vif_type": "ovs",
"device_owner": "compute:probe",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:49:56:07",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3"⇒
, "ip_address": "172.24.4.227"}],
"id": "5212d40a-c2f5-4a5d-ad18-694658047654",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "zvm2",
"trunkport:type": "subport",
"trunkport:parent_id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"trunkport:vid": "10"
}
]
}
```


Example 1 List Ports: JSON Response

5.1.3.2 Show Port

Table 5 Show Port

Verb	URI	Description
GET	/ports/*port_id*	Returns details about a specific port including trunkport attributes.

Normal response code: 200

Error response codes: Unauthorized (401), Not Found (404)

This operation returns the port attributes of a port specified in the request URI. These attributes include the trunkport attributes.

```
{
"port":
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"extra_dhcp_opts": [
{"opt_value": "testfile.1","opt_name": "bootfile-name"},
{"opt_value": "123.123.123.123", "opt_name": "tftp-server"},
{"opt_value": "123.123.123.45", "opt_name": "server-ip-address"}
],
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:52:92:3a",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.228"}],
"id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "trunk",
"trunkport:parent_id": "",
"trunkport:vid": ""
}
}
```

Example 2 Show Port with Trunkport Attributes: JSON Response

5.1.3.3 Create Port

Table 6 Create Port

Verb	URI	Description
POST	/ports	The operation creates a new port, and the supplied attributes show whether the port is a trunkport or a subport.

Normal response code: 200

Error response code: Unauthorized (401)

The operation creates a new port, and the supplied attributes show whether the port is a trunkport or a subport.

Note: When creating a subport, `mac_address` is copied from the parent port.

```
{
"port":
{
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"admin_state_up": true,
"trunkport:type": "trunk"
}
}
```

Example 3 Create Port with Trunkport Attributes: JSON Request

```
{
"port":
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:43:3c:b7",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"id": "055d27c0-0194-4782-be45-275ff2c95c61",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "trunk",
"trunkport:parent_id": "",
"trunkport:vid": ""
}
}
```


Example 4 Create Port with Trunkport Attributes: JSON Response


```
{
"port":
{
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"admin_state_up": true,
"trunkport:type": "subport",
"trunkport:parent_id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"trunkport:vid": "10"
}
}
```


Example 5 Create Port with Trunkport Attributes (Type Subport): JSON Request

```
{
"port":
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:43:3c:b7",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"id": "055d27c0-0194-4782-be45-275ff2c95c61",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "subport",
"trunkport:parent_id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"trunkport:vid": "10"
}
}
```


Example 6 Create Port with Trunkport Attributes (Type Subport): JSON Response

5.1.3.4 Update Port

No attributes can be updated.

5.1.3.5 Delete Port

When deleting a trunkport, the subports that are connected to it are also deleted.

6 Limitations

This section lists the limitations and recommendations for CEE Networking.

6.1 Limitations

This section contains the limitations of CEE Networking.

6.1.1 Segmentation IDs

The total number of segmentation IDs is limited to 4094, see Section 2.2 on page 2.

6.1.2 Deleting Trunkport and Subports

If a trunk port is deleted before the associated subports have been deleted, there are remains left in Neutron that prevent using the same IP address with a different MAC number. If a new trunk port and subports are created with the same IP address, this can prevent VMs on the ports to get an IP address.

To delete a trunk port with subports, do the following:

1. Delete the related subports.
2. Delete the trunk port.

7 Concepts and Use Cases

The following sections describe the various concepts and use cases that are connected to the CEE Networking API.

7.1 General Terms

Neutron Network L2 broadcast domain

Neutron Subnet Definition of a subnet. It s attached to a Neutron network. Used to configure the Neutron DHCP service.

7.2 IP Address Management

The following three methods are available for IP address management:

- Application handles the IP addresses on its own and do not involve Neutron.

  Note: IP addresses can be privately handled only if ARP spoofing is switched off. It is not recommended to switch off ARP spoofing, as it is a security feature of the system.

  No Neutron DHCP service is used. Only L2 services from Neutron can be used.

  Note: Nova requires a subnet attached to a Neutron network in order to start VMs on the network. User is required to create a dummy subnet with DHCP disabled as a workaround.

- Application uses Neutron for IP address management.

  This can be done in several ways: the application can specify IP addresses per ports or only specify IP subnets and let Neutron allocate the addresses.

- A mix of the above two.

  In this case, it is the responsibility of the application to align the configuration view of the application with the configuration view of Neutron.

IP addresses in Neutron can be reused on different Neutron networks. IP addresses known by Neutron must have matching Neutron subnets.

To manually assign an IP address to a port use the `fixed_ips` attribute in the Neutron port when creating the Neutron port.

The `fixed_ips` attribute includes `ip_address` and `subnet_id`. Please refer to the section "Networking API v2.0" in the *OpenStack API Complete Reference*.

7.2.1 Neutron DHCP

Neutron DHCP can be enabled or disabled per subnet.

This is done by using the `enable_dhcp` attribute on the Neutron subnet.

```
enable_dhcp <True/False>
```

7.3 MAC Address Management

The following two methods are available for MAC address management:

- The application handles MAC addresses on its own, and provides the MAC address to Neutron when creating the Neutron port.
- The application allows Neutron to allocate a MAC address when creating the Neutron port.

MAC addresses have to be unique per Neutron network, but can be reused on different Neutron networks.

To manually assign a MAC address use the `mac_address` attribute in the Neutron port when creating the Neutron port, refer to the sections "Networking API v2.0" and "Networking API v2.0 extensions" in the *OpenStack API Complete Reference*.

Neutron-allocated MAC addresses have a three byte fixed prefix. The rest will be a random number unique per Neutron network. Refer to *Configuration File Guide*, for the prefix used in the Neutron configuration file.

Note: It is advisable to use the local administered MAC ranges.

7.4 IPv6

Network Routing or Address Management for IPv6 is not supported.

7.5 Internal Neutron Network

This section describes how to connect VMs using internal L2. Create Neutron network, create Neutron ports on Neutron network, and then create VMs using those Neutron ports.

1. Create Neutron network

For the procedure of creating a Neutron network, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

2. Create Neutron subnet on Neutron network

   Nova requires a subnet attached to a Neutron network in order to start VMs on the network. User is required to create a dummy subnet with DHCP disabled as a workaround.

   For the procedure of creating a Neutron subnet on Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

3. Create Neutron ports on Neutron network

   For the procedure of creating Neutron ports on a Neutron network, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the*OpenStack API Complete Reference*.

4. Create VMs using Neutron ports

7.6 Internal Neutron Network with Neutron IP Address Management

This chapter describes how to connect VMs using internal L3.

1. Create Neutron network

   For the procedure of creating a Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

2. Create Neutron subnet on Neutron network

   For the procedure of creating a Neutron subnet on Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

3. Create Neutron ports on Neutron network

   For the procedure of creating Neutron ports on a Neutron network, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

4. Create VMs using Neutron ports.

7.7 Networks to CMX VRs

The following subsections describe how to connect VMs to CMX VRs by using an L2 network, in the case of `ericsson_cmx` configuration of CEE networking.

The L2 BGW connection is shown in Figure 2.

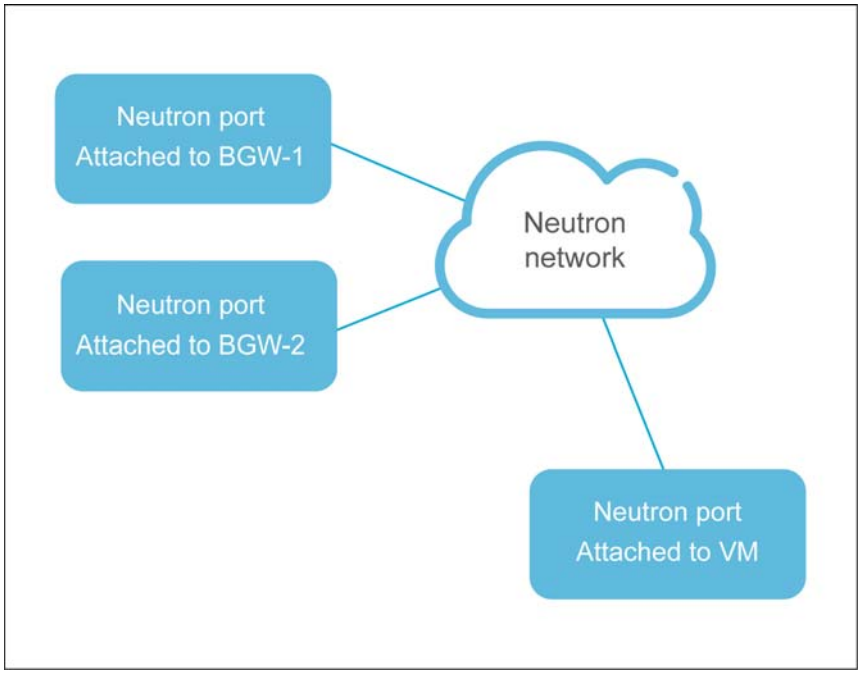

Figure 2 L2 BGW Connection

7.7.1 Create Neutron Network

Create a Neutron network using the `provider` network extension (provider segmentation ID). The cloud administrator is required for the management of these IDs. The IDs must be picked from the provider BGW pool, see Section 2.2 on page 2.

For the procedure of creating a Neutron network, refer to the section “Networking API v2.0” in the *OpenStack API Complete Reference*.

For the `provider` Extended Attributes for Networks, refer to the section “Networking API v2.0 extensions” in the *OpenStack API Complete Reference*.

```
provider:segmentation_id      <segmentation ID>
provider:network_type         vlan
provider:physical_network     default
```

7.7.2 Create Neutron Subnet

Create Neutron subnet on Neutron network. A subnet is required regardless if IP address management is going to be handled by Neutron or not.

nova requires a subnet attached to a Neutron network in order to start VMs on the network. User is required to create a dummy subnet with DHCP disabled

as a workaround in cases when IP address management is handled by the application.

For the procedure of creating a Neutron subnet on Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

7.7.3 Configure CMX

To configure CMX, refer to the following documents:

- BSP External Network Connectivity, Reference [1]
- BSP Tenant Network Connectivity, Reference [2]
- Create Control Network, Reference [3]

Note: L3 connectivity is not supported as a service from CEE on BSP. To set up L3 connectivity on BSP, refer to the documents mentioned in this section.

Reference List

[1] *BSP External Network Connectivity*, 2/1553-APP 111 01

[2] *BSP Tenant Network Connectivity*, 3/1553-APP 111 01

[3] *Create Control Network*, 23/1543-APR 901 0549/1