User's Manual

IxNetwork REST API

KEYSIGHT TECHNOLOGIES

Notices

Copyright Notice

Published by:

Keysight Technologies, Inc.
1400 Fountaingrove Parkway
Santa Rosa, CA 95403

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Declaration of Conformity

Declarations of Conformity for this product and for other Keysight products may be downloaded from the Web. Go to http://www.keysight.com/go/conformity and click on “Declarations of Conformity.” You can then search by product number to find the latest Declaration of Conformity.

U.S.Government Rights

The Software is “commercial computer software,” as defined by Federal Acquisition Regulation (“FAR”) 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement (“DFARS”) 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at http://www.keysight.com/find/sweula. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

Warranty

THE MATERIAL CONTAINED IN THIS DOCUMENT IS PROVIDED “AS IS,” AND IS SUBJECT TO BEING CHANGED, WITHOUT NOTICE, IN FUTURE EDITIONS. FURTHER, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KEYSIGHT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED WITH REGARD TO THIS MANUAL AND ANY INFORMATION CONTAINED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT AND THE USER HAVE A SEPARATE WRITTEN AGREEMENT WITH WARRANTY TERMS COVERING THE MATERIAL IN THIS DOCUMENT THAT CONFLICT WITH THESE TERMS, THE WARRANTY TERMS IN THE SEPARATE AGREEMENT WILL CONTROL.

CAUTION

A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met

TABLE OF CONTENTS

1. Overview: 1

2. Configure BGP through GUI: 3

2.1 Add Chassis and Lock Ports: 3

2.2 Add Topology: 4

2.3 Emulate a Protocol: 5

2.4 Device Group Multiplier: 6

2.5 Edit Protocol Grid: 7

2.6 Configure BGP: 7

2.7 Add Network Group: 8

2.8 Start Protocols: 9

2.9 Configure Traffic: 9

2.10 Add Endpoints to Traffic: 10

2.11 Edit Packet and Setup Flow Groups: 11

2.12 Setup Frame Size and Rate: 12

2.13 Setup Flow Tracking and Protocol Behavior: 13

2.14 Validate Traffic: 14

2.15 Apply Traffic, Start Traffic, and Statistics View: 15

3. Configure BGP through Automation (REST API): 16

3.1 Initialize Environment: 16

3.2 Add Chassis and Lock Ports: 16

3.3 Create Topology: 18

3.4 Create Device Group: 20

3.5 Create Ethernet Stack: 21

3.6 Create Ipv4 Stack: 24

3.7 Create BGP: 28

3.8 Create Network Group: 36

3.9 Start Protocols: 39

3.10 Configure Traffic: 40

3.11 Start Traffic and Stop Traffic: 43

3.12 Disconnect: 44

4. Configure BGP through Automation(Python-OpenIxia): 45
4.1 Initialize Environment: 45
4.2 Add Chassis and Lock Ports: 45
4.3 Create Topology: 46
4.4 Create DeviceGroup: 47
4.5 Create Ethernet Stack: 48
4.6 Create Ipv4 Stack: 49
4.7 Create BGP: 50
4.8 Create Network Group: 51
4.9 Start Protocols: 52
4.10 Configure Traffic: 52
4.11 Start Traffic and Get Statistics: 53
5. To Know More on REST API: 54
6. Support: 54

1. Overview:

- ✓ REST Stands for Representational State Transfer (REST)
- ✓ First introduced by Roy Fielding in 2000.
- ✓ REST is a web service-based architecture.
- ✓ Uses HTTP Protocol for data communication.
  - ➢ Example: http://10.154.162.75:11009/api/v1/sessions/1/ixnetwork/topology/1
- ✓ REST uses various representations to represent a resource like: Text, JSON, YAML and XML.

REST Services:

- ✓ GET: Provides a read only access to a resource.
- ✓ POST: Used to create a new resource or to perform an operation.
- ✓ PATCH: Used to update an existing resource.
- ✓ OPTIONS: Used to get additional resources and descriptions.
- ✓ DELETE: Used to remove a resource.

REST resource:

- ✓ REST architecture treats every content as a resource.
- ✓ REST server simply provides access to resources.
- ✓ REST client accesses and modify the resources.
- ✓ REST server returns resources in JSON format for viewing or for modifying.
  - ➢ Example: http://10.154.162.75:11009/api/v1/sessions/1/ixnetwork/topology/1


```
{
    "id": 1,
    "name": "Topo1",
    "descriptiveName": "Topo1",
    "vports": [
        "/api/v1/sessions/1/ixnetwork/vport/1"
    ],
    "status": "notStarted",
    "errors": [],
    "portCount": 1,
    "links": [
        {
            "rel": "self",
            "method": "GET",
            "href": "/api/v1/sessions/1/ixnetwork/topology/1"
        },
        {
            "rel": "meta",
            "method": "OPTIONS",
            "href": "/api/v1/sessions/1/ixnetwork/topology/1"
        }
    ]
}
```


Fig 1.1 JSON format

HTTP status codes:

- ✓ REST API server responds with a status code to the client on every request.
- ✓ Status code could also mean that the server is unreachable.
- ✓ Each status code represents the type of success or failure.
  - ✓ 200: OK.
  - ✓ 201: Successfully created.
  - ✓ 204: No content. When response body is empty. For example, a DELETE request.
  - ✓ 304: Not modified.
  - ✓ 400: Bad request. Bad input is provided.
  - ✓ 401: Unauthorized. States that the user is using an invalid or wrong authentication token.
  - ✓ 403: Forbidden. User is not having access to the method being used.
  - ✓ 404: Not found. The method is not available.
  - ✓ 409: Conflict.
  - ✓ 500: Internal server error. Server has thrown some exception while executing the method.

2. Configure BGP through GUI:

This section provides a walk-through of a scenario, which configures BGP emulation manually to get the user introduced with most of the basic features of NGPF.

2.1 Add Chassis and Lock Ports:

- ✓ The Port Selection window allows you to manage the ports.

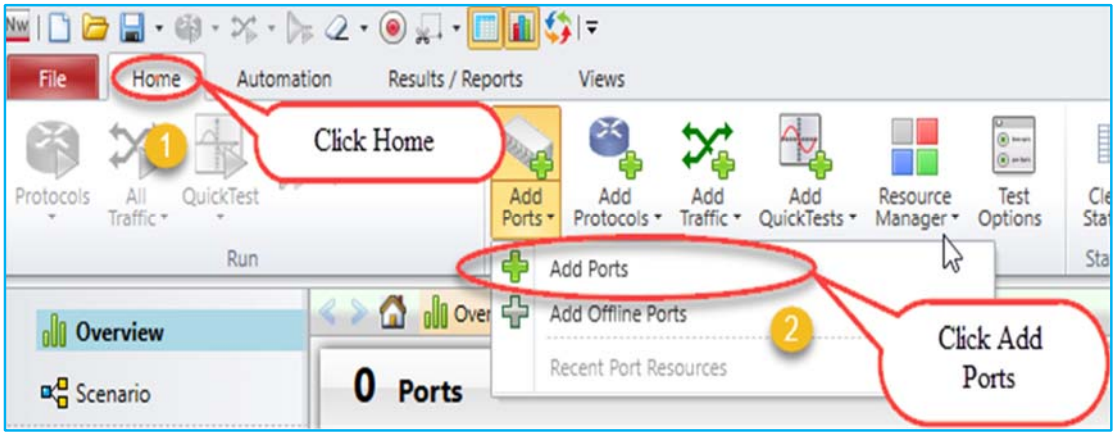


Fig 2.1 Selecting Ports

- ✓ Select chassis by entering chassis IP or select chassis from the list of recently used chassis and then click Connect all checked to add them to the configuration.

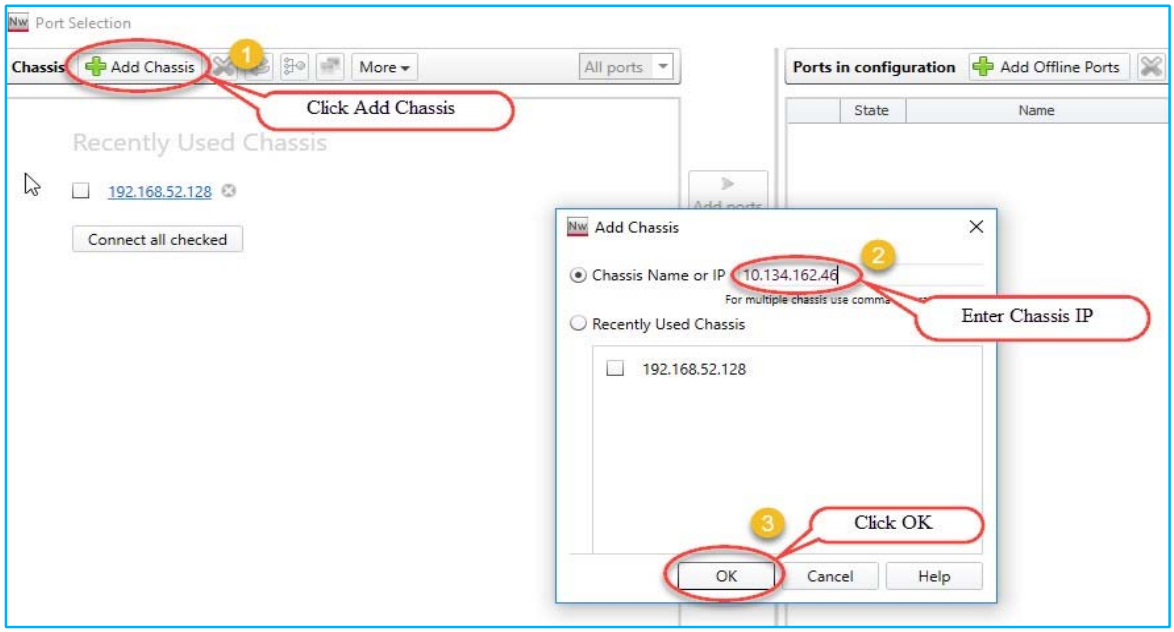


Fig 2.1.1 Adding chassis IP

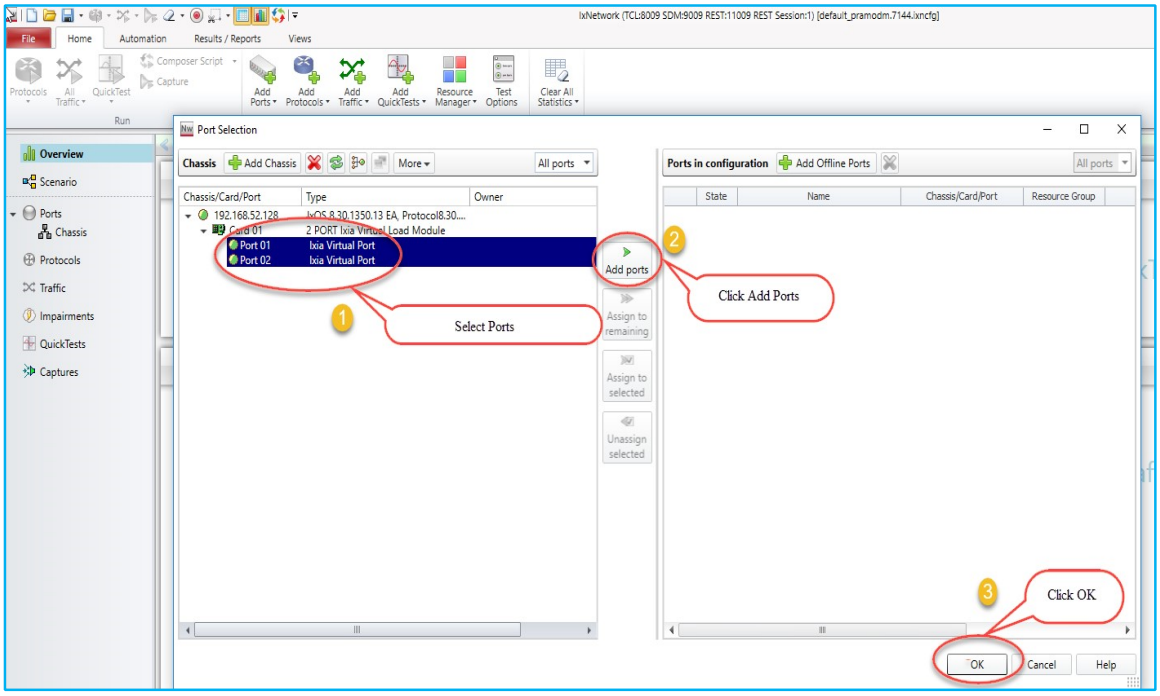


Fig 2.1.2 Port selection

2.2 Add Topology:

- ✓ An **IxNetwork** instance supports one Scenario, which can contain multiple topologies. Each Topology is a collection of one or more test ports. Each Topology is bound to a virtual port and that virtual port in-turn binds physical port.

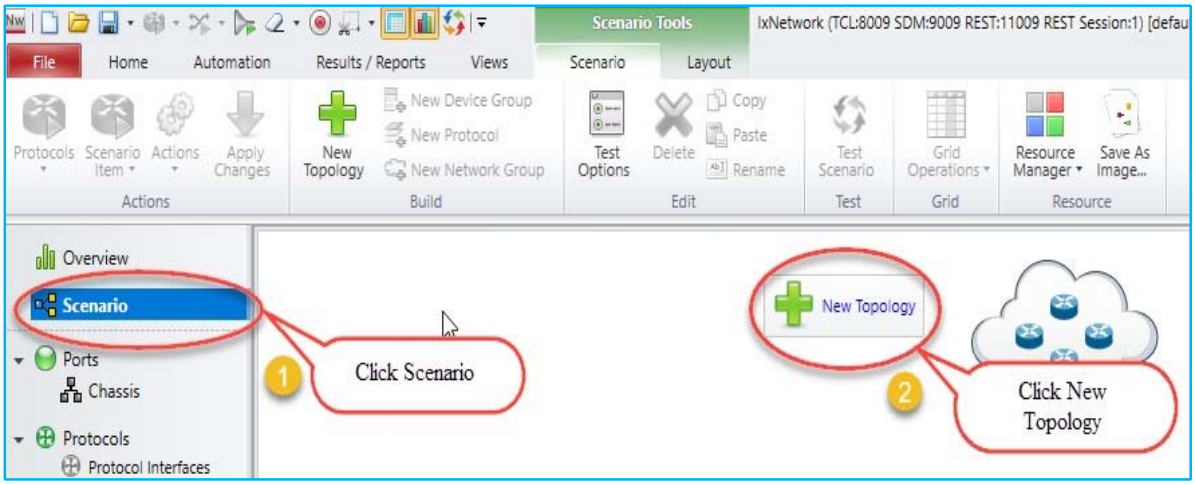


Fig 2.1.3 Adding New Topology

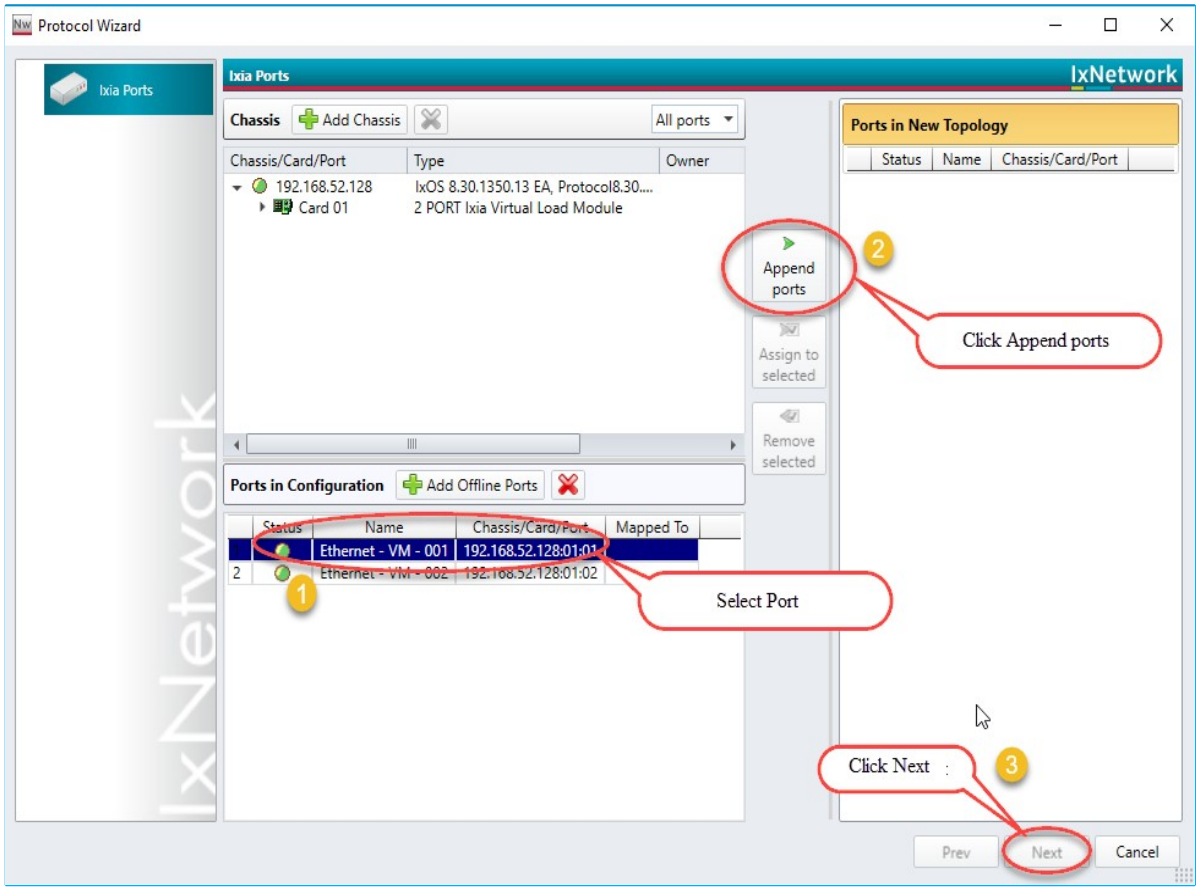


Fig 2.2 Topology with selected ports

2.3 Emulate a Protocol:

- ✓ The **Protocols** page in the **Protocol Wizard** allows you to select the protocols in the Topology.
  - ✓ The **Protocols** page lists the available **Classic** and **NextGen** protocols under the respective tabs.
  - ✓ Click **NextGen**, and then select the required protocol for the test.

- ✓ Presents all supported protocols in Next Generation Protocol Framework in a single window.

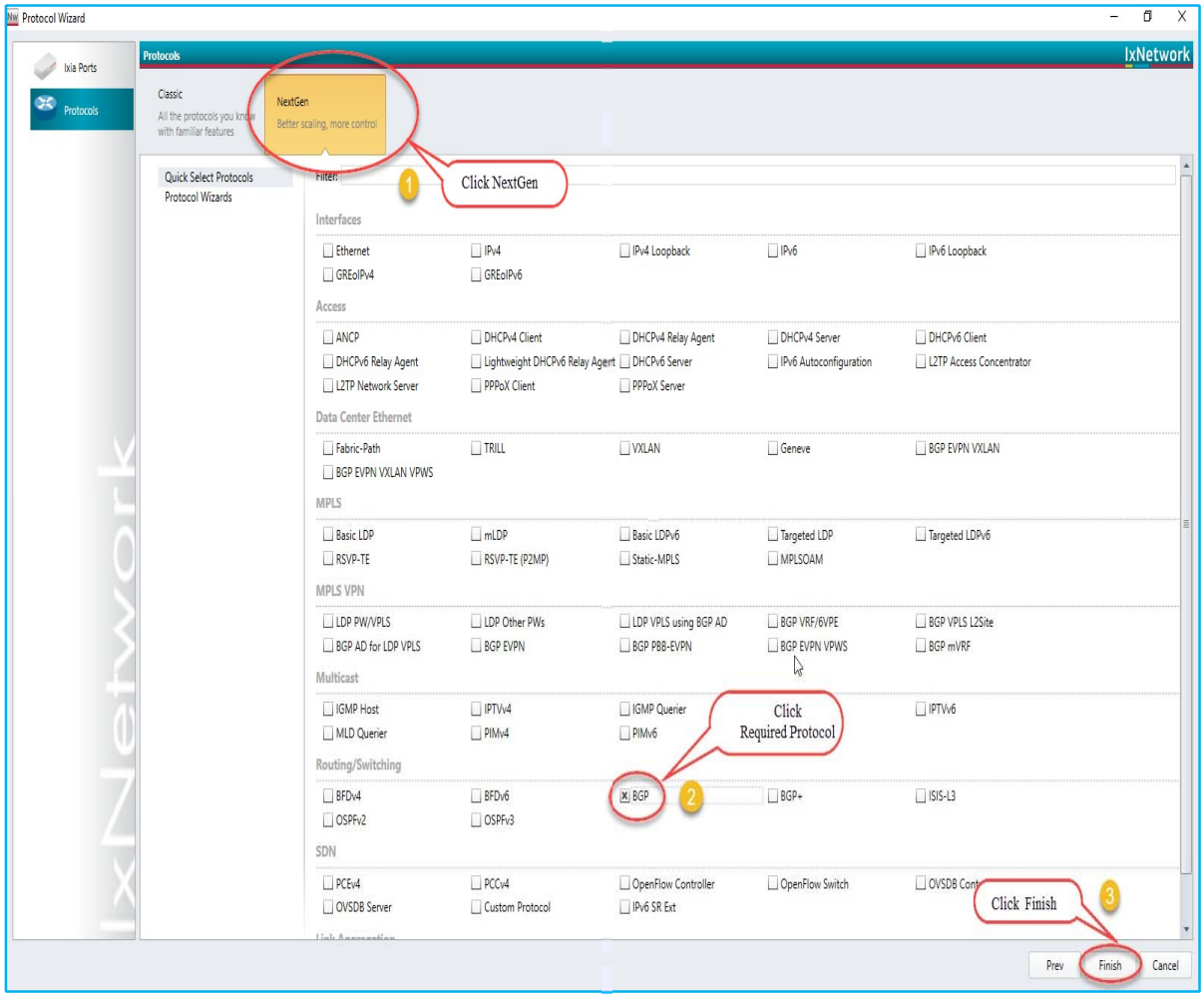


Fig 2.3 Selected Protocol BGP

2.4 Device Group Multiplier:

- ✓ A Device Group has similar Devices per test port. A Device can be a router, host, switch, and so on. It can run multiple protocols and protocol stacks.

- ✓ A Device Group count is the number of instances in the group. A configuration can be scaled by modeling a group of *n* Devices per test port by changing the multiplier.

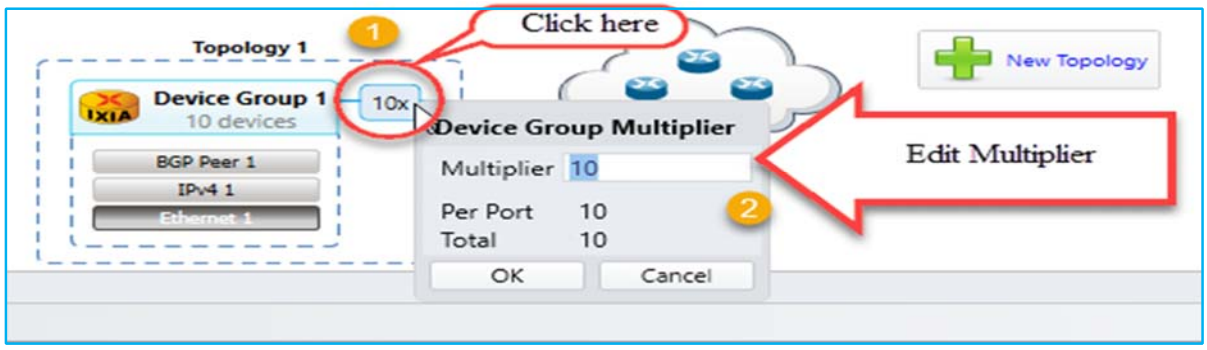


Fig 2.4 Emulate number of devices by using device group multiplier

2.5 Edit Protocol Grid:

- ✓ The protocol stack shown in the Scenario view is interactive. Click any protocol stack and edit the values according to the requirement.

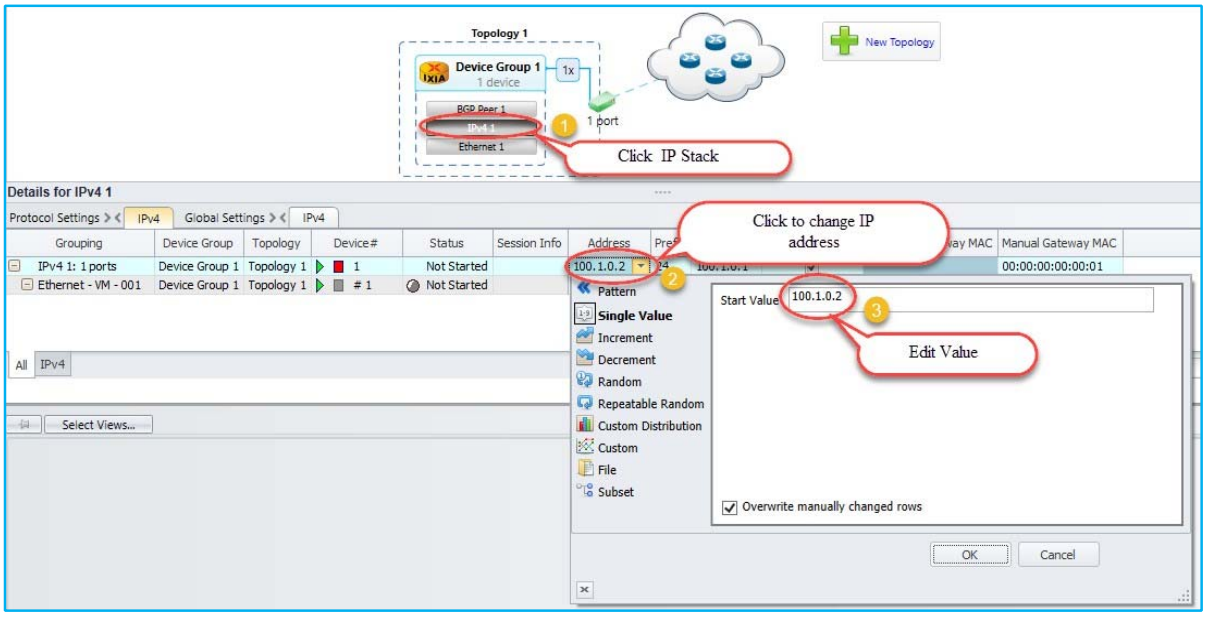


Fig 2.5 Configuring interface's

2.6 Configure BGP:

- ✓ Configure Interface IP Address to 20.20.20.2 and Gateway Address to 20.20.20.1 in device group 1 IP stack by using method 2.5. Configure Interface IP Address to 20.20.20.1 and Gateway Address to 20.20.20.2 in device group 2 IP stack using by method 2.5

- ✓ Similarly, configure Local IP to 20.20.20.2 and DUT IP to 20.20.20.1 in BGP Stack in device group 1 by clicking the BGP stack. Configure Local IP to 20.20.20.1 and DUT IP to 20.20.20.2 in BGP Stack in device group 2 by clicking the BGP stack.

2.7 Add Network Group:

- ✓ A Network Group represents a set of L3 networks (sub-netted or switched) with optional information explaining the reachability to each of these networks.

- ✓ All Devices connected to a Network Group must belong to one of the networks modeled by that Network Group.

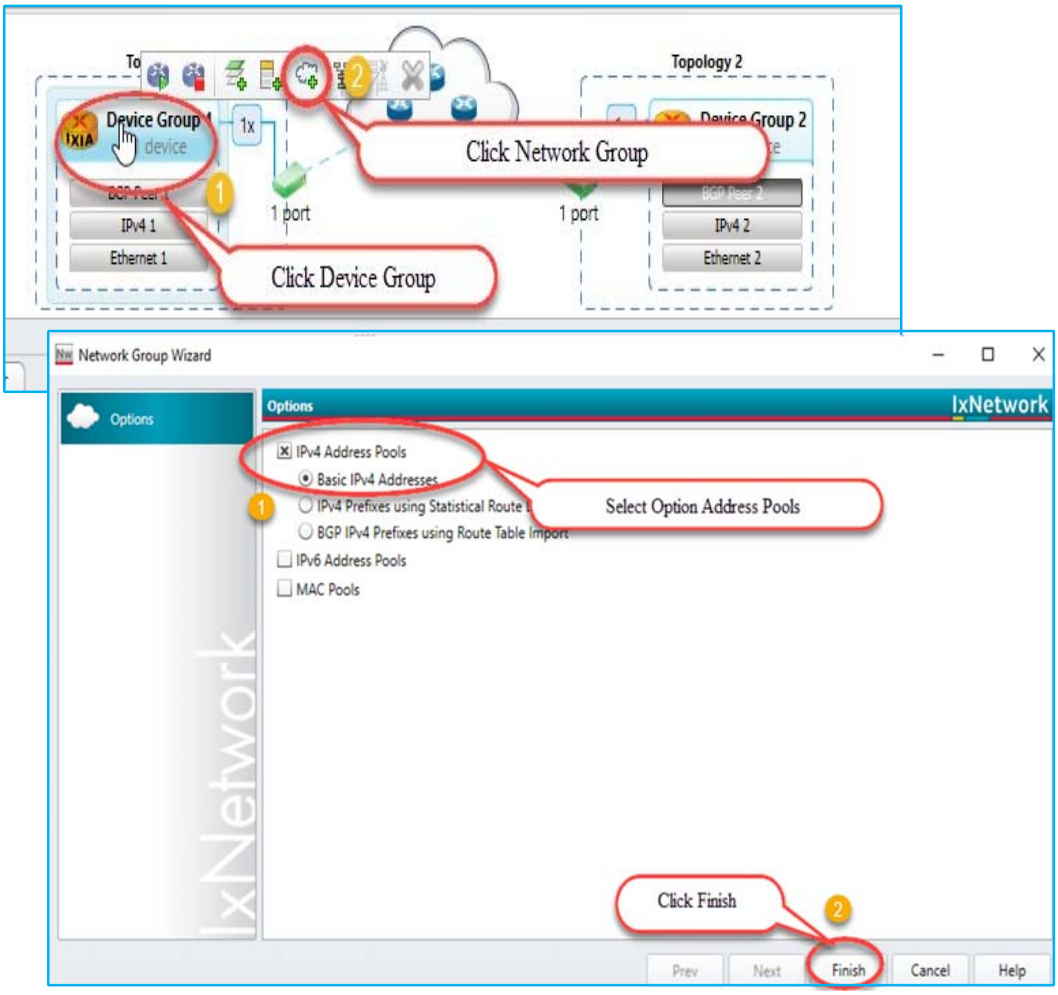


Fig 2.7 Route Profile addition by using network groups

2.8 Start Protocols:

- ✓ Click Start All to start all the protocols configured in the test.

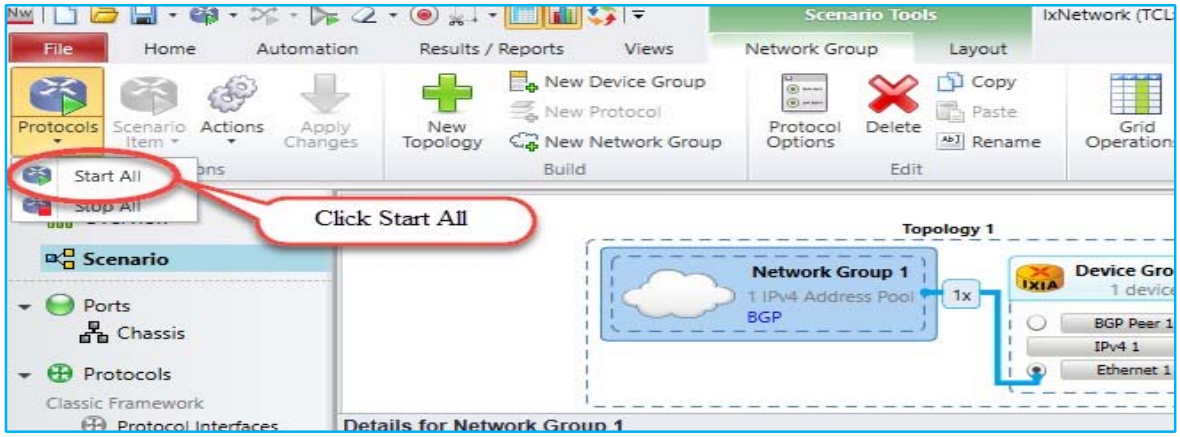


Fig 2.8 Brings up all protocol stacks

2.9 Configure Traffic:

- ✓ Traffic Wizard helps to integrate the options for traffic configuration in the control plane and data plane of IxNetwork, thereby facilitating quick setup of large scale testing.

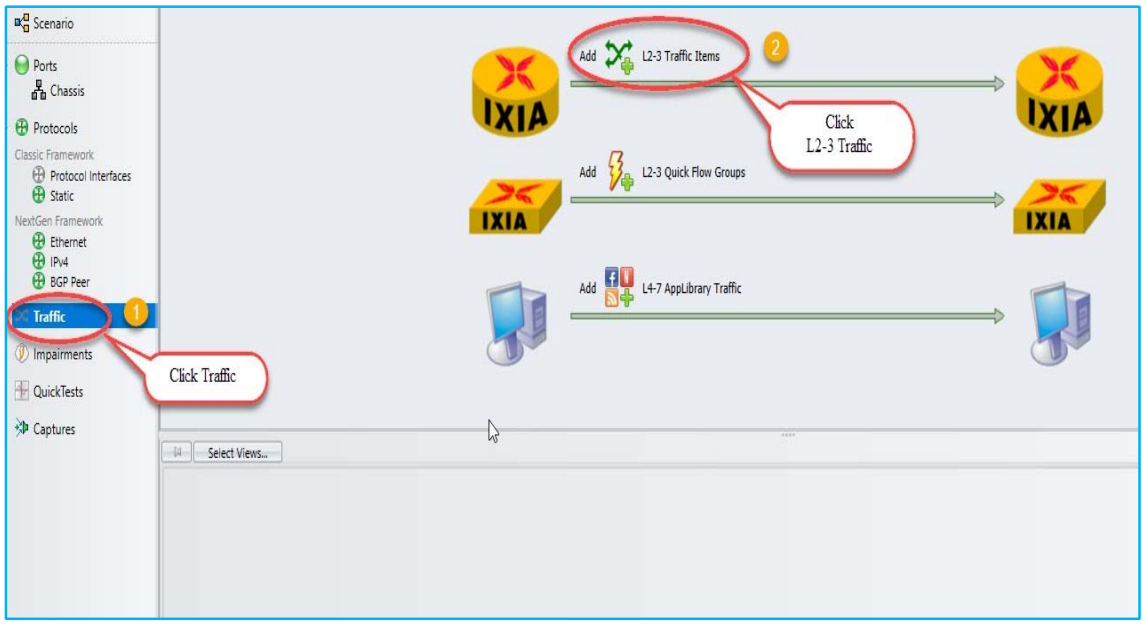


Fig 2.9 Configures L2-3 traffic item

2.10 Add Endpoints to Traffic:

- ✓ The Endpoints dialog box is the first dialog box in a series that form the Advanced Traffic Wizard. To access the Endpoints dialog box, click the Endpoints tab in the left pane of Advanced Wizard window.

- ✓ The Endpoints dialog box shows the options to select the traffic endpoints.

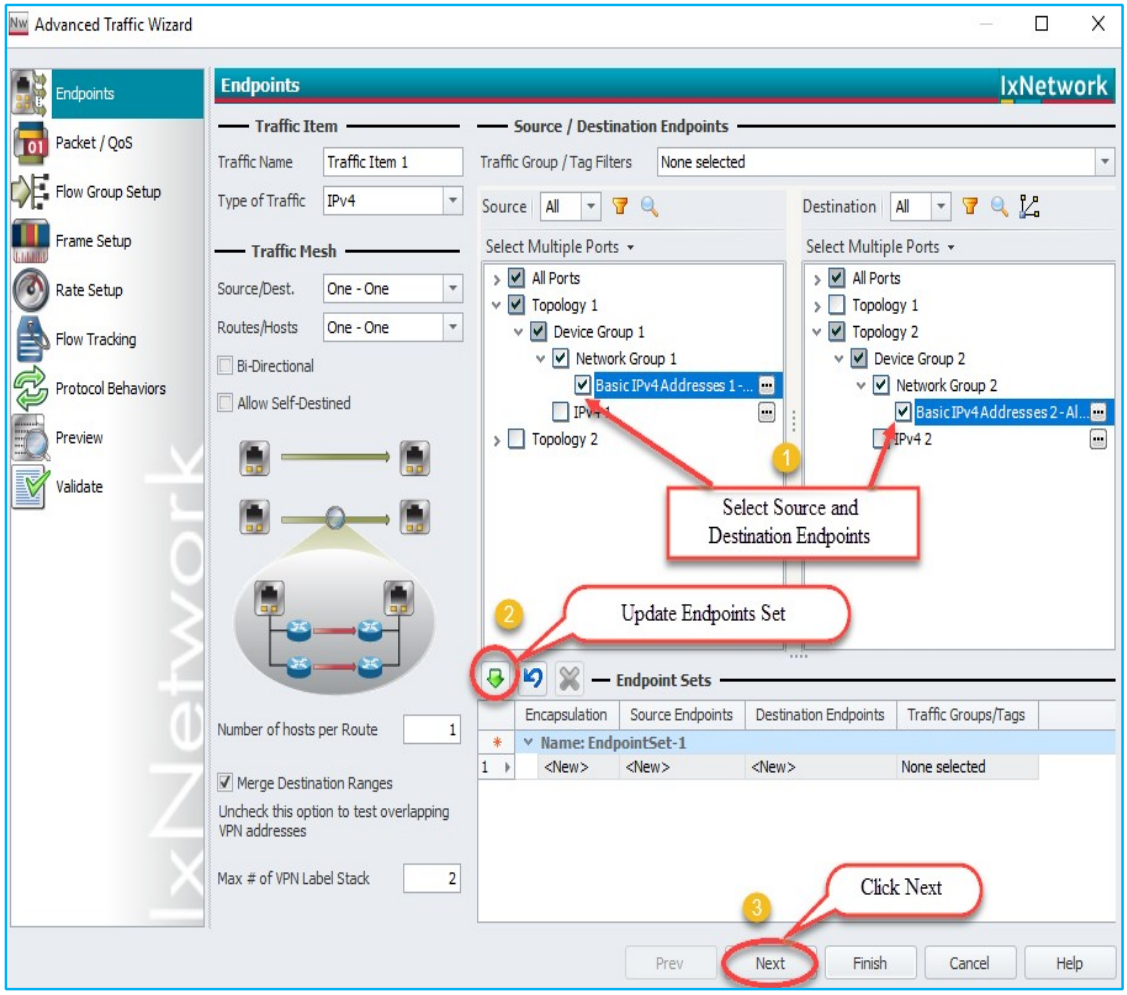


Fig 2.10 Configures source and destinations endpoints set

2.11 Edit Packet and Setup Flow Groups:

- ✓ Editing the packet and setting up flow groups is option

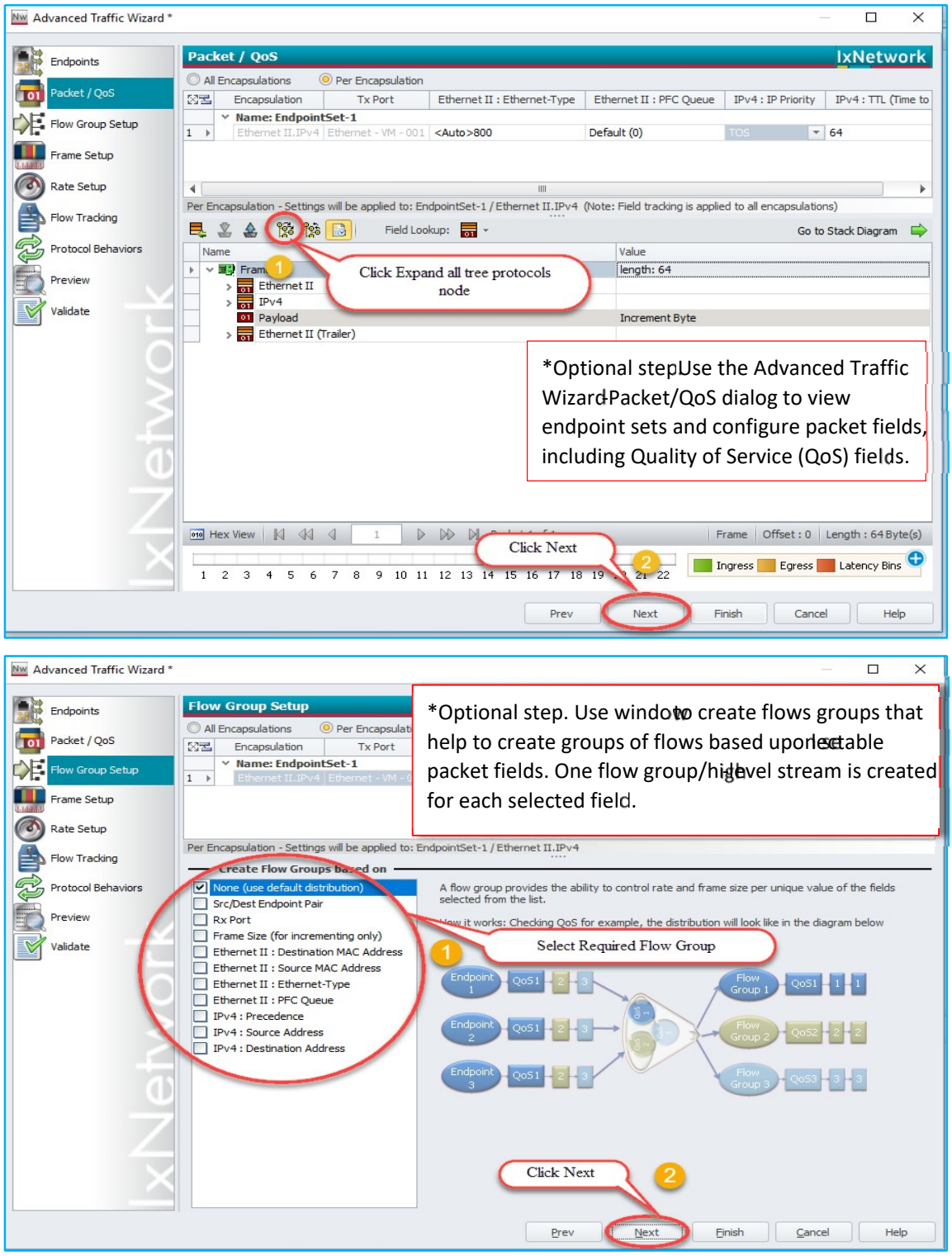
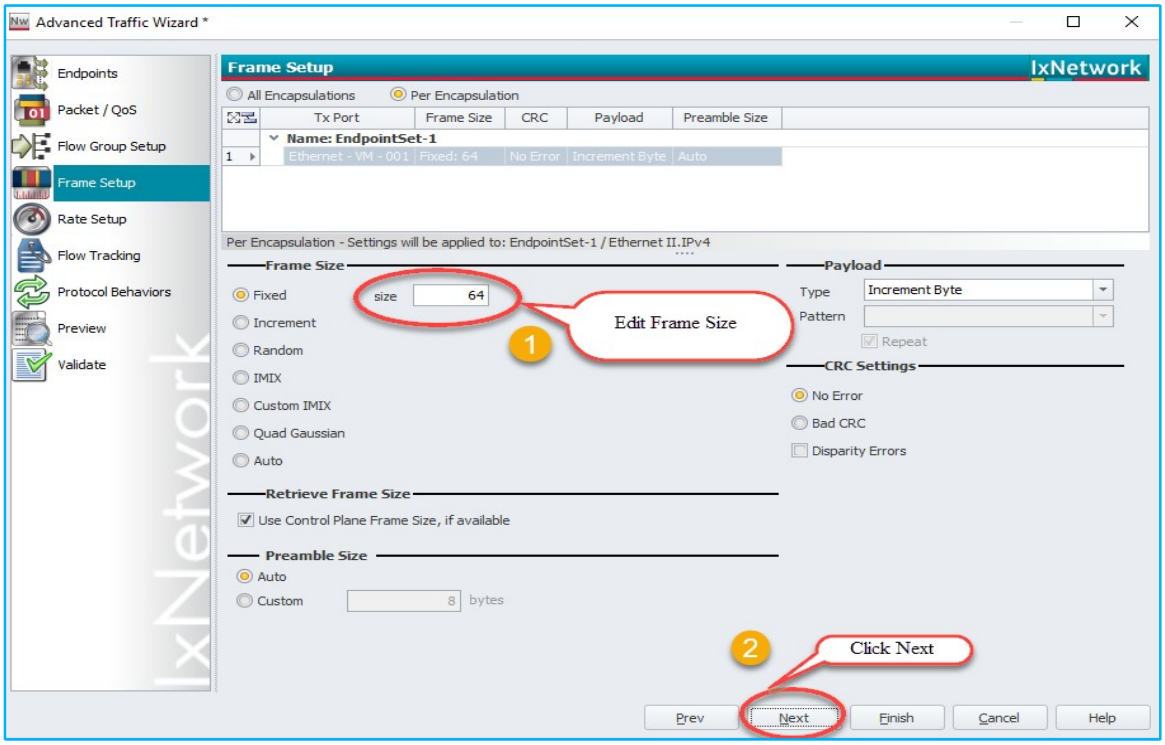


Fig 2.11 Customizing the packet and creating flow groups

2.12 Setup Frame Size and Rate:

✓ Setting up the frame Size and Line rate is optional.



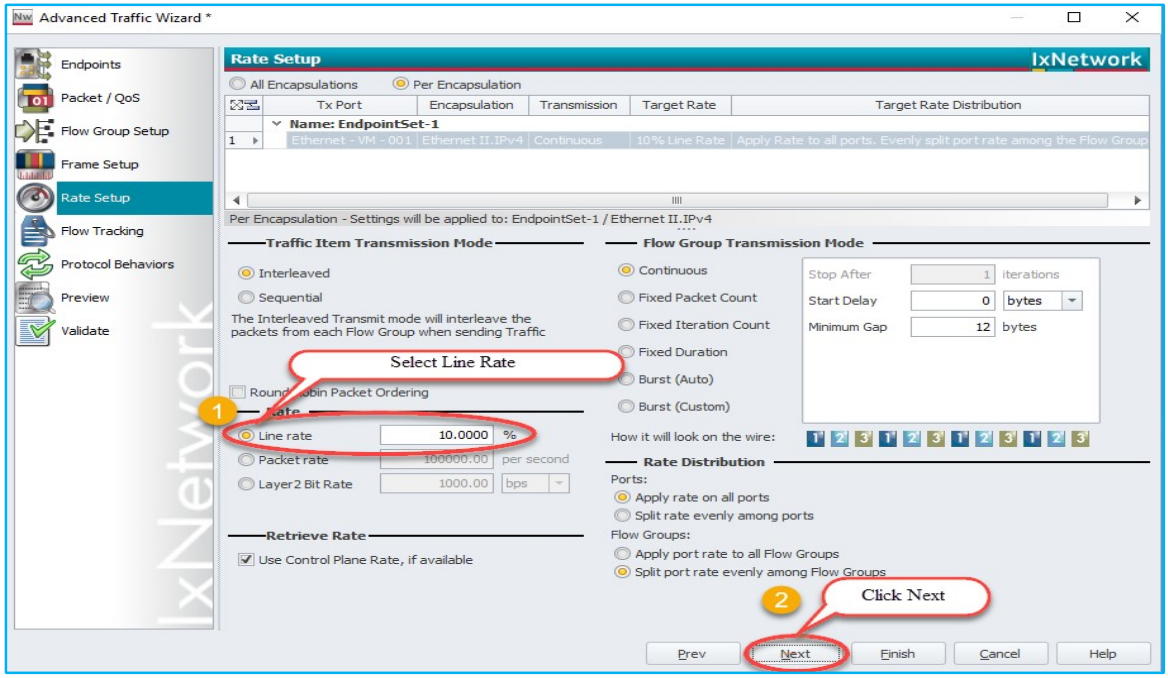
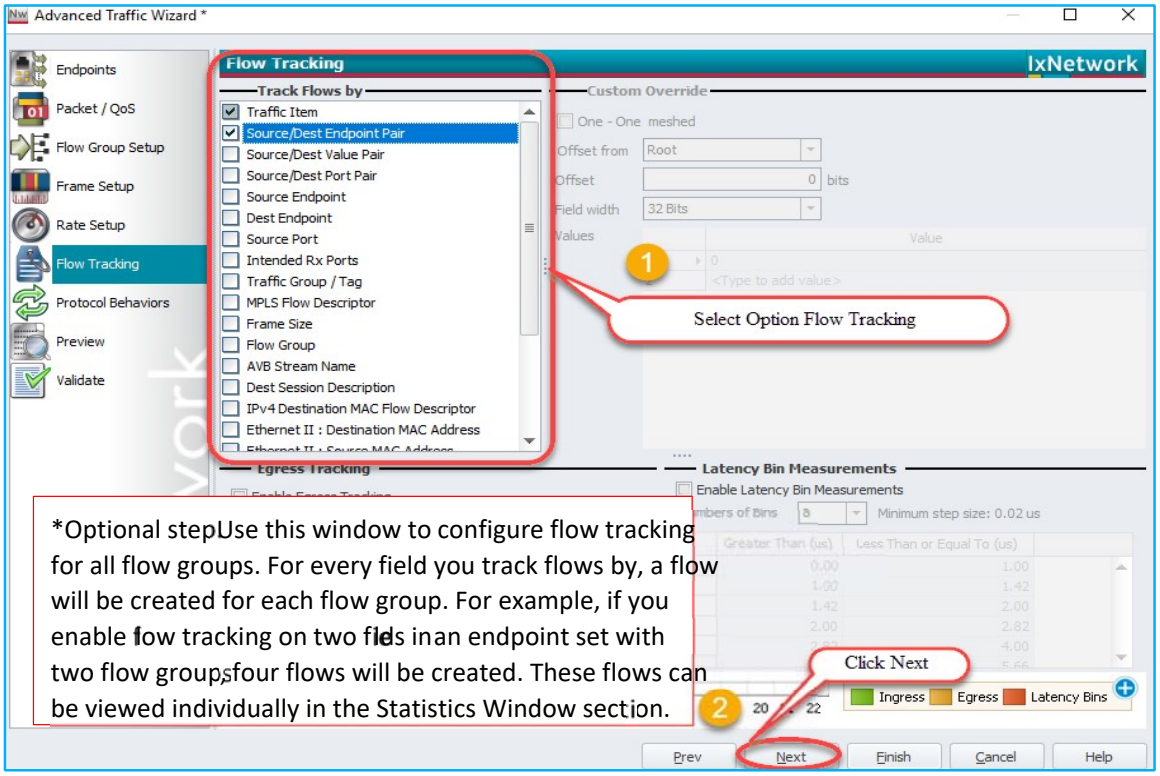


Fig 2.12 Setting up the frame size and line rate of the traffic

2.13 Setup Flow Tracking and Protocol Behavior:

✓ Setting up the flow tracking and Protocol Behavior is optional.



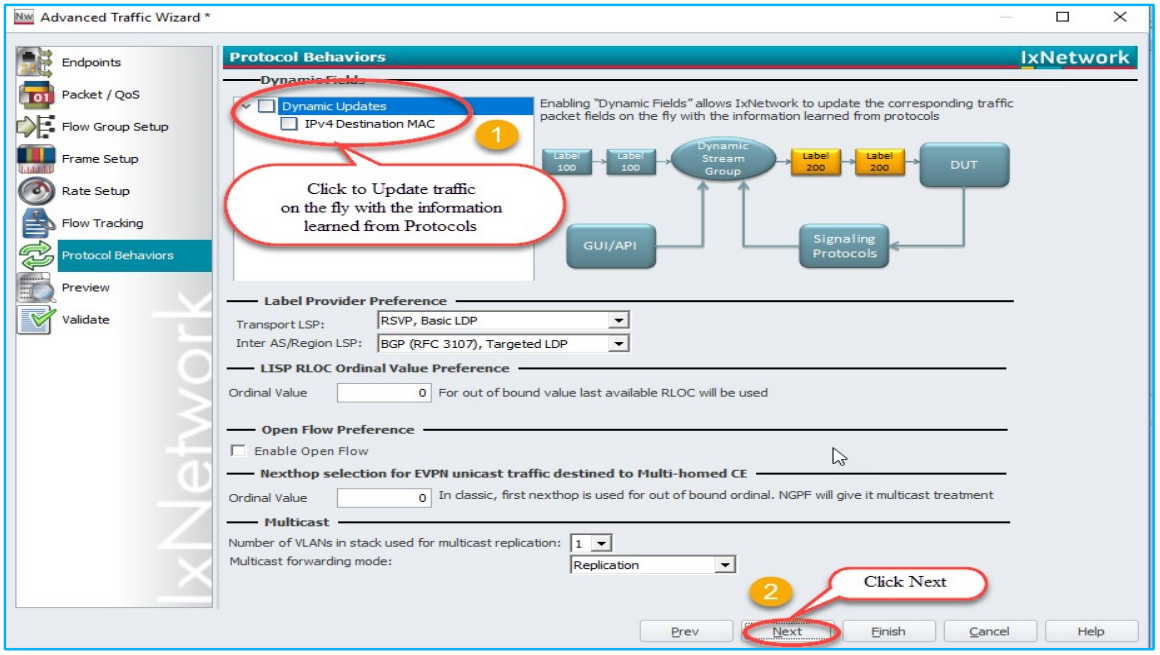
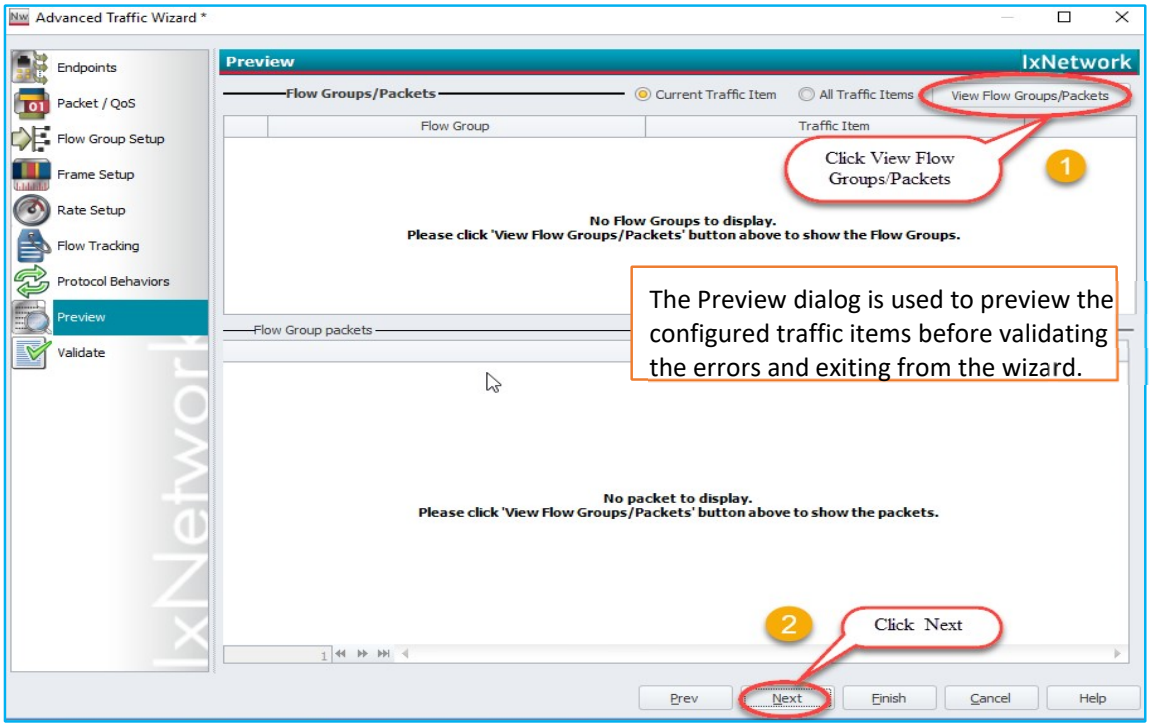


Fig 2.13 Setting up the flow tracks and traffic update option

2.14 Validate Traffic:



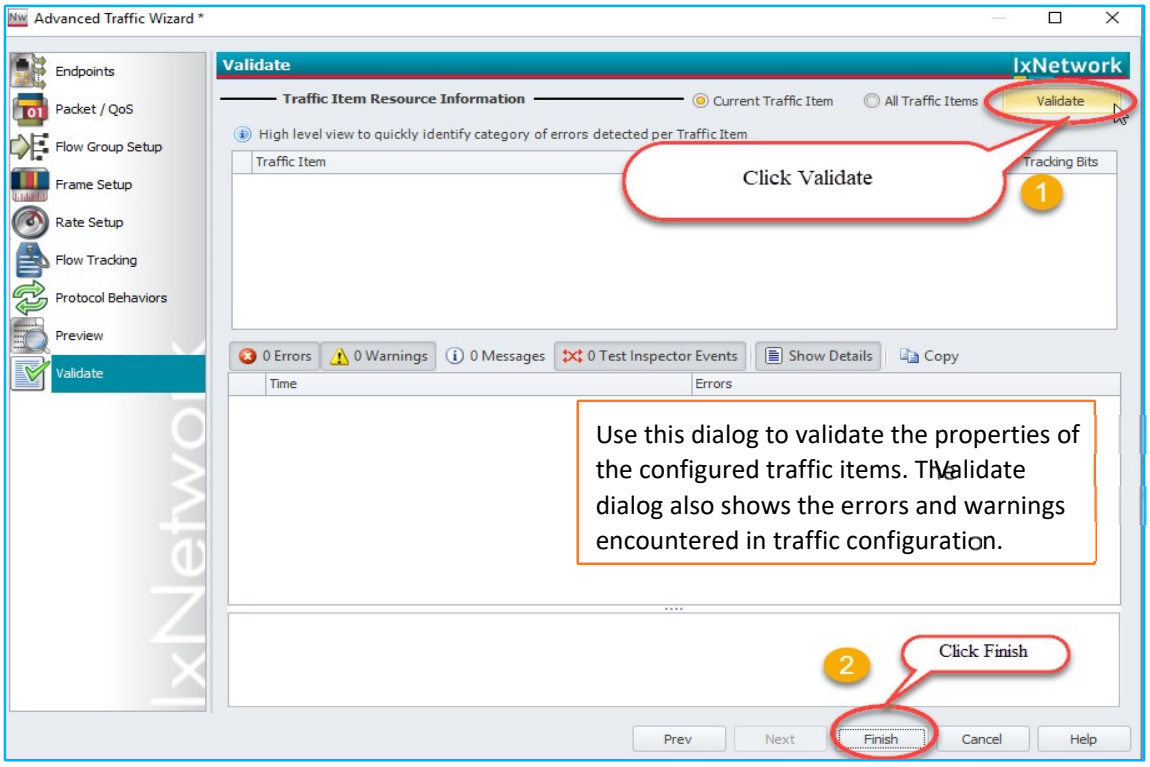


Fig 2.14 Viewing the flow groups and validating the traffic

2.15 Apply Traffic, Start Traffic, and Statistics View:

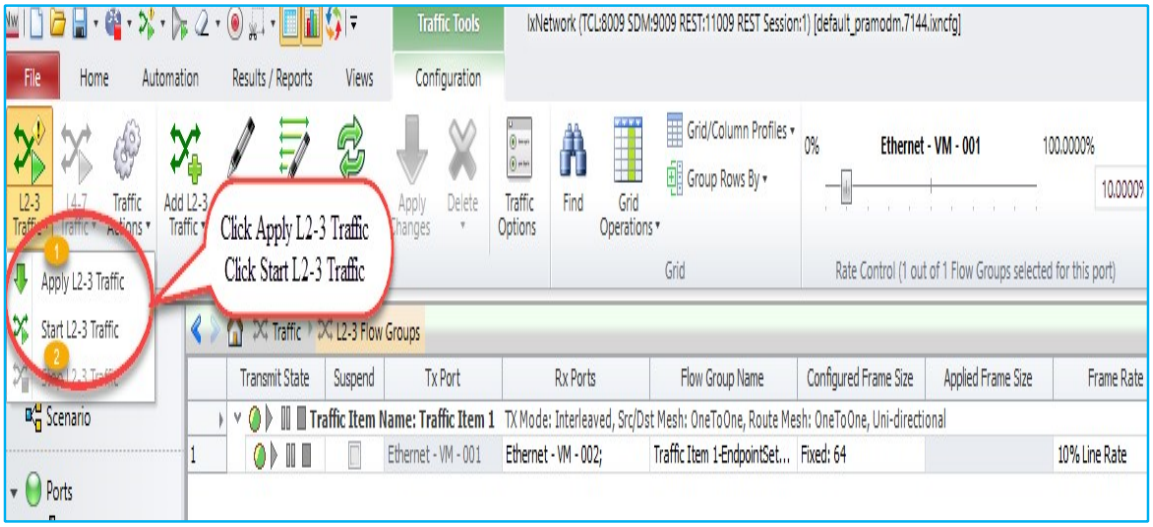


Fig 2.15 Applying and Starting L2-3 Traffic items

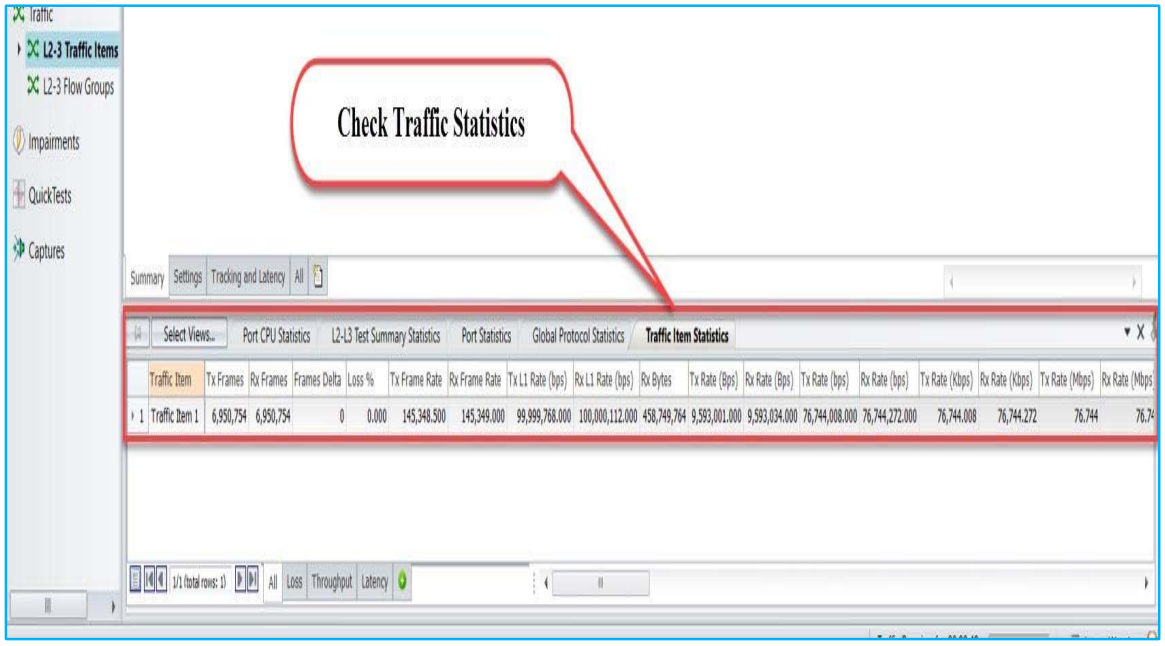


Fig 2.15.1 Check for traffic item statistics

3. Configure BGP through Automation (REST API):

This section provides a walk-through to reproduce the BGP emulation scenario by using Python to send REST API.

3.1 Initialize Environment:

Requirement: Python2.7 - 3.6 and Python modules “requests”
Import Rest API Packages “pip install ixnetwork”

3.2 Add Chassis and Lock Ports:

```
import json
import requests

def waitForComplete(response, sessionUrl, timeout=90):
    if response.json().has_key("errors"):
        print(response.json()["errors"][0])
        return 1
    print("\n", sessionUrl)
    print("\t\tState:", response.json()["state"])
    while response.json()["state"] == "IN_PROGRESS" or response.json()["state"] == "down":
        if timeout == 0:
            return 1
    time.sleep(1)
    response = requests.get(sessionUrl)
    state = response.json()["state"]
    print("\t\tState:", state)
    timeout = timeout - 1
    return 0

def getVportMapping(portNumber):
    for vport in vportList:
        response = requests.get(vport , verify=False)
        connectedTo = response.json()["connectedTo"]
        chassisId = connectedTo.split("/")[8]
        card = connectedTo.split("/")[10]
        portNum = connectedTo.split("/")[12]
        port = chassisId+"/"+card+"/"+portNum
        if port == portNumber:
            print "\nReturing vport:", vport
            return vport
```

The connection session ID:

root = "http://10.154.162.75:11009/api/v1/sessions/1/ixnetwork"

REST API Command syntax:

'xpath' *: REST API path on IxN - ["http://10.154.162.75:11009/api/v1/sessions/1/ixnetwork"]*
'data' *: data input for REST command should be passed in json format as - "json.dumps({})"*
'headers' *: header to be used for all REST commands - {'content-type': 'application/json'}*
'verify' *: verify set to 'False' to disable the SSL Certificate*

REST commands used in general - ***GET, POST, PATCH , DELETE , OPTIONS***
GET *: GET command is used to retrieve the configurations on IxNetwork*
POST *: POST command is used to create/configure new configurations on IxNetwork*
PATCH *: PATCH command is for modifying existing configurations on IxNetwork*
DELETE *: DELETE command is for Deleting the configurations on IxNetwork*
OPTIONS *: OPTIONS command is for viewing available API command options under given 'xpath'*

```
response = requests.REST( '<xpath>' , data=json.dumps({}) , headers = {'content-type': 'application/json'} )
```

Create blank configuration:

```
# To create new configuration or remove existing configuration use the POST command with xpath root+'/operations/newconfig' followed by the headers

response = requests.post(root+'/operations/newconfig',
                         headers={'content-type': 'application/json'} , verify=False)
```

Create virtual port 1:

```
# To create new vport use the POST command with xpath root+'/vport' followed by the headers

response = requests.post(root+'/vport', headers={'content-type': 'application/json'} , verify=False)
```

Create virtual port 2:

```
# To create new vport use the POST command with xpath root+'/vport' followed by the headers

response = requests.post(root+'/vport' , headers={'content-type': 'application/json'} , verify=False)
```

Get a list of virtual ports:

```
# To retrieve all available/configured vports use the GET with xpath root+'/vport' followed by the headers

response = requests.get(root+'/vport', headers={'content-type': 'application/json'}, verify=False)
```

```
# vportList is a list of virtual ports configured.

vportList = ["%s/vport/%s" % (root, str(i["id"])) for i in response.json()]
```

Assign ports:

```
# To assign physical ports to vports use the POST command with xpath root+'/operations/assignports'
followed by the json.dumps() data input and headers

response = requests.post(root+'/operations/assignports',
                         data=json.dumps({'arg1': [{'arg1': '10.39.64.132', 'arg2': '1', 'arg3': '1'},
                                                   {'arg1': '10.39.64.132', 'arg2': '1', 'arg3': '2'}],
                                          'arg2': [],'arg3': vportList,'arg4': 'true'}),
                         headers={'content-type': 'application/json'}, verify=False)
```

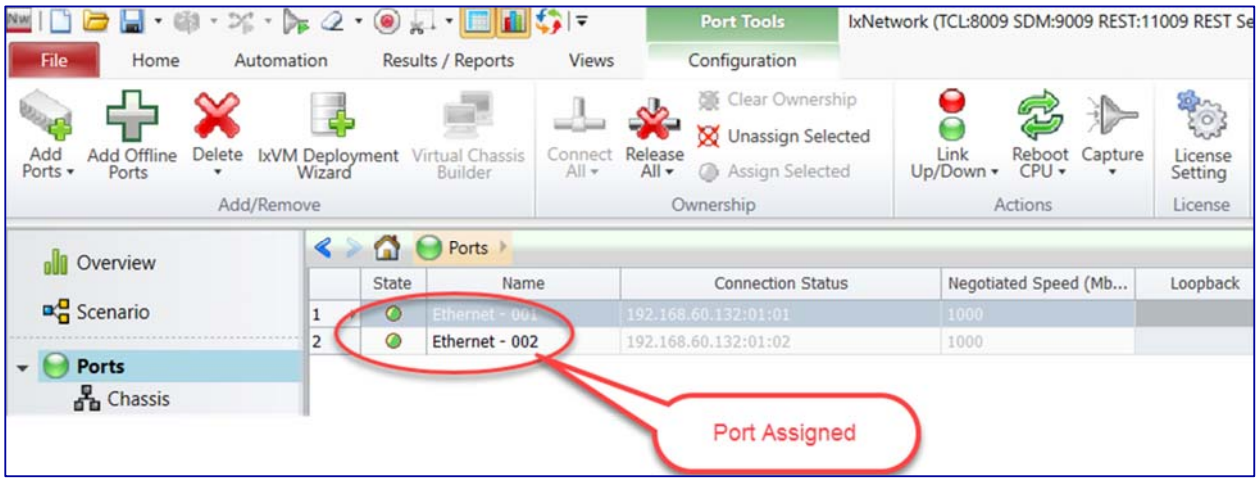


Fig 3.2: Chassis connected and selected ports are assigned

3.3 Create Topology:

Create BGP-1 Topology:

```
# To create new topology use the POST command with xpath root+'/topology' followed by the headers

response = requests.post(root+'/topology',
                         headers={'content-type': 'application/json'}, verify=False)

topology1Vports = []

topology1Vports.append(getVportMapping("1/1/1"))
```

```
# To modify name of topology 1 use the PATCH command with xpath root+'/topology/ followed by the
json.dumps() data input and headers

response = requests.patch(root+'/topology/1',
                          data=json.dumps({'name': 'BGP_1 Topology'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To map physical ports to topology 1 use the PATCH command with xpath root+'/topology/1' followed by
the json.dumps() data input and headers

response = requests.patch(root+'/topology/1',
                          data=json.dumps({'vports': topology1Vports}),
                          headers={'content-type': 'application/json'}, verify=False)
```

Create BGP-2 Topology:

```
# To create new topology, use the POST command with xpath root+'/topology' followed by the headers

response = requests.post(root+'/topology',
                         headers={'content-type': 'application/json'}, verify=False)

topology2Vports = []

topology2Vports.append(getVportMapping("1/1/2"))

# To modify name of topology 2 use the PATCH command with xpath root+'/topology/2 followed by the
json.dumps() data input and headers

response = requests.patch(root+'/topology/2',
                          data=json.dumps({'name': 'BGP_2 Topology'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To map physical ports to topology 2 use the PATCH command with xpath root+'/topology/2' followed by
the json.dumps() data input and headers

response = requests.patch(root+'/topology/2',
                          data=json.dumps({'vports': topology2Vports}),
                          headers={'content-type': 'application/json'}, verify=False)
```

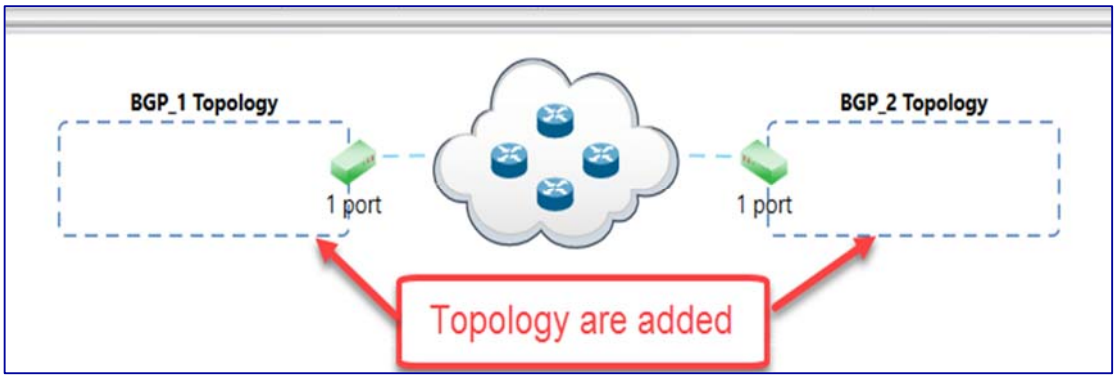


Fig 3.3: Topology are added

3.4 Create Device Group:

Create BGP-1 Device Group:

```
# To create new device group use the POST command with xpath root+'/topology/1/deviceGroup'
followed by the headers

response = requests.post(root+'/topology/1/deviceGroup',
                         headers={'content-type': 'application/json'}, verify=False)

# To modify multiplier of device group use the PATCH command with xpath
root+'/topology/1/deviceGroup/1' followed by the json.dumps() data input and headers

response = requests.patch(root+'/topology/1/deviceGroup/1',
                          data=json.dumps({'multiplier': 1}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify name of device group use PATCH command with xpath
root+'/topology/1/deviceGroup/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/topology/1/deviceGroup/1',
                          data=json.dumps({'name': 'BGP_1 Device Group'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

Create BGP-2 Device Group:

```
# To create new device group use the POST command with xpath root+'/topology/2/deviceGroup'
followed by the headers

response = requests.post(root+'/topology/2/deviceGroup',
                         headers={'content-type': 'application/json'}, verify=False)
```

```
# To modify multiplier of device group use the PATCH command with xpath
root+'/topology/2/deviceGroup/1' followed by the json.dumps() data input and headers

response = requests.patch(root+'/topology/2/deviceGroup/1',
                          data=json.dumps({'multiplier': 1}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify name of device group use PATCH command with xpath
root+'/topology/2/deviceGroup/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/topology/2/deviceGroup/1',
                          data=json.dumps({'name': 'BGP_2 Device Group'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

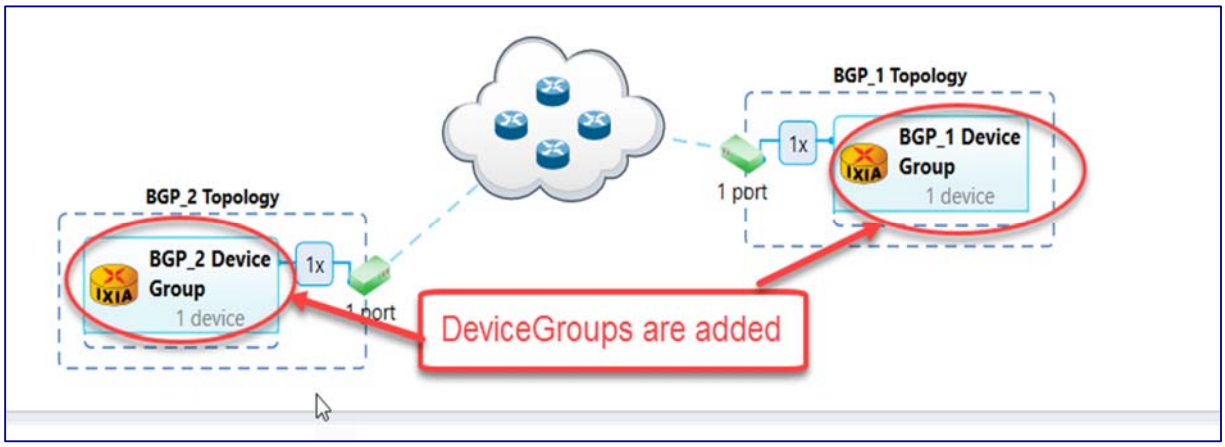


Fig 3.4: Device groups added to respective topologies

3.5 Create Ethernet Stack:

Create Ethernet Stack-1 on topology-1 device group 1:

```
# To create new ethernet stack use the POST command with xpath
root+'/topology/1/deviceGroup/1/ethernet' followed by the headers.

response = requests.post(root+'/topology/1/deviceGroup/1/ethernet',
                         headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of ethernet stack 1 use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1' followed by the headers.

response = requests.get(root+'/topology/1/deviceGroup/1/ethernet/1',
                        headers={'content-type': 'application/json'}, verify=False)
```

```
# To fetch ethernet 'MAC' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['mac']

# To modify ethernet 'MAC' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by the json.dumps() data input and
headers.

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '00:01:01:00:00:01','direction': 'increment',
                          'step': '00:00:00:00:00:01'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify name of ethernet stack use the PATCH command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1' followed by the json.dumps() data input and headers

response = requests.patch(root+'/topology/1/deviceGroup/1/ethernet/1',
                          data=json.dumps({'name': 'ethernet1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of ethernet stack 1 use the GET command with xpath
(root+'/topology/1/deviceGroup/1/ethernet/1' followed by the headers.

response = requests.get(root+'/topology/1/deviceGroup/1/ethernet/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch ethernet 'MTU' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['mtu']

# To modify ethernet 'MTU' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 1500}),
                          headers={'content-type': 'application/json'}, verify=False)
```

Create Ethernet Stack-1 on topology-2 device group 1:

```
# To create new ethernet stack use the POST command with xpath
root+'/topology/2/deviceGroup/1/ethernet' followed by the headers

response=requests.post(root+'/topology/2/deviceGroup/1/ethernet',
                       headers={'content-type': 'application/json'}, verify=False)
```

```
# To retrieve contents of ethernet stack 1 use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1' followed by the headers.

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch ethernet 'MAC' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['mac']

# To modify ethernet 'MAC' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by the json.dumps() data input and
headers.

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '00:01:03:00:00:01', 'direction': 'increment',
                          'step': '00:00:00:00:00:01'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify name of ethernet stack use the PATCH command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1' followed by the json.dumps() data input and headers

response = requests.patch(root+'/topology/2/deviceGroup/1/ethernet/1',
                          data=json.dumps({'name': 'ethernet2'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of ethernet stack 1 use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1' followed by the headers.

response = requests.get(root+'/topology/2/deviceGroup/1/ethernet/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch ethernet 'MTU' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['mtu']

# To modify ethernet 'MTU' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 1500}),
                          headers={'content-type': 'application/json'}, verify=False)
```

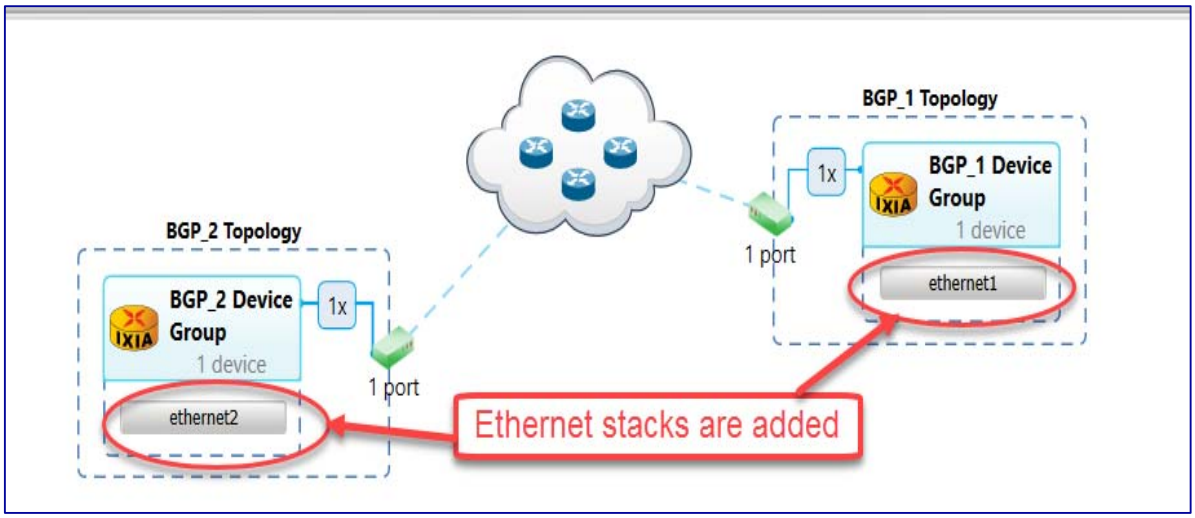


Fig 3.5: Ethernet stacks added to device groups

3.6 Create Ipv4 Stack:

Creating Ipv4 Stack on topology-1 device group 1:

```
# To create new IPv4 stack use the POST command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4' followed by the headers.

response = requests.post(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4',
                         headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1' followed by the headers.

response = requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1',
                         headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 ''resolveGateway' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['resolveGateway']

# To modify IPv4 'resolveGateway' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': True}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1' followed by the headers.

response = requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 'prefix' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['prefix']

# To modify IPv4 'prefix' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 24}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1' followed by the headers.

response = requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 'gatewayIp' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['gatewayIp']

# To modify the IPv4 'gatewayIp' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '100.1.1.100',
                         'direction': 'increment', 'step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1' followed by the headers.

response = requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 'address' from the GET command response and storing to variable 'multiValue'
multiValue = response.json()['address']
```

```
# To modify the IPv4 'address' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '100.1.1.1',
                          'direction': 'increment', 'step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

Creating Ipv4 Stack on topology-2 device group 1:

```
# To create new IPv4 stack use the POST command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4' followed by the headers.

response = requests.post(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4',
                         headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1' followed by the headers

response = requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 'resolveGateway' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['resolveGateway']

# To modify IPv4 'resolveGateway' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': True}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1' followed by the headers

response = requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 'prefix' from the GET command response and storing to variable 'multiValue'

multiValue = response.json()['prefix']
```

```
# To modify IPv4 ‘prefix’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 24}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1’ followed by the headers

response = requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 ‘gatewayIp' from the GET command response and storing to variable ‘multiValue’

multiValue = response.json()['gatewayIp']

# To modify the IPv4 'gatewayIp' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '100.1.1.1',
                          'direction': 'increment', 'step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve contents of IPv4 stack use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1’ followed by the headers

response = requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1',
                        headers={'content-type': 'application/json'}, verify=False)

# To fetch IPv4 ‘address’ from the GET command response and storing to variable ‘multiValue’

multiValue = response.json()['address']

# To modify the IPv4 'address' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '100.1.1.100',
                          'direction': 'increment', 'step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

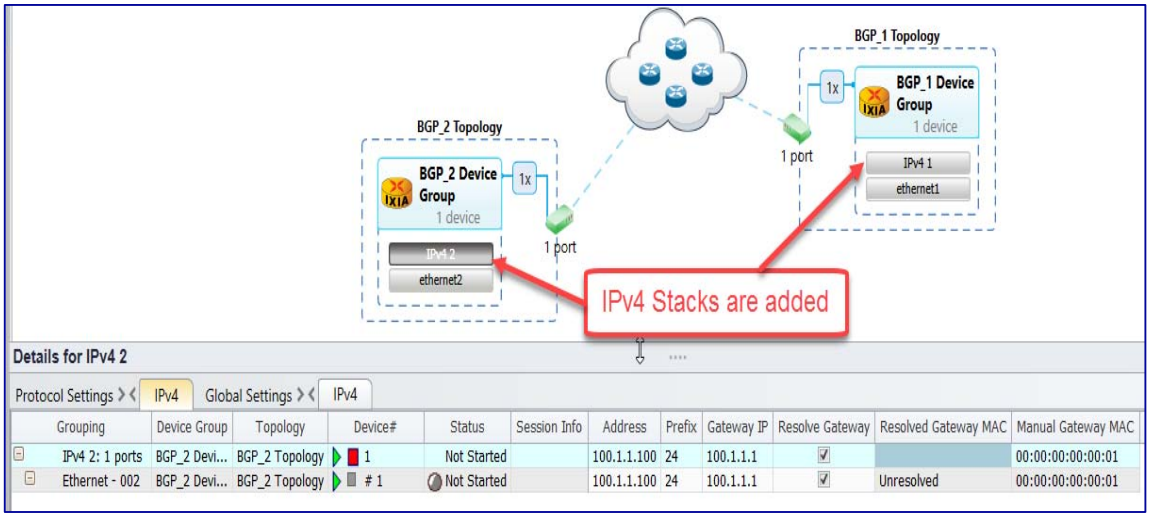


Fig 3.6: IPv4 stacks added to ethernet stacks

3.7 Create BGP:

Create BGP on topology-1 device group 1:

```
# To create new BgpIPv4 protocol use the POST command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer' followed by the headers

response=requests.post(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer',
                       headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "enableBgpId" from GET command response and storing to ‘multiValue’

multiValue = response.json()['enableBgpId']

# To modify the BgpIpv4 "enableBgpId" use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': True}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To modify 'bgpIpv4Peer' name use the PATCH command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1' followed by json.dumps() data input
and headers

response=requests.patch(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                        data=json.dumps({'name': 'Topo1Bgp'}),
                        headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "holdTimer" from GET command response and storing to 'multiValue'

multiValue = response.json()['holdTimer']

# To modify BgpIPv4 'holdTimer' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 90}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)


# To fetch BgpIpv4 "dutIp" from GET command response and storing to 'multiValue'

multiValue = response.json()['dutIp']

# To modify BgpIPv4 'dutIp' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '100.1.1.100',
                          'direction': 'increment', 'step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "enableGracefulRestart" from GET command response and storing to ‘multiValue’

multiValue = response.json()['enableGracefulRestart']

# To modify BgpIPv4 ‘enableGracefulRestart’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': False}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "restartTime" from GET command response and storing to ‘multiValue’

multiValue = response.json()['restartTime']

# To modify BgpIPv4 ‘restartTime’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 45}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "type" from GET command response and storing to ‘multiValue’
multiValue = response.json()['type']
```

```
# To modify BgpIPv4 ‘type’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 'internal'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "enableBgpIdSameasRouterId" from GET command response and storing to
‘multiValue’

multiValue = response.json()['enableBgpIdSameasRouterId']

# To modify BgpIPv4 ‘enableBgpIdSameasRouterId’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': True}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "staleTime" from GET command response and storing to ‘multiValue’

multiValue = response.json()['staleTime']

# To modify BgpIPv4 ‘staleTime’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 0}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "flap" from GET command response and storing to ‘multiValue’

multiValue = response.json()['flap']

# To modify BgpIPv4 ‘flap’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': ['false', 'false', 'false', 'false']}),
                          headers={'content-type': 'application/json'}, verify=False)
```

Create BGP on topology-2 device group1:

```
# To create new BgpIPv4 protocol use the POST command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer' followed by the headers

response=requests.post(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer',
                       headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "enableBgpId" from GET command response and storing to ‘multiValue’

multiValue = response.json()['enableBgpId']

# To modify the BgpIpv4 "enableBgpId" use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': True}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To modify 'bgpIpv4Peer' name use the PATCH command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1' followed by json.dumps() data input
and headers

response = requests.patch(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                          data=json.dumps({'name': 'Topo2Bgp1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "holdTimer" from GET command response and storing to 'multiValue'

multiValue = response.json()['holdTimer']

# To modify BgpIPv4 'holdTimer' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 90}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "dutIp" from GET command response and storing to 'multiValue'

multiValue = response.json()['dutIp']

# To modify BgpIPv4 'dutIp' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '100.1.1.1',
                          'direction': 'increment','step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers
```

```
response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "enableGracefulRestart" from GET command response and storing to ‘multiValue’

multiValue = response.json()['enableGracefulRestart']

# To modify BgpIPv4 ‘enableGracefulRestart’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': False}),
                          headers={'content-type': 'application/json'}, verify=False)


# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "restartTime" from GET command response and storing to ‘multiValue’

multiValue = response.json()['restartTime']

# To modify BgpIPv4 ‘restartTime’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 45}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "type” from GET command response and storing to ‘multiValue’
multiValue = response.json()['type']
```

```
# To modify BgpIPv4 ‘type’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers


response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 'internal'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "enableBgpIdSameasRouterId” from GET command response and storing to
‘multiValue’
multiValue = response.json()['enableBgpIdSameasRouterId']

# To modify BgpIPv4 ‘enableBgpIdSameasRouterId’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': True}),
                          headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch BgpIpv4 "staleTime” from GET command response and storing to ‘multiValue’

multiValue = response.json()['staleTime']

# To modify BgpIPv4 ‘staleTime’ use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': 0}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To retrieve details of BgpIPv4 protocol configured use the GET command with xpath
root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/ followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/ethernet/1/ipv4/1/bgpIpv4Peer/1',
                      headers={'content-type': 'application/json'}, verify=False)


# To fetch BgpIpv4 "flap" from GET command response and storing to 'multiValue'

multiValue = response.json()['flap']

# To modify BgpIPv4 'flap' use the PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/singleValue" followed by json.dumps() data input and
headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/singleValue",
                          data=json.dumps({'value': False}),
                          headers={'content-type': 'application/json'}, verify=False)
```

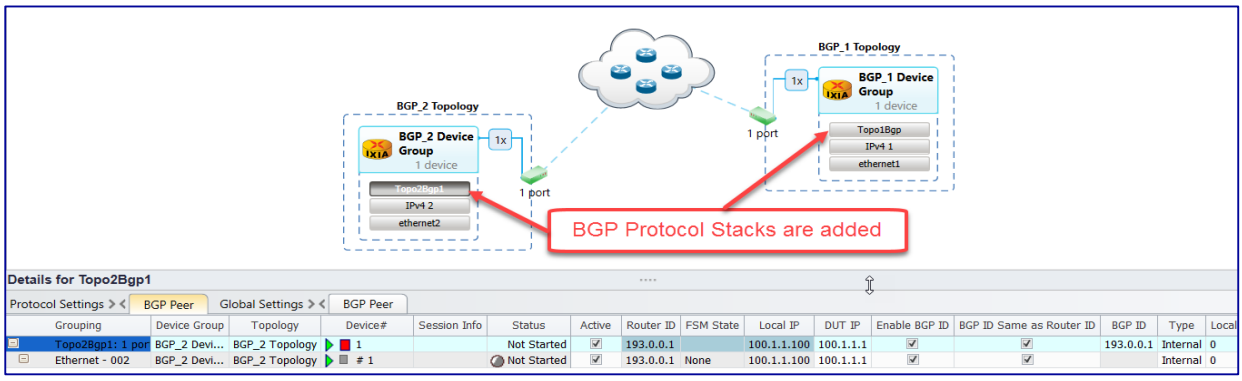


Fig 3.7: BGP stacks added to ipv4 stacks

3.8 Create Network Group:

Create Network Group on topology-1 device group1:

```
# To create new network group, use POST Command with xpath
root+'/topology/1/deviceGroup/1/networkGroup' followed by the headers

response=requests.post(root+'/topology/1/deviceGroup/1/networkGroup',
                       headers={'content-type': 'application/json'}, verify=False)
```

```
# To modify the name of network group use PATCH command with xpath
root+'/topology/1/deviceGroup/1/networkGroup/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/topology/1/deviceGroup/1/networkGroup/1',
                          data=json.dumps({'name': 'BGP_1_Network_Group1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the multiplier of network group use PATCH command with xpath
root+'/topology/1/deviceGroup/1/networkGroup/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/topology/1/deviceGroup/1/networkGroup/1',
                          data=json.dumps({'multiplier': 1}),
                          headers={'content-type': 'application/json'}, verify=False)

# To create IPv4 Prefix Pool for network group use POST command with xpath
root+'/topology/1/deviceGroup/1/networkGroup/1/ipv4PrefixPools' followed by json.dumps() data input
and headers

response=requests.post(root+'/topology/1/deviceGroup/1/networkGroup/1/ipv4PrefixPools',
                       headers={'content-type': 'application/json'}, verify=False)

# To retrieve details of configured IPv4 Prefix Pool on network group use GET command with xpath
root+'/topology/1/deviceGroup/1/networkGroup/1/ipv4PrefixPools/1' followed by the headers

response=requests.get(root+'/topology/1/deviceGroup/1/networkGroup/1/ipv4PrefixPools/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch ipv4PrefixPools "networkAddress" from GET command response and storing 'multiValue'

multiValue = response.json()['networkAddress']

# To modify the ipv4PrefixPools "networkAddress" value use PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '160.1.0.0','direction': 'increment','step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

Create Network Group on topology-2 device group 1:

```
# To create new network group use POST Command with xpath
root+'/topology/2/deviceGroup/1/networkGroup' followed by the headers

response=requests.post(root+'/topology/2/deviceGroup/1/networkGroup',
                       headers={'content-type': 'application/json'}, verify=False)
```

```
# To modify the name of network group use PATCH command with xpath
root+'/topology/2/deviceGroup/1/networkGroup/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/topology/2/deviceGroup/1/networkGroup/1',
                          data=json.dumps({'name': 'BGP_2_Network_Group1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the multiplier of network group use PATCH command with xpath
root+'/topology/2/deviceGroup/1/networkGroup/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/topology/2/deviceGroup/1/networkGroup/1',
                          data=json.dumps({'multiplier': 1}),
                          headers={'content-type': 'application/json'}, verify=False)


# To create IPv4 Prefix Pool for network group use POST command with xpath
root+'/topology/2/deviceGroup/1/networkGroup/1/ipv4PrefixPools' followed by json.dumps() data input
and headers

response=requests.post(root+'/topology/2/deviceGroup/1/networkGroup/1/ipv4PrefixPools',
                       headers={'content-type': 'application/json'}, verify=False)


# To retrieve details of IPv4 Prefix Pool on network group use GET command with xpath
root+'/topology/1/deviceGroup/1/networkGroup/1/ipv4PrefixPools/1' followed by the headers

response=requests.get(root+'/topology/2/deviceGroup/1/networkGroup/1/ipv4PrefixPools/1',
                      headers={'content-type': 'application/json'}, verify=False)

# To fetch ipv4PrefixPools "networkAddress" from GET command response and storing 'multiValue'

multiValue = response.json()['networkAddress']


# To modify the ipv4PrefixPools "networkAddress" value use PATCH command with xpath
'http://10.154.162.75:11009'+multiValue+"/counter" followed by json.dumps() data input and headers

response = requests.patch('http://10.154.162.75:11009'+multiValue+"/counter",
                          data=json.dumps({'start': '150.1.0.0','direction': 'increment', 'step': '0.0.0.1'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

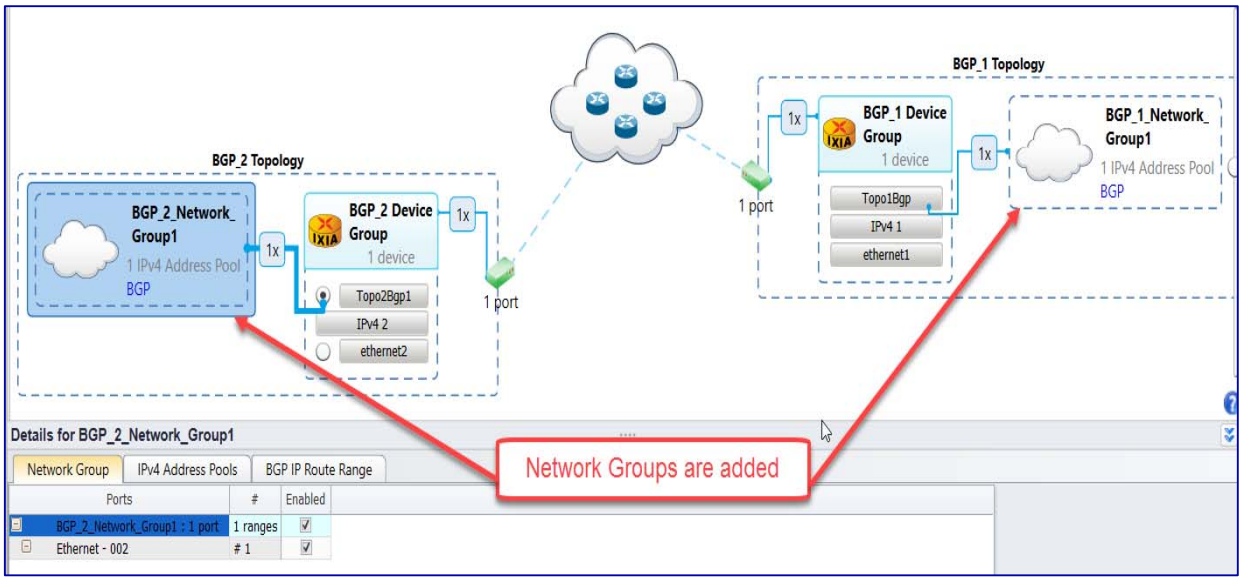


Fig 3.8 Adding BGP network group to device group

3.9 Start Protocols:

 # To start all protocols configured use POST command with xpath root+'/operations/startallprotocols' followed by headers

```
response = requests.post(root+'/operations/startallprotocols',
                         headers={'content-type': 'application/json'}, verify=False)
```

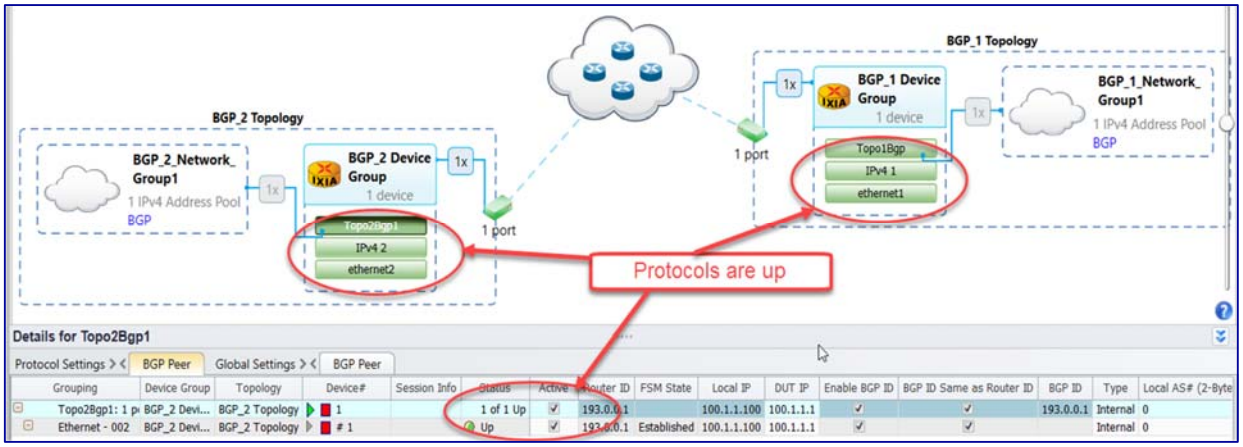


Fig 3.9 Protocol stacks are up

3.10 Configure Traffic:

Create Traffic Item:

```
# To create new traffic items use POST command with xpath
root+'/traffic/trafficItem' followed by headers

response = requests.post(root+'/traffic/trafficItem',
                         headers={'content-type': 'application/json'}, verify=False)

# To modify the name of the traffic item 1  use PATCH command with xpath root+'/traffic/trafficItem/1'
followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1',
                          data=json.dumps({'name': 'BGP topo1 to topo2'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify traffic item 1 'srcDestMesh' use PATCH command with xpath root+'/traffic/trafficItem/1'
followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1',
                          data=json.dumps({'srcDestMesh': 'one-to-one'}),
                          headers={'content-type': 'application/json'}, verify=False)


# To modify traffic item 1 'allowSelfDestined' use PATCH command with xpath root+'/traffic/trafficItem/1'
followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1',
                          data=json.dumps({'allowSelfDestined': False}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify traffic item 1 'trafficType' use PATCH command with xpath root+'/traffic/trafficItem/1'
followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1',
                          data=json.dumps({'trafficType': 'ipv4'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify traffic item 1 'routeMesh' use PATCH command with xpath root+'/traffic/trafficItem/1'
followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1',
                          data=json.dumps({'routeMesh': 'oneToOne'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To modify traffic item 1 'biDirectional' use PATCH command with xpath root+'/traffic/trafficItem/1'
followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1',
                          data=json.dumps({'biDirectional': True}),
                          headers={'content-type': 'application/json'}, verify=False)
```

Create Endpoints:

```
# To create endpointSet for the traffic item use POST command with xpath
root+'/traffic/trafficItem/1/endpointSet' followed by headers

response = requests.post(root+'/traffic/trafficItem/1/endpointSet',
                         headers={'content-type': 'application/json'}, verify=False)

# To modify the endpoint set 'sources' use PATCH command with xpath
root+'/traffic/trafficItem/1/endpointSet/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1/endpointSet/1',
                          data=json.dumps({'sources':
                          [root+'/topology/1/deviceGroup/1/networkGroup/1/ipv4PrefixPools/1']}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the endpoint set 'name' use PATCH command with xpath
root+'/traffic/trafficItem/1/endpointSet/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1/endpointSet/1',
                          data=json.dumps({'name': 'FlowGroup-1'}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the endpoint set 'destinations' use PATCH command with xpath
root+'/traffic/trafficItem/1/endpointSet/1' followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1/endpointSet/1',
                          data=json.dumps({'destinations':
                          [root+'/topology/2/deviceGroup/2/ethernet/1/ipv4/1']}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the traffic item transmissionControl 'type' use PATCH command with xpath
root+'/traffic/trafficItem/1/configElement/1/transmissionControl' followed by json.dumps() data input
and headers

response = requests.patch(root+'/traffic/trafficItem/1/configElement/1/transmissionControl',
                          data=json.dumps({'type': 'fixedFrameCount'}),
                          headers={'content-type': 'application/json'}, verify=False)
```

```
# To modify the traffic item transmissionControl ‘frameCount’ use PATCH command with xpath
root+'/traffic/trafficItem/1/configElement/1/transmissionControl' followed by json.dumps() data input
and headers

response = requests.patch(root+'/traffic/trafficItem/1/configElement/1/transmissionControl',
                          data=json.dumps({'frameCount': 50000}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the traffic item frameRate ‘rate’ use PATCH command with xpath
root+'/traffic/trafficItem/1/configElement/1/frameRate' followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1/configElement/1/frameRate',
                          data=json.dumps({'rate': 88}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the traffic item frameRate ‘type’ use PATCH command with xpath
root+'/traffic/trafficItem/1/configElement/1/frameRate' followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1/configElement/1/frameRate',
                          data=json.dumps({'type': 'percentLineRate'}),
                          headers={'content-type': 'application/json'}, verify=False)


# To modify the traffic item frameSize ‘fixedSize’ use PATCH command with xpath
root+'/traffic/trafficItem/1/configElement/1/frameSize' followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic/trafficItem/1/configElement/1/frameSize',
                          data=json.dumps({'fixedSize': 512}),
                          headers={'content-type': 'application/json'}, verify=False)

# To modify the traffic item ‘tracking’ use PATCH command with xpath
root+'/traffic//trafficItem/1/tracking' followed by json.dumps() data input and headers

response = requests.patch(root+'/traffic//trafficItem/1/tracking',
                          data=json.dumps({'trackBy': ['flowGroup0', 'sourceDestValuePair0']}),
                          headers={'content-type': 'application/json'}, verify=False)
```

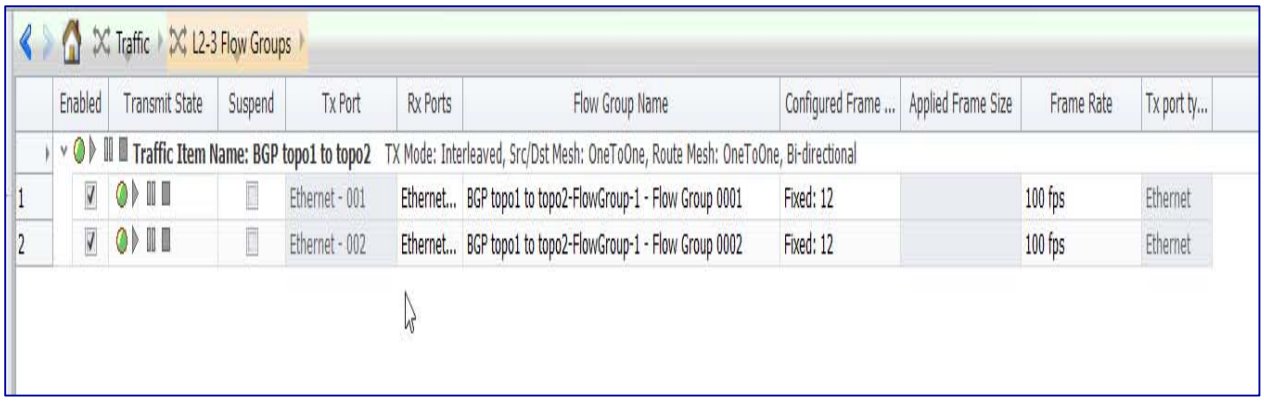


Fig 3.10 L2-3 Traffic configured with the specified options

3.11 Start Traffic and Stop Traffic:

```
# To apply all configured traffic items,  use POST command with xpath  root+'/traffic/operations/apply'
followed by json.dumps() data input and headers

response = requests.post(root+'/traffic/operations/apply',
                         data=json.dumps({'arg1': root+'/traffic'}),
                         headers={'content-type': 'application/json'}, verify=False)

# Wait till all configured traffic items configurations are applied

waitForComplete(response, root+"/traffic/operations/apply"+response.json()["id"])


# To start all configured traffic items, use POST command with xpath root+'/traffic/operations/start'
followed by json.dumps() data input and headers

response = requests.post(root+'/traffic/operations/start',
                         data=json.dumps({'arg1': root+'/traffic'}),
                         headers={'content-type': 'application/json'}, verify=False)
```

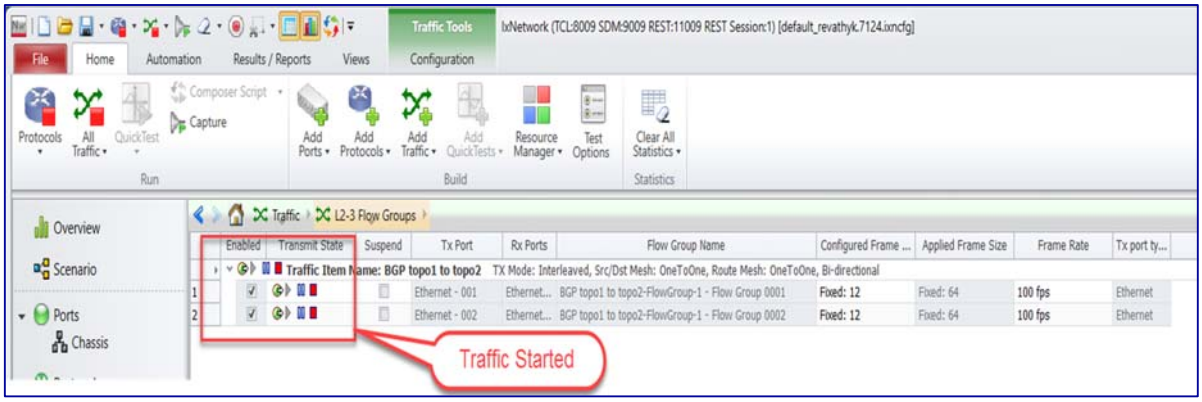


Fig 3.11 Traffic Started

```
# To stop all the running traffic items, use POST command with xpath
 root+'/traffic/operations/stop' followed by json.dumps() data input and headers

response = requests.post(root+'/traffic/operations/stop',
                         data=json.dumps({'arg1': root+'/traffic'}),
                         headers={'content-type': 'application/json'}, verify=False)
```

3.12 Disconnect:

```
# To delete the IxNetwork REST API session use DELETE command with xpath root

requests.delete(root , headers={'content-type': 'application/json'}, verify=False)
```

4. Configure BGP through Automation(Python-OpenIxia):

This section provides a walk-through to reproduce the BGP emulation scenario through OpenIxia (Python), which calls the REST API.

4.1 Initialize Environment:

Import the Required Packages and Check for the sanity of the System

Python2.7 - 3.6
Python modules: requests

```
import sys, traceback
sys.path.insert(0, '../Modules')
from IxNetRestApi import *
from IxNetRestApiPortMgmt import PortMgmt
from IxNetRestApiTraffic import Traffic
from IxNetRestApiProtocol import Protocol
from IxNetRestApiStatistics import Statistics
```

4.2 Add Chassis and Lock Ports:

```
forceTakePortOwnership = True
releasePortsWhenDone = False
enableDebugTracing = True
deleteSessionAfterTest = False
ixChassisIp = '10.39.64.132'
portList = [[ixChassisIp, '1', '11'],  [ixChassisIp, '1', '12']]
```

Connect - Connects to the IxNetwork REST API server

```
mainObj = Connect (apiServerIp='192.168.70.3',
                   serverIpPort='11009',
                   serverOs=osPlatform,
                   deleteSessionAfterTest=deleteSessionAfterTest,
                   httpInsecure=True
                   )
```

connectIxChassis - Connects to the Ixia chassis

```
portObj = PortMgmt(mainObj)

portObj.connectIxChassis(ixChassisIp)
```

```
if portObj.arePortsAvailable(portList, raiseException=False) != 0:
    if forceTakePortOwnership == True:
        portObj.releasePorts(portList)
        portObj.clearPortOwnership(portList)
    else:
        raise IxNetRestApiException('Ports are owned by another user and forceTakePortOwnership is set to False')
```

newBlankConfig - Create a new Ix-Network session

```
mainObj.newBlankConfig()
```

assignPorts - Assign the ports

```
portObj.assignPorts(portList)
```

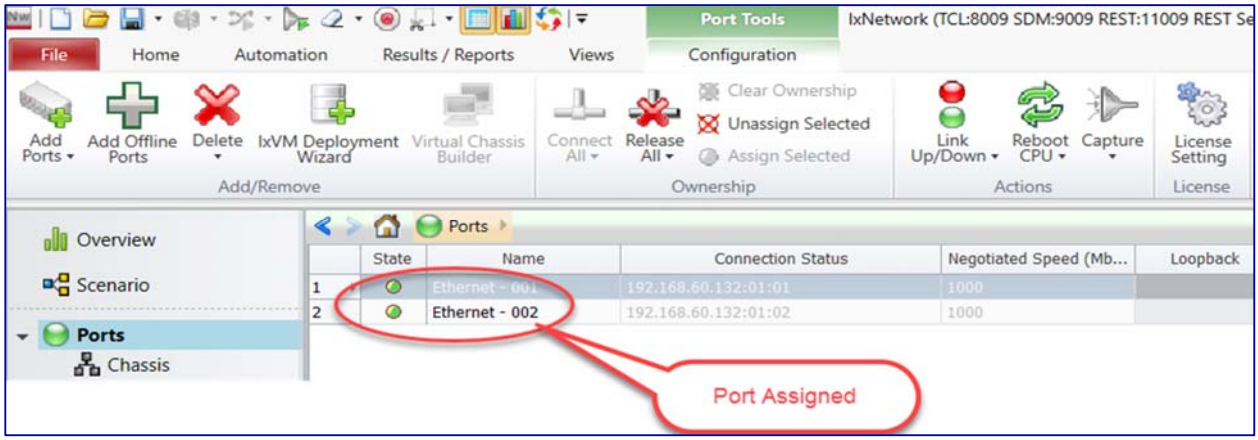


Fig 4.2: Chassis connected and selected ports are assigned

4.3 Create Topology:

createTopologyNgpf - Adds Topology to the specified protocol object.

```
protocolObj = Protocol(mainObj, portObj)

topologyObj1 = protocolObj.createTopologyNgpf(portList=[portList[0]],
                                              topologyName='BGP_1 Topology')

topologyObj2 = protocolObj.createTopologyNgpf(portList=[portList[1]],
                                              topologyName='BGP_2 Topology')
```

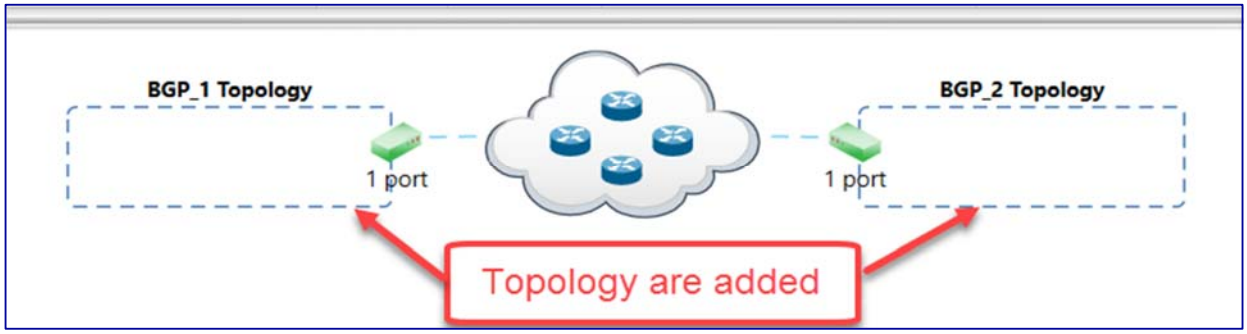


Fig 4.3: Topology are added

4.4 Create DeviceGroup:

createDeviceGroupNgpf - Adds device groups to the specified protocol object.

```
deviceGroupObj1 = protocolObj.createDeviceGroupNgpf(topologyObj1,
                                                    multiplier=1,
                                                    deviceGroupName='BGP_1_DeviceGroup')

deviceGroupObj2 = protocolObj.createDeviceGroupNgpf(topologyObj2,
                                                    multiplier=1,
                                                    deviceGroupName='BGP_2_DeviceGroup')
```

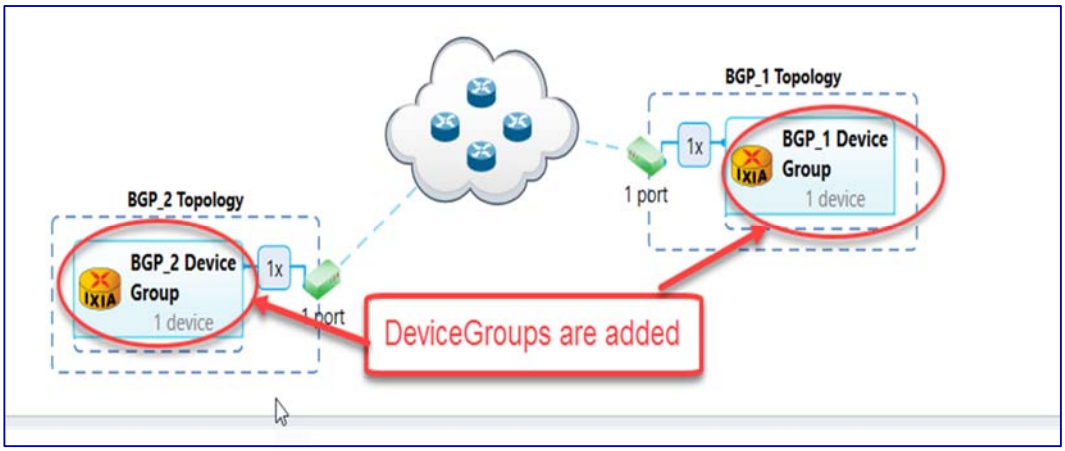


Fig 4.4: Device groups added to respective topologies

4.5 Create Ethernet Stack:

createEthernetNgpf - Configures the ethernet stack with the Specified Options.

```
ethernetObj1 = protocolObj.createEthernetNgpf(deviceGroupObj1,
                                               ethernetName='Ethernet1',
                                               macAddress={'start': '00:01:01:00:00:01',
                                               'direction': 'increment',
                                               'step': '00:00:00:00:00:01'},
                                               macAddressPortStep='disabled')

ethernetObj2 = protocolObj.createEthernetNgpf(deviceGroupObj2,
                                               ethernetName='Ethernet2',
                                               macAddress={'start': '00:01:02:00:00:01',
                                               'direction': 'increment',
                                               'step': '00:00:00:00:00:01'},
                                               macAddressPortStep='disabled')
```

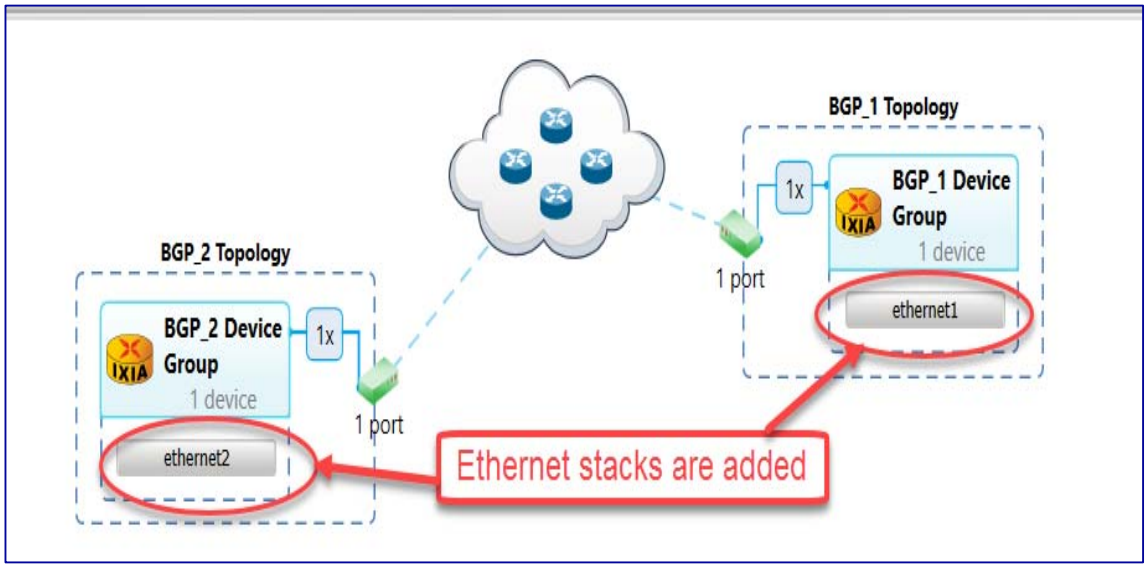


Fig 4.5: Ethernet stacks added to device groups

4.6 Create Ipv4 Stack:

createIpv4Ngpf - Configures the Ipv4 stack with the Specified Options

```
ipv4Obj1 = protocolObj.createIpv4Ngpf(ethernetObj1,
                                      ipv4Address={'start': '1.1.1.1',
                                                   'direction': 'increment',
                                                   'step': '0.0.0.1'},
                                      ipv4AddressPortStep='disabled',
                                      gateway={'start': '1.1.1.2',
                                               'direction': 'increment',
                                               'step': '0.0.0.0'},
                                      gatewayPortStep='disabled',
                                      prefix=24,
                                      resolveGateway=True)

ipv4Obj2 = protocolObj.createIpv4Ngpf(ethernetObj2,
                                      ipv4Address={'start': '1.1.1.2',
                                                   'direction': 'increment',
                                                   'step': '0.0.0.1'},
                                      ipv4AddressPortStep='disabled',
                                      gateway={'start': '1.1.1.1',
                                               'direction': 'increment',
                                               'step': '0.0.0.0'},
                                      gatewayPortStep='disabled',
                                      prefix=24,
                                      resolveGateway=True)
```

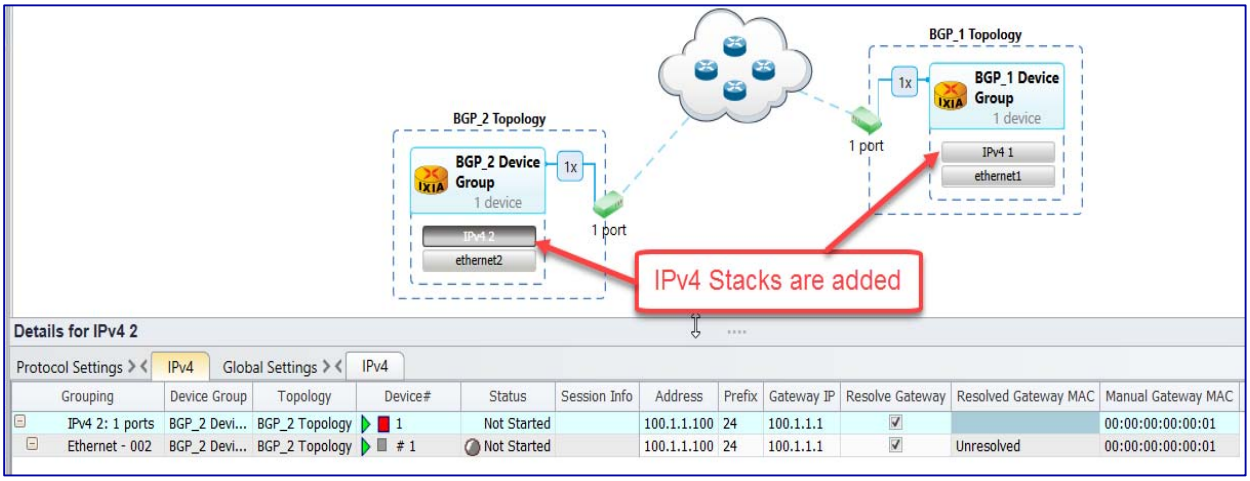


Fig 4.6: IPv4 stacks added to ethernet stacks

4.7 Create BGP:

configBgp - Configures BGP stack with the specified options

```
bgpObj1 = protocolObj.configBgp(ipv4Obj1,
                                name = 'bgp_1',
                                enableBgp = True,
                                holdTimer = 90,
                                dutIp={'start': '1.1.1.2', 'direction': 'increment', 'step': '0.0.0.0'},
                                localAs2Bytes = 101,
                                enableGracefulRestart = False,
                                restartTime = 45,
                                type = 'internal',
                                enableBgpIdSameasRouterId = True)

bgpObj2 = protocolObj.configBgp(ipv4Obj2,
                                name = 'bgp_2',
                                enableBgp = True,
                                holdTimer = 90,
                                dutIp={'start': '1.1.1.1', 'direction': 'increment', 'step': '0.0.0.0'},
                                localAs2Bytes = 101,
                                enableGracefulRestart = False,
                                restartTime = 45,
                                type = 'internal',
                                enableBgpIdSameasRouterId = True)
```

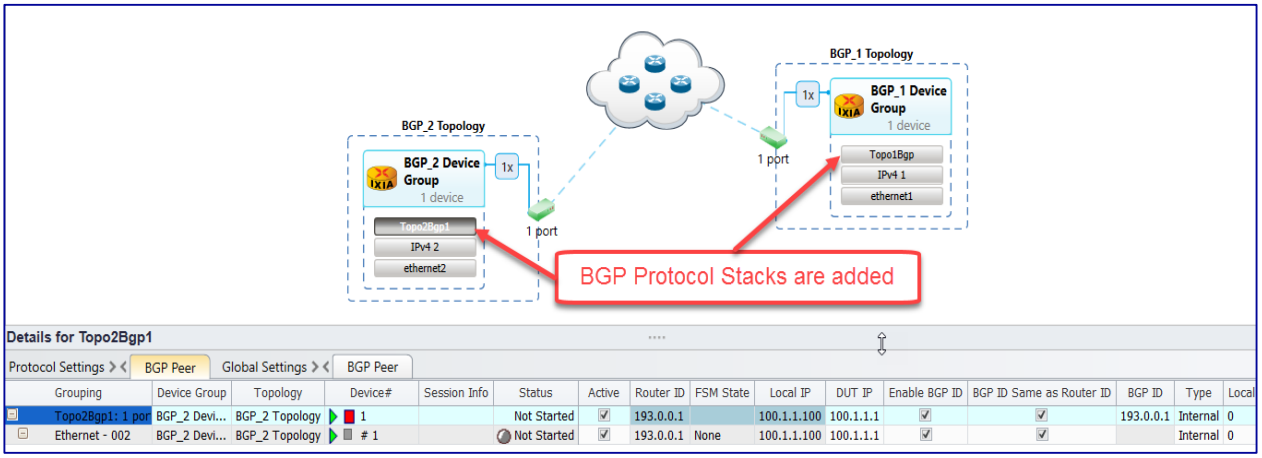


Fig 4.7: BGP stacks added to ipv4 stacks

4.8 Create Network Group:

configNetworkGroup -Configures multivalue with specified.

```
networkGroupObj1 = protocolObj.configNetworkGroup(create=deviceGroupObj1,
                                                  name='BGP_1_NetworkGroup1',
                                                  multiplier = 1,
                                                  networkAddress = {'start': '200.1.0.0',
                                                  'step': '0.0.0.1',
                                                  'direction': 'increment'},
                                                  prefixLength = 24)

networkGroupObj2 = protocolObj.configNetworkGroup(create=deviceGroupObj2,
                                                  name=''BGP_2_NetworkGroup1',
                                                  multiplier = 100,
                                                  networkAddress = {'start': '200.1.0.0',
                                                  'step': '0.0.0.1',
                                                  'direction': 'increment'},
                                                  prefixLength = 24)
```

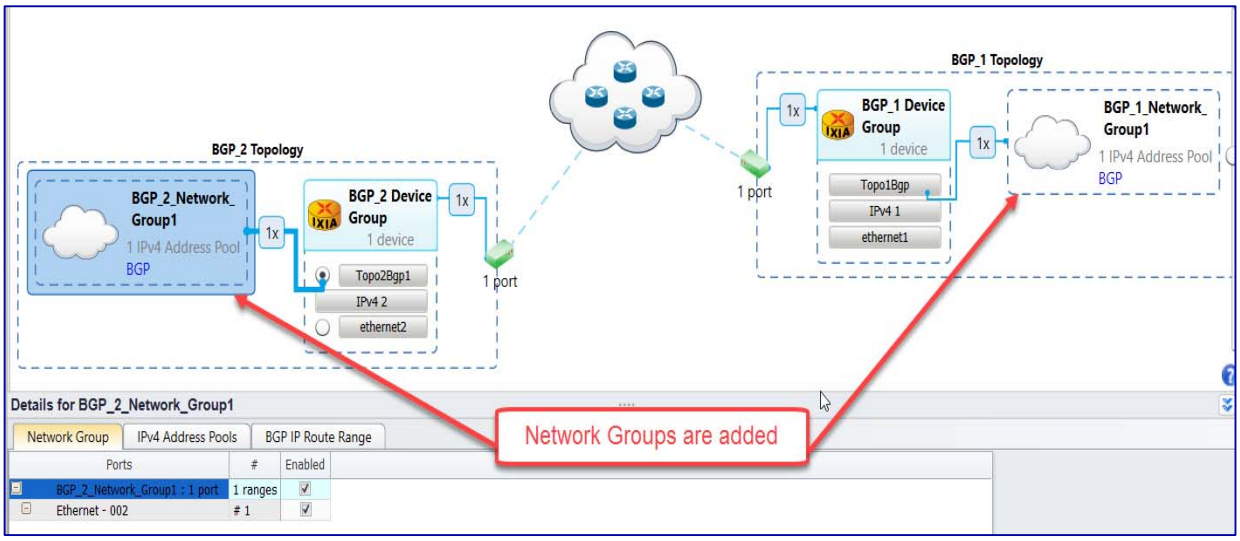


Fig 4.8 Adding BGP network group to device group

4.9 Start Protocols:

startAllProtocols - Start all the protocols configured in the test session

protocolObj.startAllProtocols()

verifyAllProtocolSessionsNgpf – To Verfiy Protocols are up

protocolObj.verifyAllProtocolSessionsNgpf()

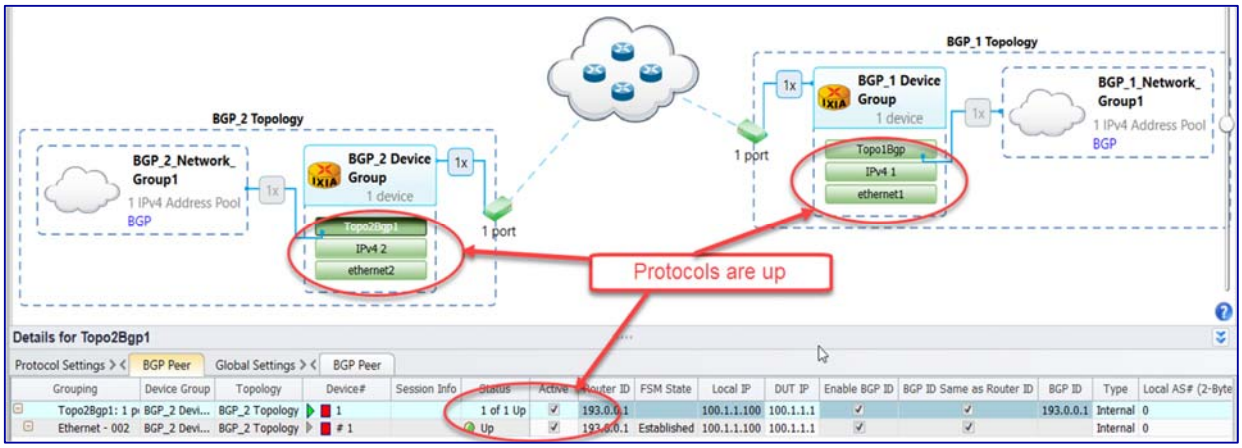


Fig 4.9 Protocol stacks are up

4.10 Configure Traffic:

configTrafficItem - Configures the traffic streams on the specified ports with specified options

trafficObj = Traffic(mainObj)

```
trafficStatus = trafficObj.configTrafficItem( mode='create',
                    trafficItem = {'name':'Topo1 to Topo2', 'trafficType':'ipv4',
                                   'biDirectional':True, 'srcDestMesh':'one-to-one',
                                   'routeMesh':'oneToOne', 'allowSelfDestined':False,
                                   'trackBy': ['flowGroup0','vlanVlanId0']},
                    endpoints = [{'name':'Flow-Group-1','sources': [topologyObj1],
                                  'destinations': [topologyObj2]}],
                    configElements = [{'transmissionType':'fixedFrameCount',
                                       'frameCount': 50000,'frameRate': 88,
                                       'duration': 10, 'frameRateType': 'percentLineRate',
                                      'frameSize': 128 }])
```

trafficItemObj = trafficStatus[0]
endpointObj = trafficStatus[1][0]
configElementObj = trafficStatus[2][0]

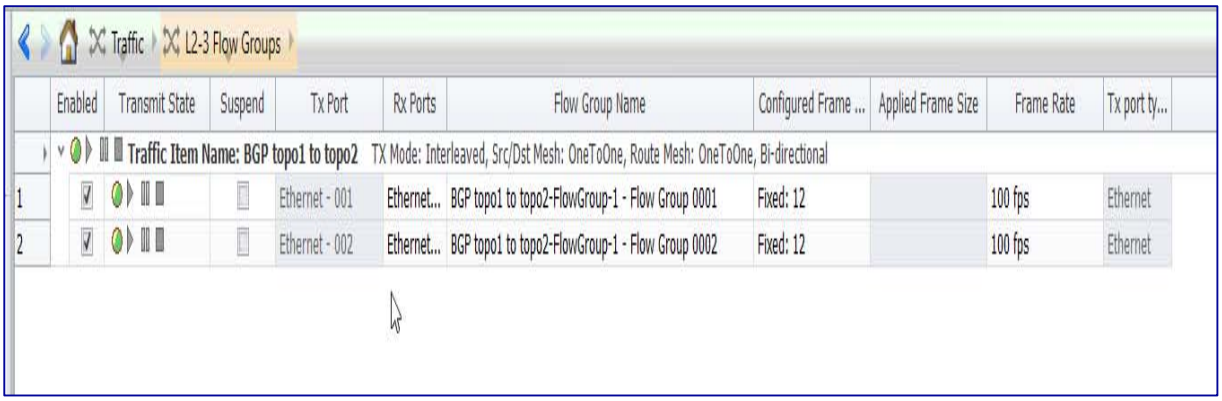


Fig 3.10 L2-3 Traffic configured with the specified options

4.11 Start Traffic and Get Statistics:

startTraffic - Start traffic allows to modify global traffic options

trafficObj.startTraffic(regenerateTraffic=True, applyTraffic=True)

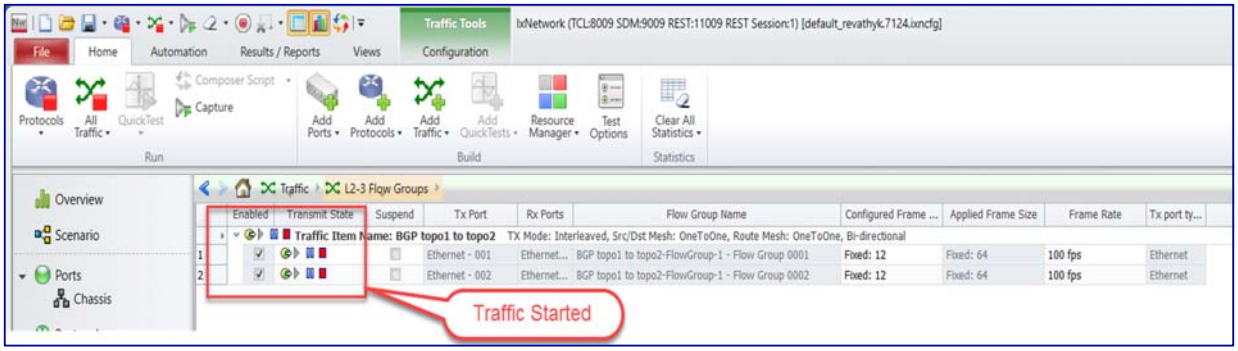


Fig 4.11 Traffic Started

trafficObj.checkTrafficState(expectedState=['stopped'], timeout=45)

getStats - Collect Traffic statistics with the specified options

statObj = Statistics(mainObj)

```
stats = statObj.getStats(viewName='Flow Statistics')

print('\n{txPort:10} {txFrames:15} {rxPort:10} {rxFrames:15}
{frameLoss:10}'.format(txPort='txPort', txFrames='txFrames', rxPort='rxPort',
rxFrames='rxFrames', frameLoss='frameLoss'))
print('-'*90)

for flowGroup,values in stats.items():
    txPort = values['Tx Port']
    rxPort = values['Rx Port']
    txFrames = values['Tx Frames']
    rxFrames = values['Rx Frames']
    frameLoss = values['Frames Delta']
 print('{txPort:10} {txFrames:15} {rxPort:10} {rxFrames:15}
          {frameLoss:10}'.format(txPort=txPort, txFrames=txFrames,
          rxPort=rxPort, rxFrames=rxFrames, frameLoss=frameLoss))
 if releasePortsWhenDone == True:
    portObj.releasePorts(portList)
mainObj.deleteSession()
```

5. To Know More on REST API:

- ✓ https://www.youtube.com/watch?v=f3KnTJ6GS1Q
- ✓ https://www.openixia.com/ixNetworkRestOverview/
- ✓ https://github.com/OpenIxia/IxNetwork/tree/master/RestApi/Python/SampleScripts

6. Support:

- ✓ For more information: https://support.ixiacom.com/
- ✓ For support assistance, contact : support.ix@keysight.com