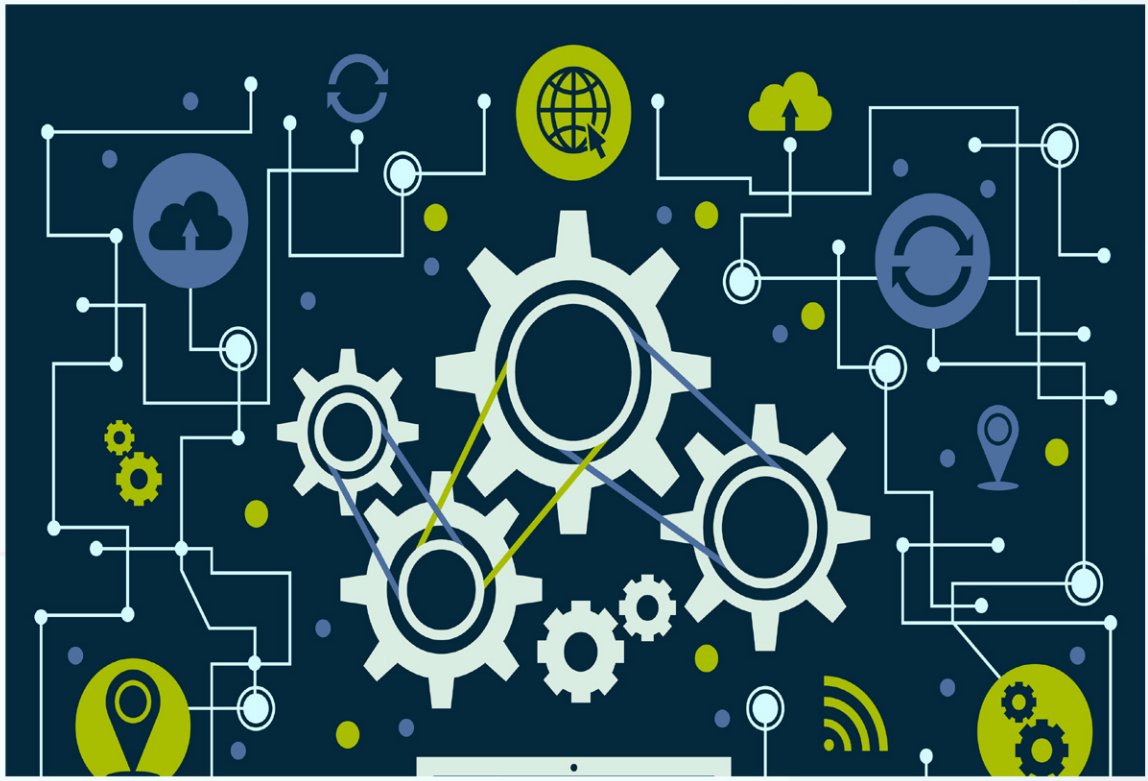
IXNETWORK-CLASSIC
QUICK REFERENCE GUIDE



ixia | A Keysight Business

PREFACE

1. Overview

IxNetwork is a comprehensive network infrastructure performance testing solution. It scales to handle most powerful devices and largest networks, from routing and switching to data center ethernet and software defined networking. IxNetwork has to protocol emulation frameworks for simulating networks. Both are capable of emulating a wide array of protocol activities running on simulated network devices. The next generation protocol framework (NGPF) is the latest generation of protocol emulation engine and supports the widest array of protocols with the highest scalability in the industry. The previous protocol framework (Classic) continues to be supported in IxNetwork, primarily to support existing customers with legacy configurations created prior to NGPF introduction. Ixia recommends using NGPF for all new network infrastructure test projects.

IxNetwork Classic protocol framework provides the following features and benefits:

- IxNetwork Classic offers the performance and functionality testing of Routers/Switches.
- Provides a powerful, yet easy-to-use, graphical user interface (GUI) that you can use to configure and run complex tests.
- Offers the flexibility to customize the application to meet a wide range of requirements for testing complex network topologies, consisting of thousands of routing or switching devices.
- Emulate millions of routes and reachable hosts within the topology. Provides with the ability to customize millions of traffic flows to stress the data plane performance.
- Create sophisticated configurations using powerful wizards and grid controls in GUI.
- Capable of reporting comprehensive protocol status and detailed per-flow traffic performance metrics.

2. Configure OSPFv2 through GUI

This section visualizes the scenario to configure OSPFv2 protocol through GUI and verify IPv4 traffic. Section includes the following tasks.

- Add chassis and reserve ports.
- Configure OSPFv2 protocol on interfaces. Disable 'Discard Learned LSAs' on port 1/1/7 for the peer (1/1/8) to learn LSAs. Create route ranges to advertise prefixes to the peer.
- Start OSPFv2 protocol. Once OSPFv2 protocol sessions UP, check for learned LSAs in the peer (1/1/8) side. Please refer Fig 7.1.
- Create IPv4 unicast traffic (1/1/7 -> 1/1/8). Start traffic and wait for some time for the traffic to flow. Stop the traffic. Verify Tx == Rx packets count from Traffic Item Statistics section.

2.1 Add Chassis and Reserve Ports

Add Chassis and reserve ports.

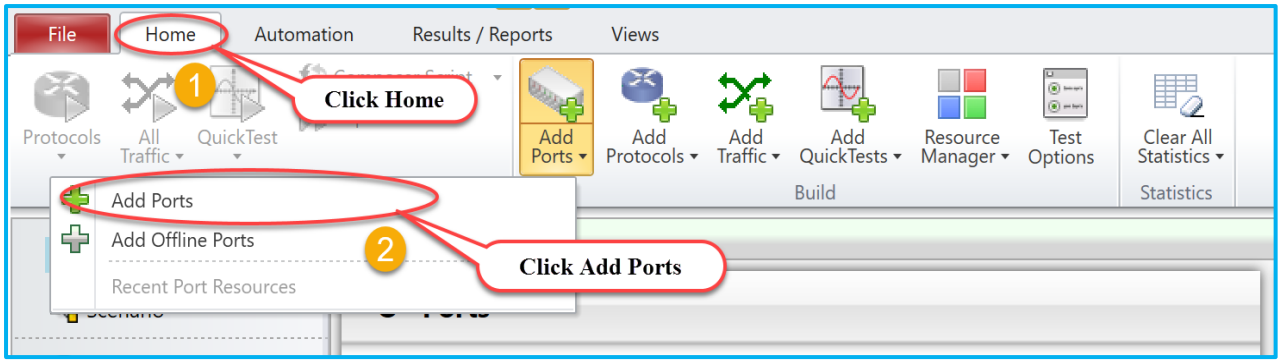


Fig 1.1 Add Ports

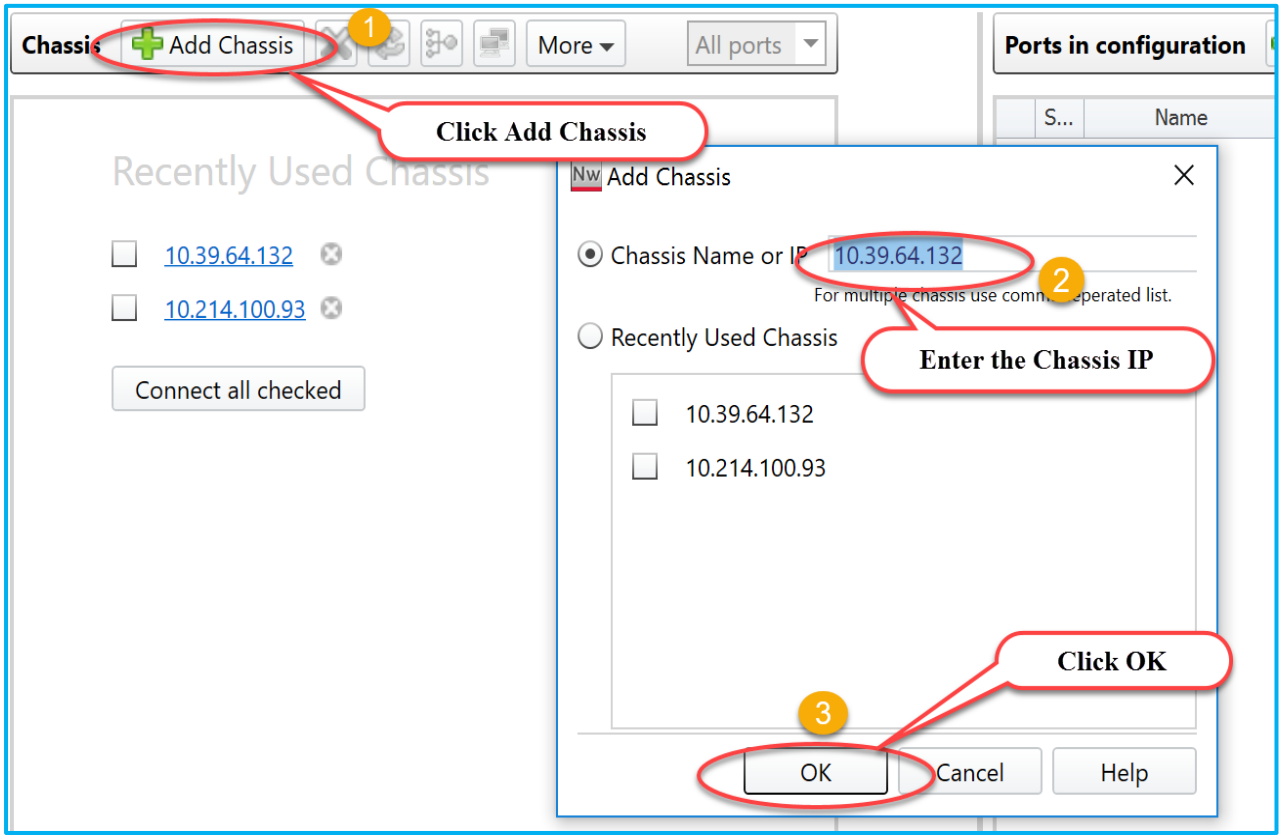


Fig 1.2: Add Chassis to reserve ports

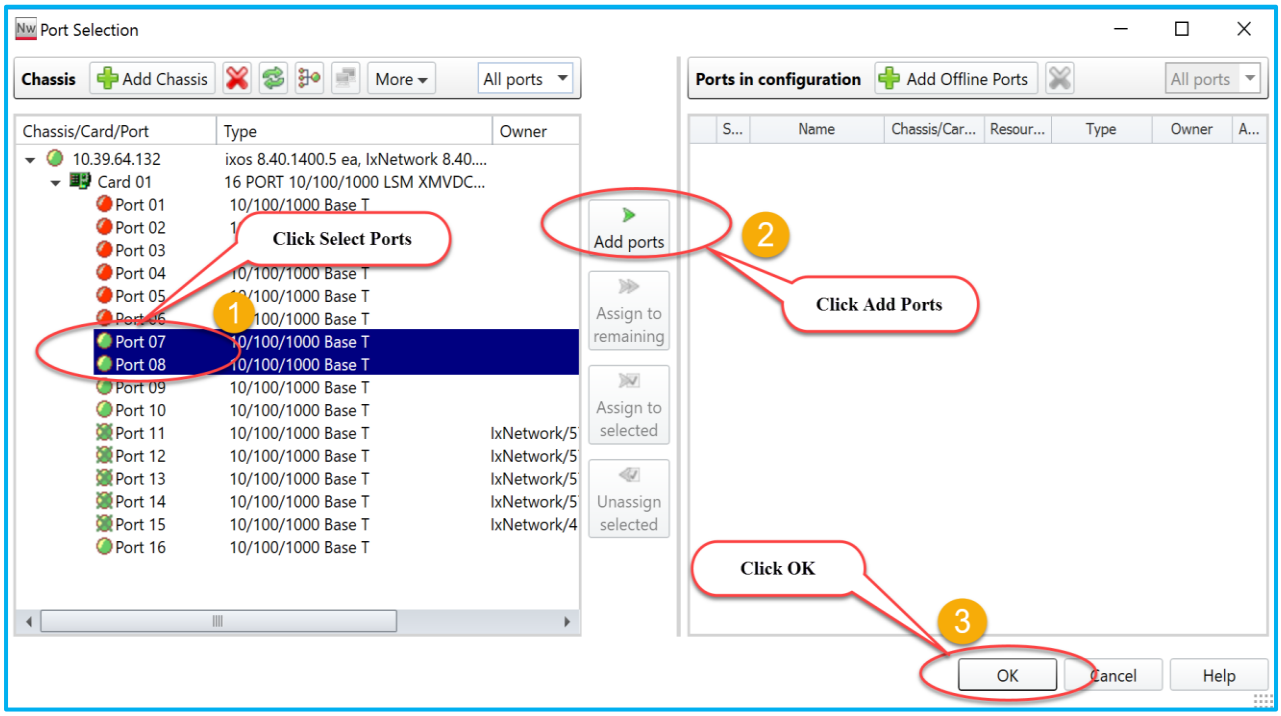


Fig 1.3: Select Ports and Reserve Ports

2.2 Configure Interfaces

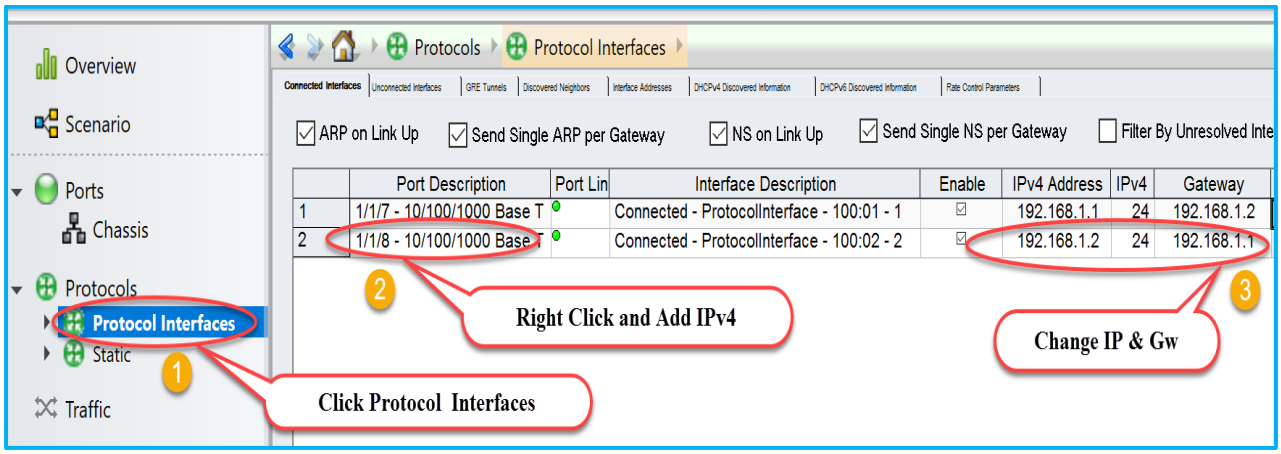


Fig 2.1: Configure Port's attributes

2.3 Emulate OSPFv2 Protocol

Enable OSPFv2 on interfaces.

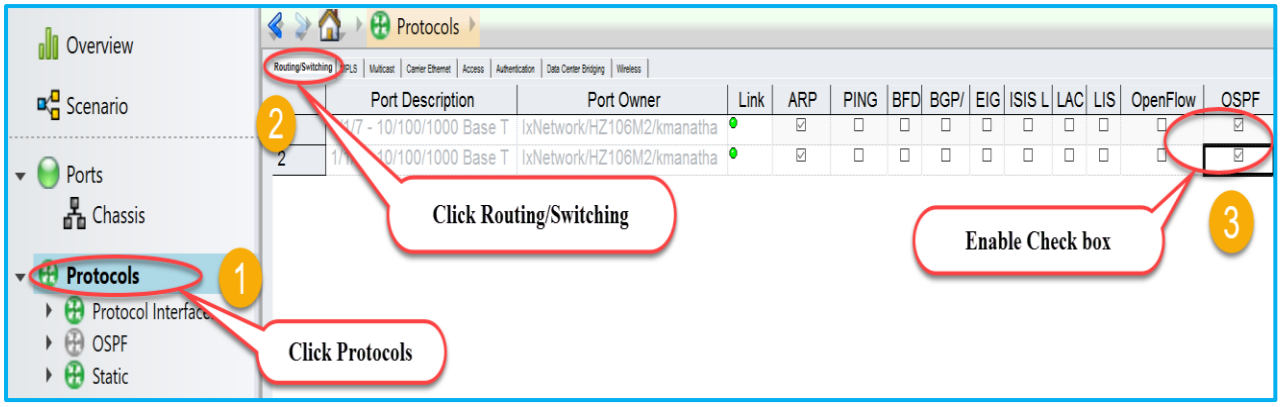


Fig 3.1: Emulate OSPFv2

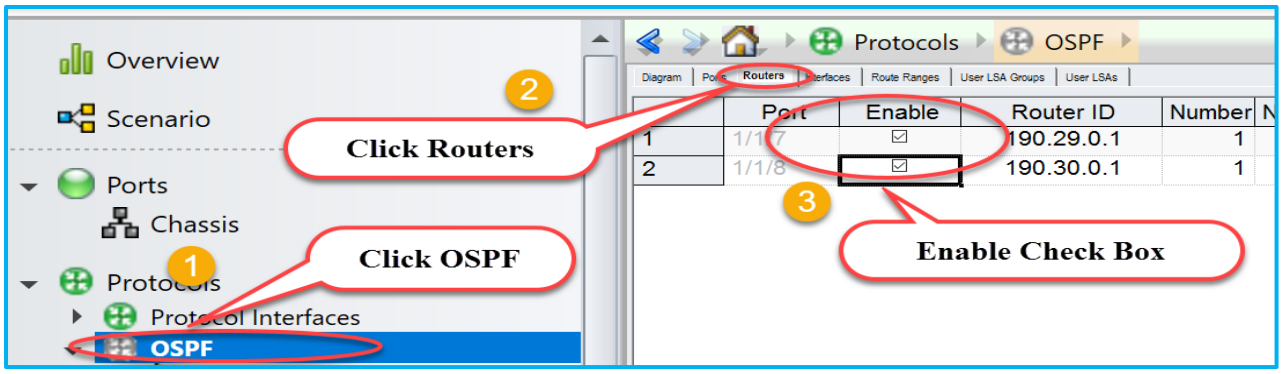


Fig 3.2: Enable OSPFv2 on interfaces

2.4 Configure OSPFv2

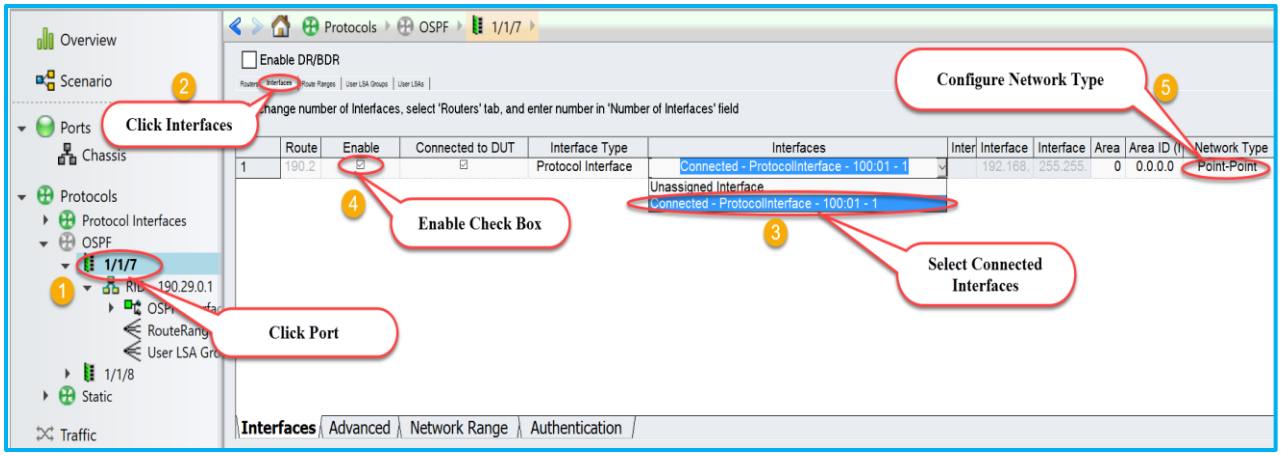


Fig 4.1: Map connected interfaces to configure OSPFv2 attributes

2.5 Create OSPFv2 Route Ranges

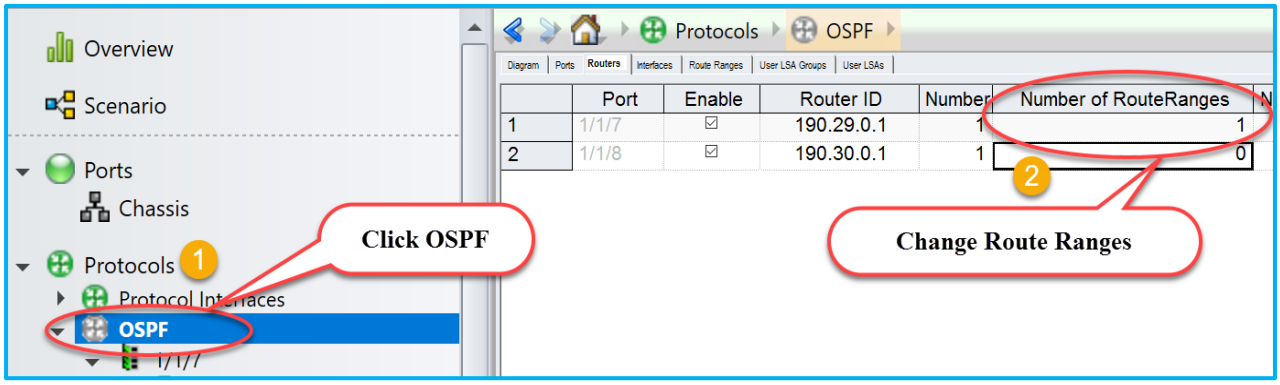


Fig 5.1: Configure number of route ranges required

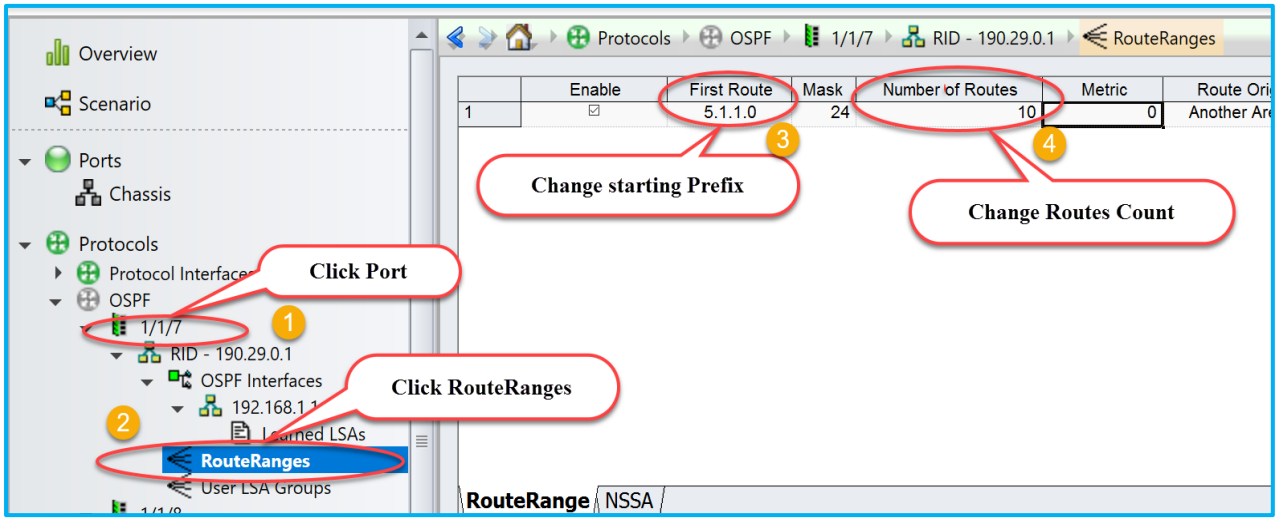


Fig 5.2: Create Ipv4 prefix pool on selected port 1/1/7

2.6 Start OSPFv2 Protocol

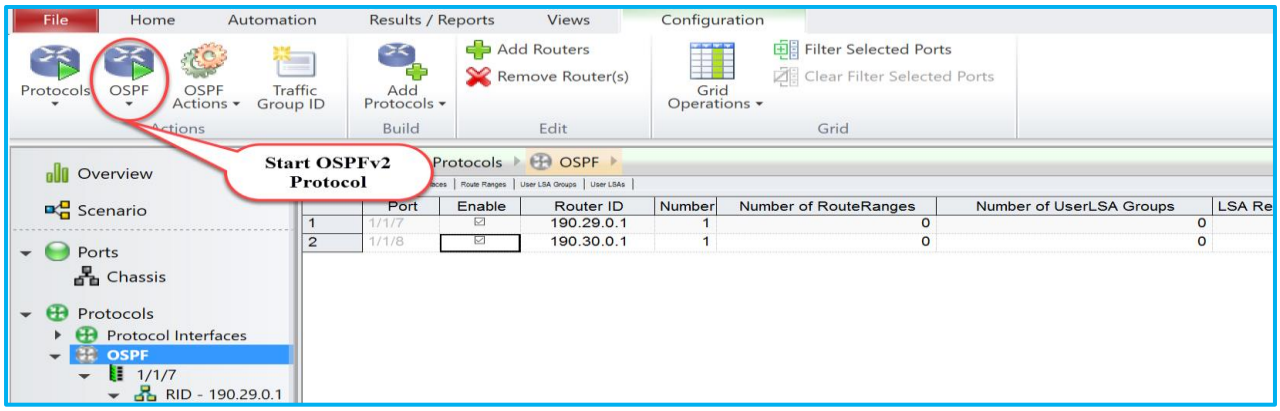


Fig 6.1: Start OSPFv2 protocols

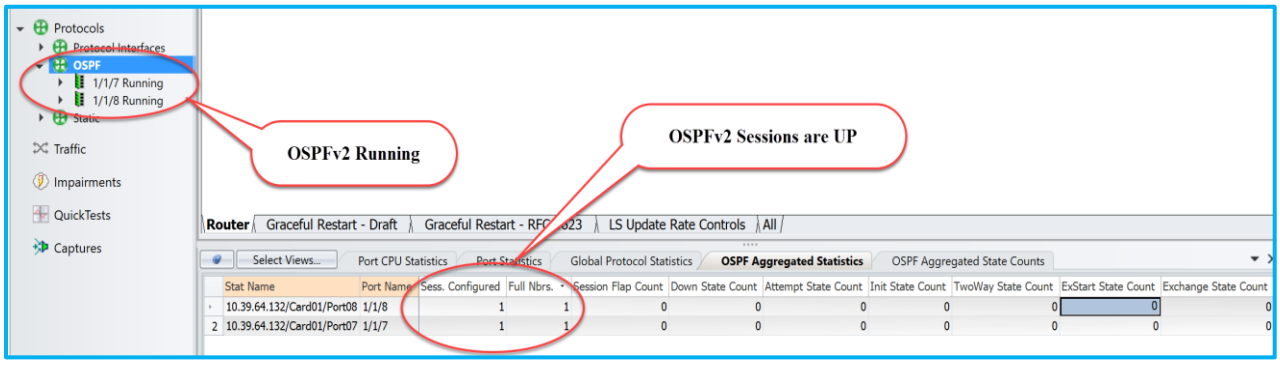


Fig 6.2 OSPFv2 is in Running state and sessions are UP

2.7 Check Learned LSAs

Disable “Discard Learned LSAs” to view learned LSAs on 1/1/8. Please refer Fig 7.1

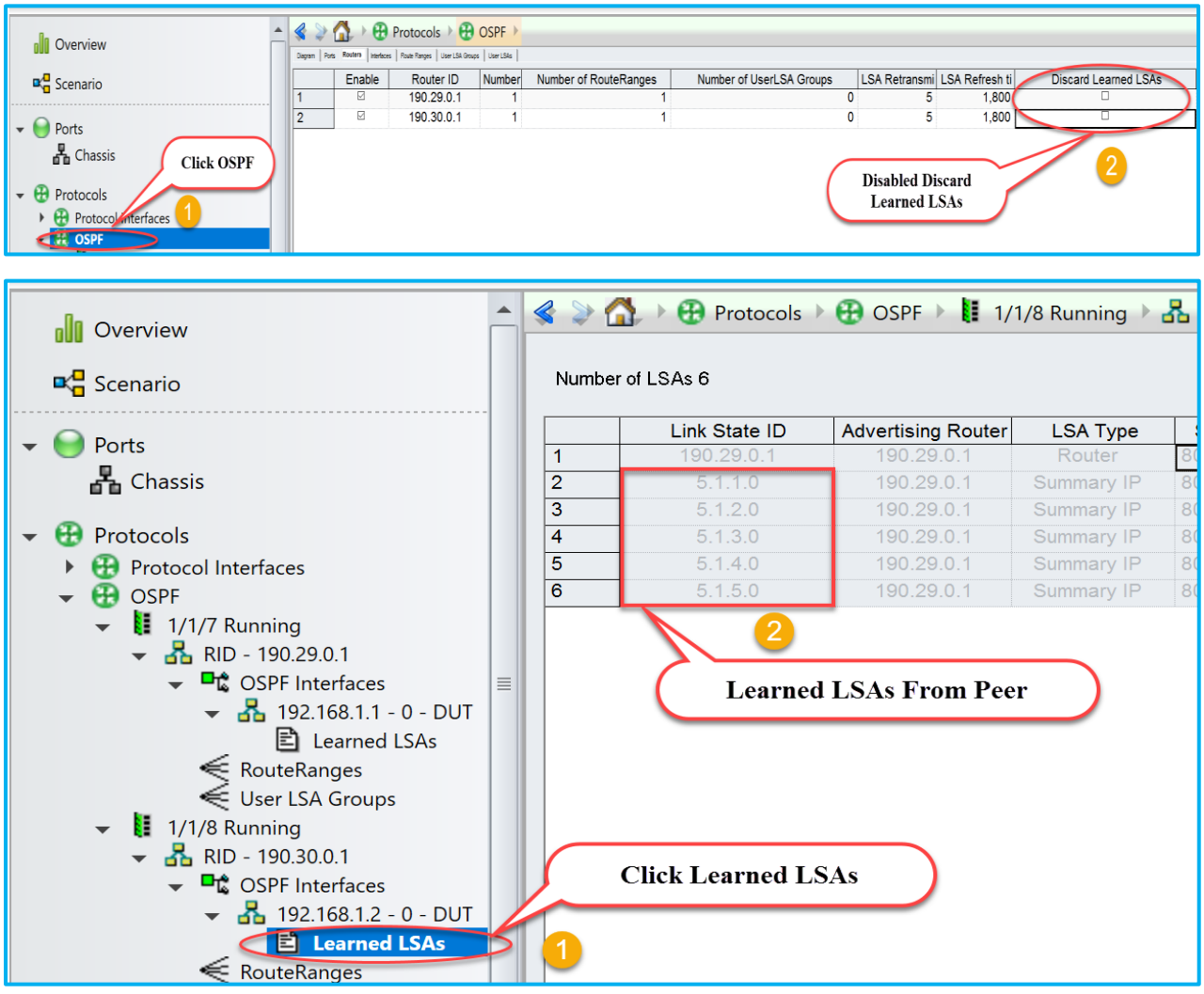


Fig 7.1 Disable discard Learned LSA and check Learned LSAs on 1/1/8 from peer router 1/1/7

2.8 Configure Traffic

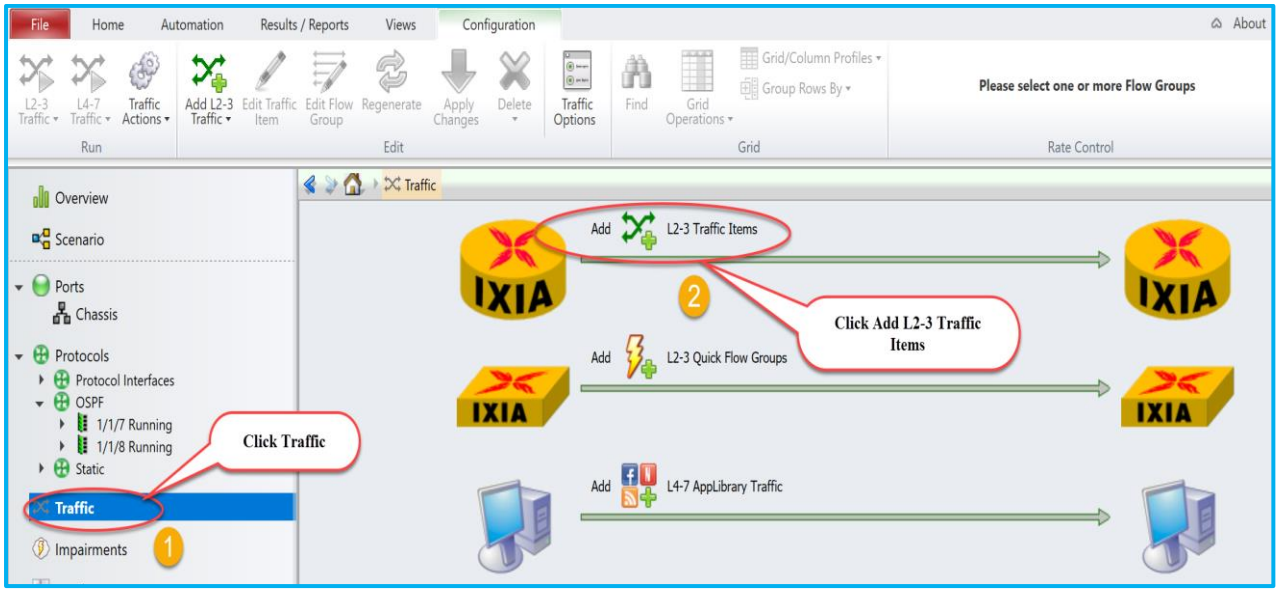


Fig 8.1: Create traffic stream

2.9 Add Endpoints To Traffic

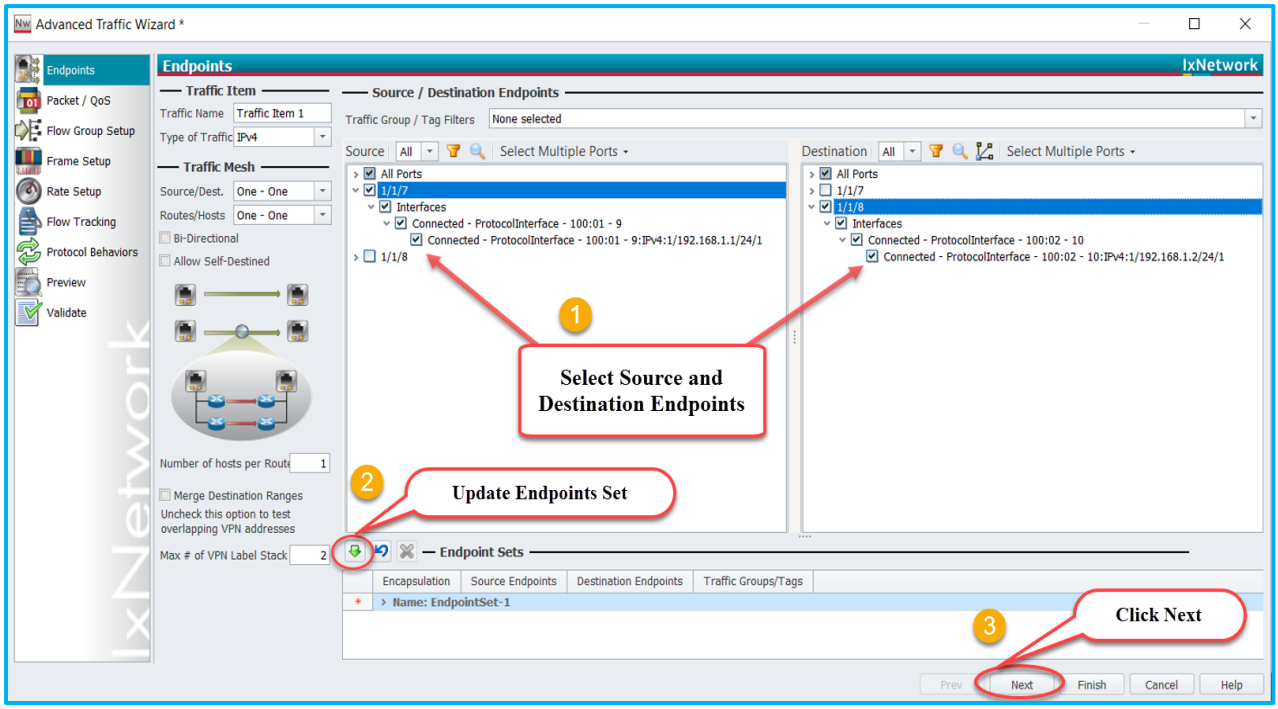


Fig 8.2: Setup source and destination endpoints

2.10 Edit Packet

*Edit Packet and Flow Group Setup are optional.

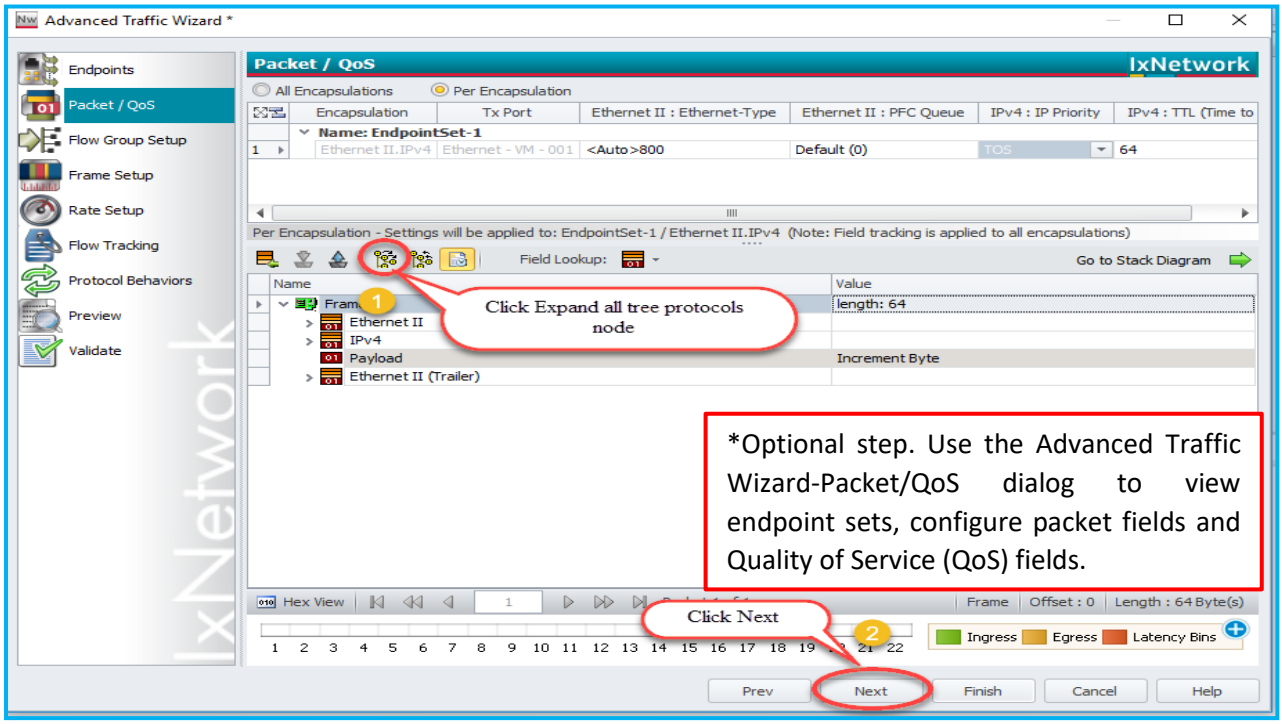


Fig 8.3: Edit packet header

2.11 Setup Flow Group

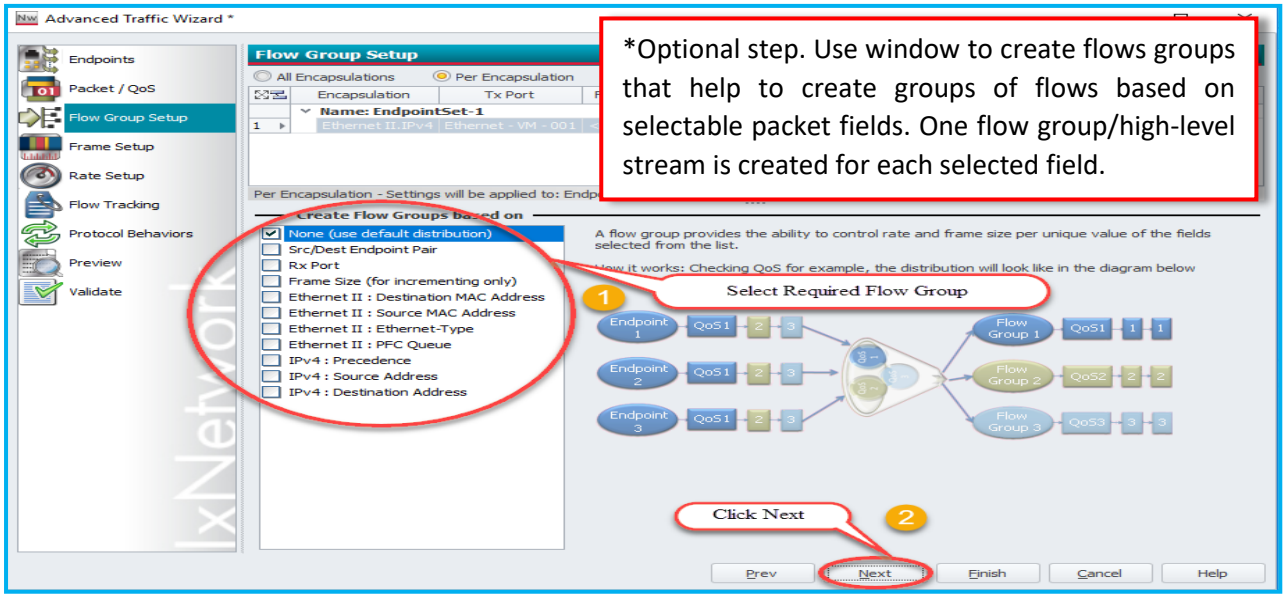


Fig 8.4: Setup flow group

2.12 Setup Frame Size

*Setup frame Size and Line rate are optional.

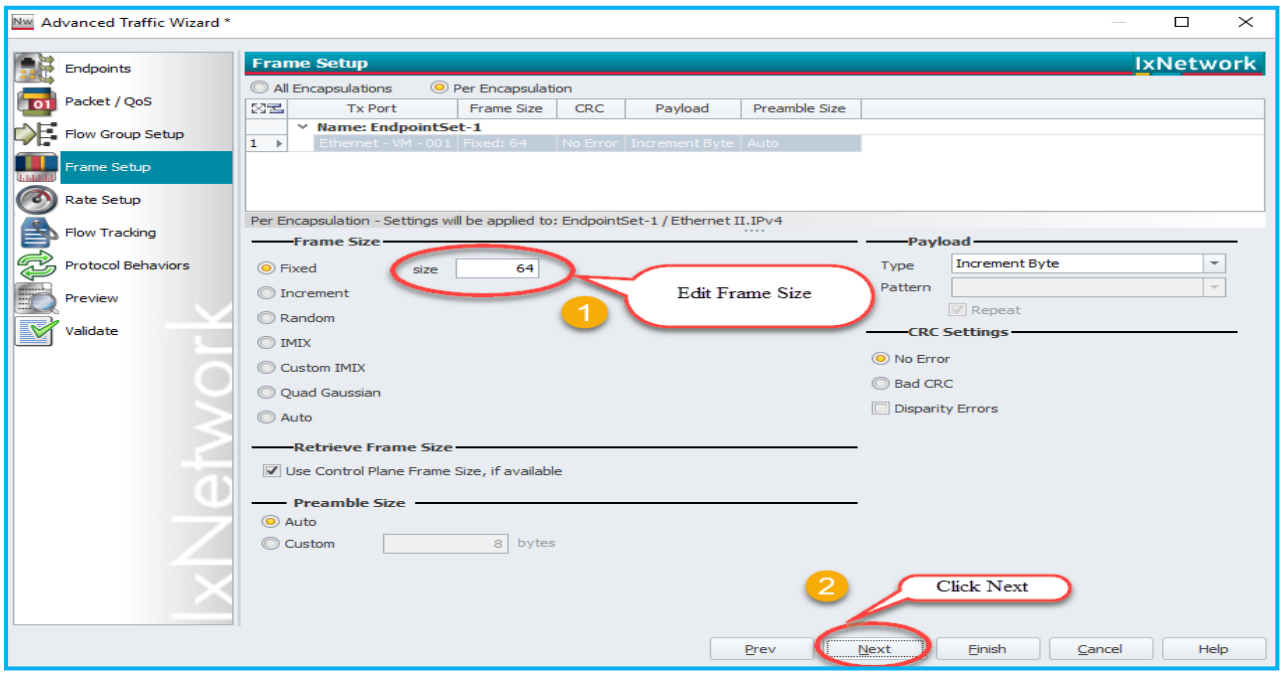
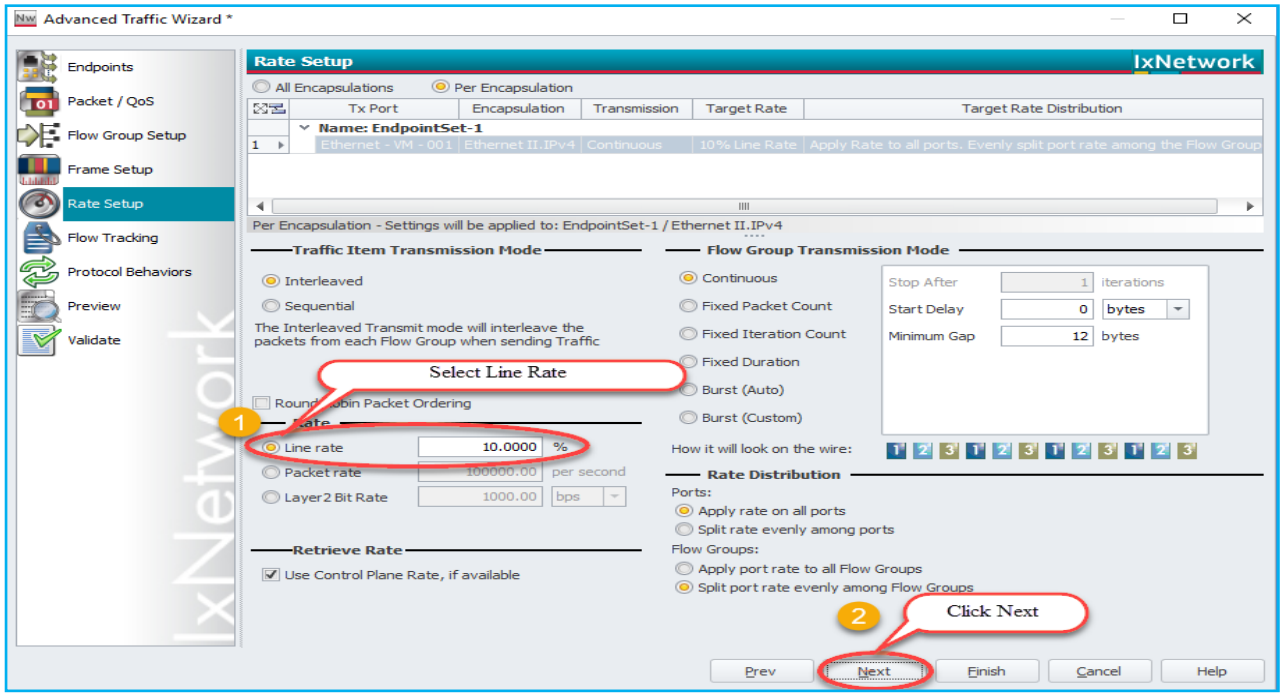


Fig 8.5: Setup Frame size as per test scenario

2.13 Setup Line Rate



8.6: Setup line rate

2.14 Setup Flow Tracking

*Setup Flow Tracking and Protocol Behaviors are optional.

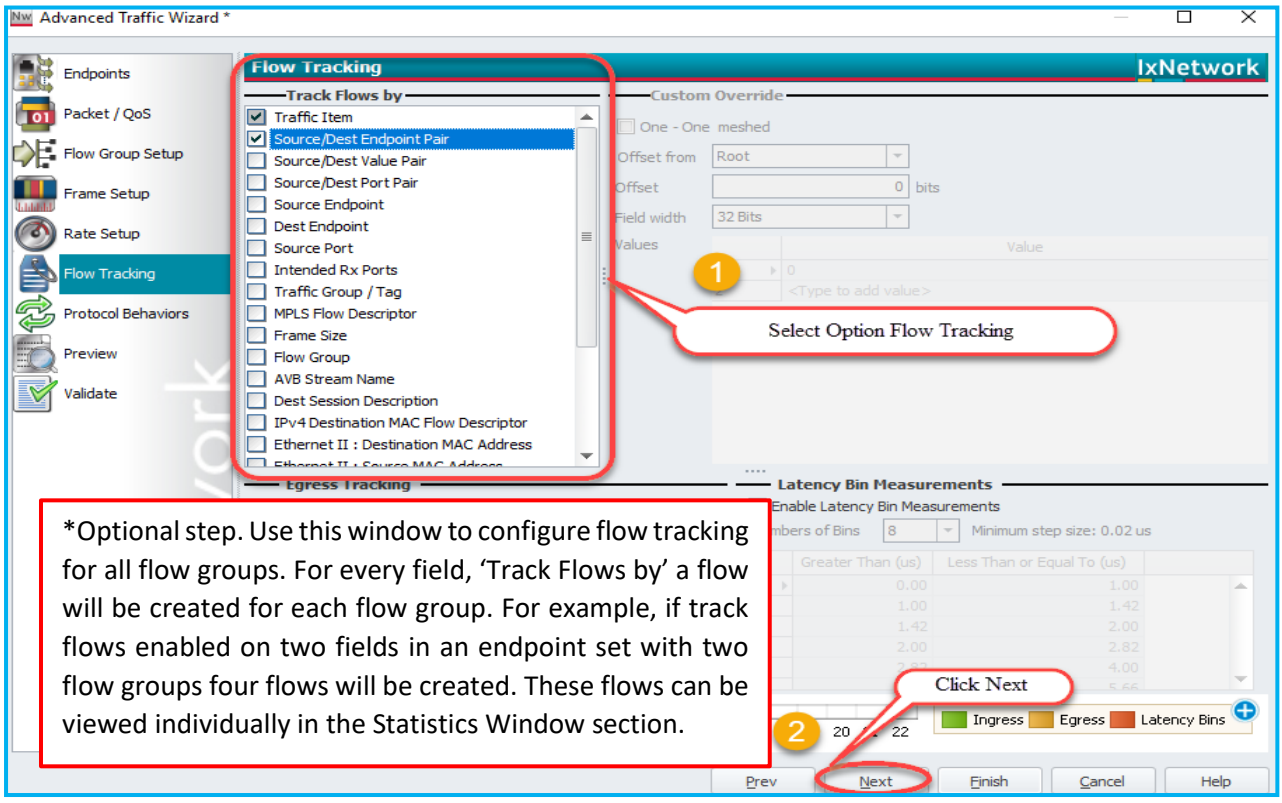


Fig 8.7 Setup flow tracking to track specific field

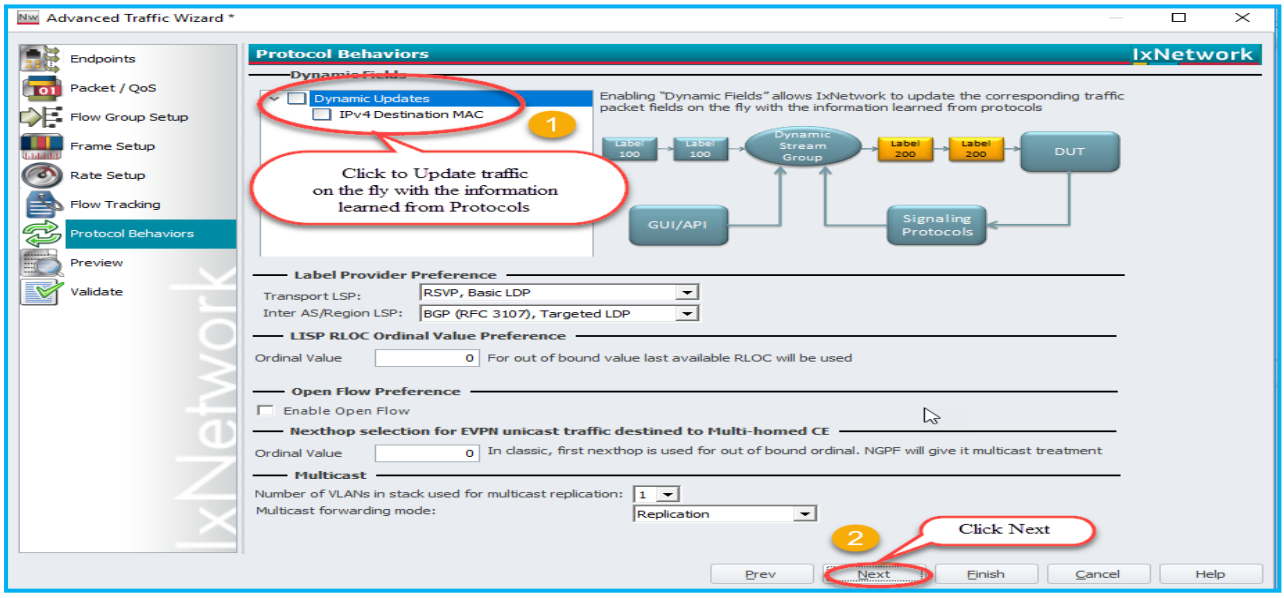


Fig 8.8 Update traffic on the fly with information learned from protocols

2.15 Preview Flow Groups

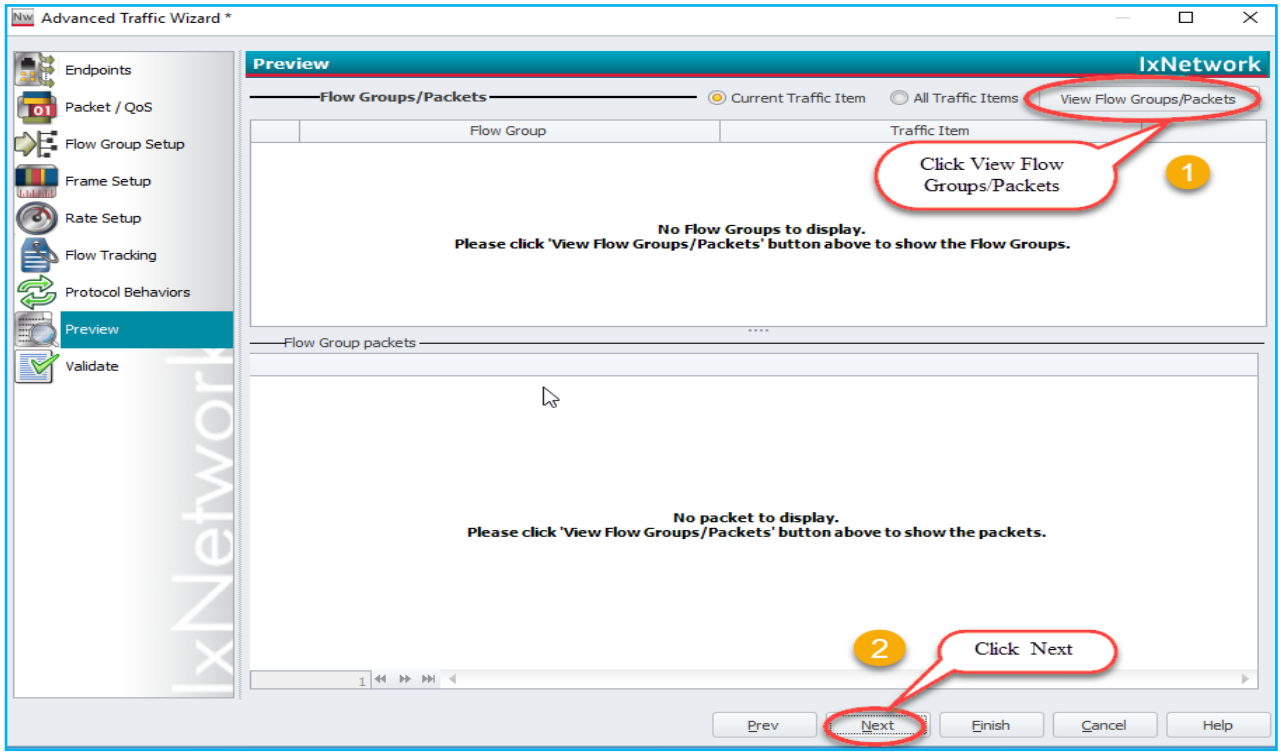


Fig 8.9 View flow group which is currently configured

2.16 Validate Traffic Items

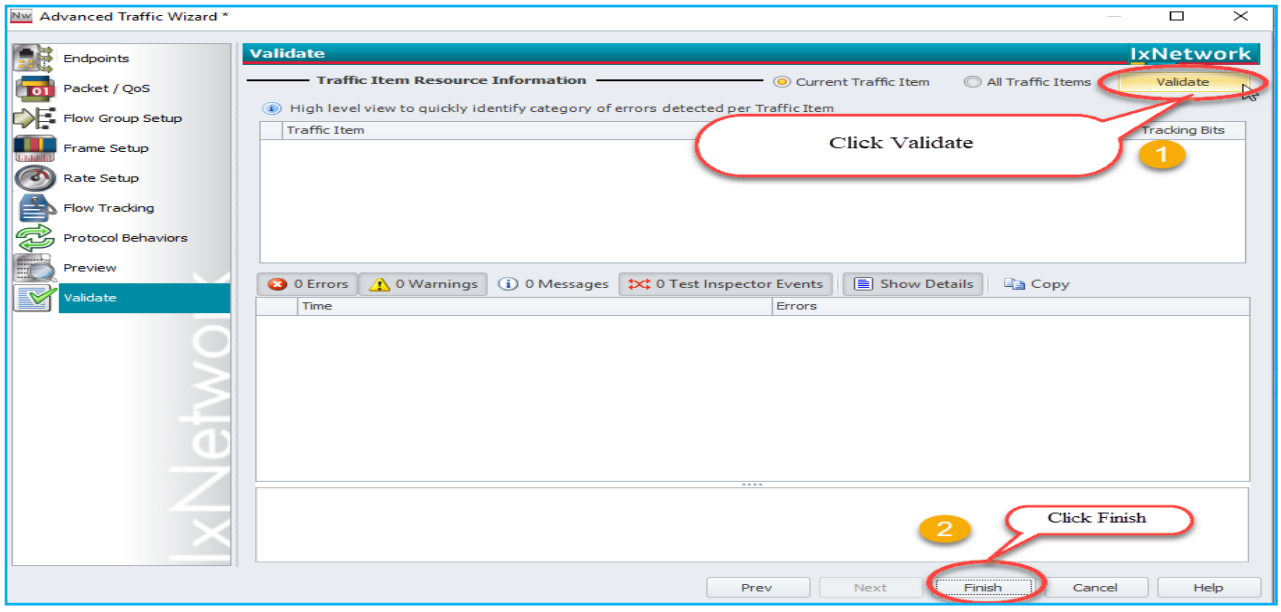


Fig 8.10 Validate the traffic item to identify errors

2.17 Apply Traffic, Start Traffic and Statistics View

Click this 

button to regenerate the traffic before apply. If this button clicked, IxNetwork detects conflicts between existing and newly-generated flow groups (If created) within the traffic item and accept the newly-generated flow groups if needed.

Apply the L2-L3 Traffic by selecting **Apply L2-L3 Traffic** from the L2-3Traffic drop-down menu (on the **Traffic Tools** tab) and Start L2-L3 Traffic.

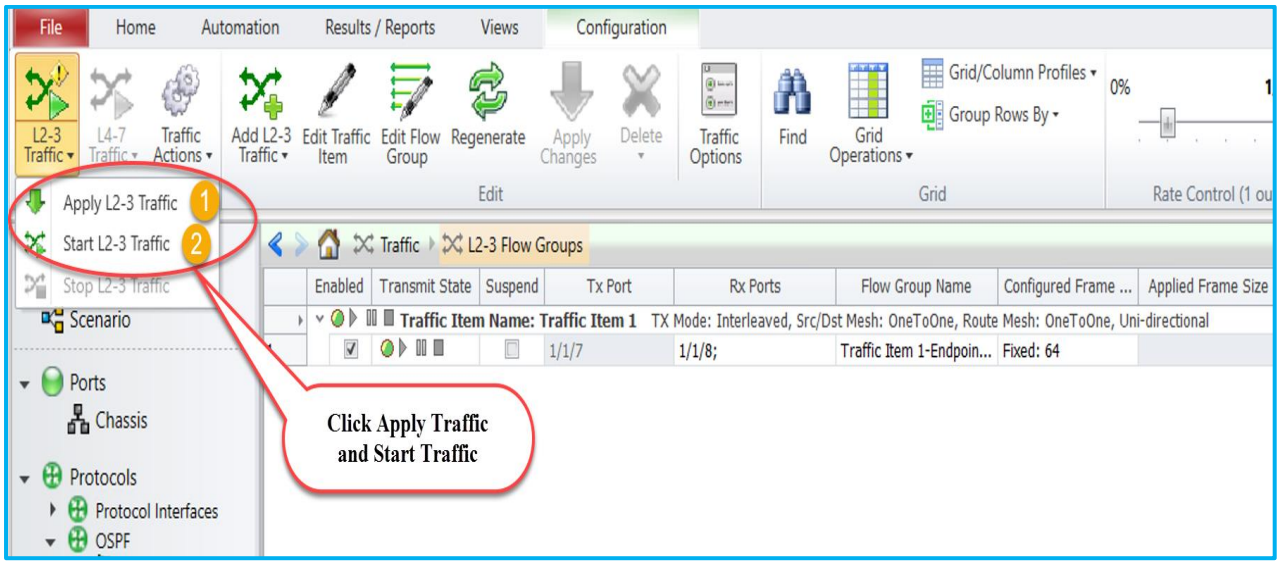


Fig 9.1 Apply and start traffic

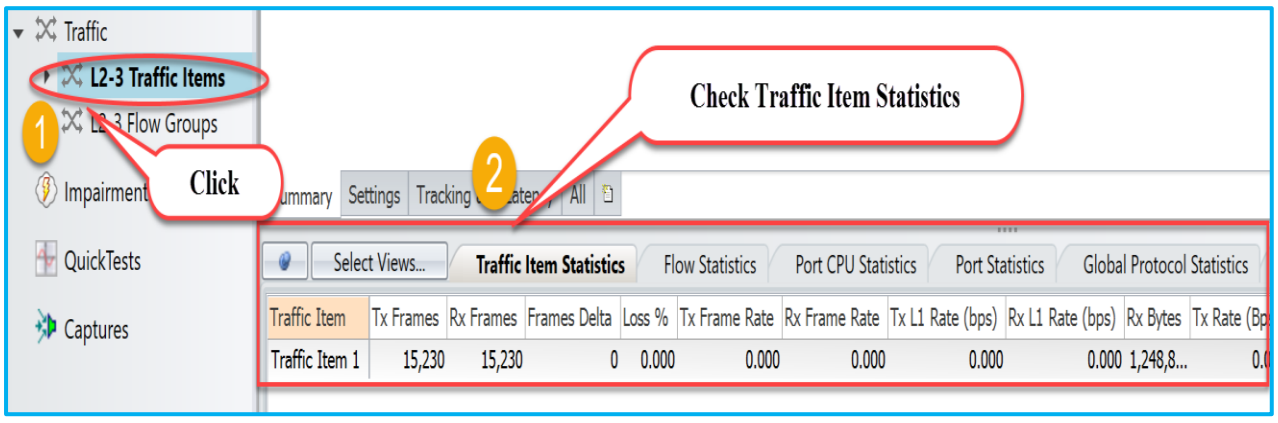


Fig 9.2 Verify traffic statistics

3. Configure OSPFv2 through Automation

IxNetwork provides a wide array of automation APIs. The REST API is the recommended method for creating new test scripts for IxNetwork. Please refer to the IxNetwork REST API quick reference guide for more details. The legacy API libraries, including Tcl, and Python traditional API libraries continue to be supported for existing test scripts.

This section explains the method to automate OSPFv2 emulation scenario through High Level TCL/Python APIs. Covers common and OSPFv2 specific HLAPIs used in IxNetwork classic framework.

3.1 Initialize Environment

Source Ixia package and proceed with HLTAPI execution.

TCL

```
> package require Ixia
Tcl 8.5 is installed on 64bit architecture.
IXIA_VERSION env variable is set to 8.20.0.10, but this value is not matching any HLTSET.
Using default HLTSET (HLTSET210) instead.
Using products based on HLTSET210
IxTclHal is not be used for current HLTSET.
Loaded IxTclNetwork 8.20.1071.8
Loaded Mpexpr 1.0
HLT release 8.20.136.2
Loaded ixia_hl_lib-8.20
8.20
> package require Tclx
8.4
```

PYTHON

```
import sys, os
import time, re
from ixiatcl   import IxiaTcl
from ixiahlt   import IxiaHlt
from ixiaerror import IxiaError
tcl_dependencies = ['/usr/local/lib/', '/usr/lib/',
    '/usr/share/tcl8.5', '/usr/lib/tcl8.5', '/usr/lib/tk8.5',
    '/usr/share/tk8.5' ]
ixiatcl = IxiaTcl(tcl_autopath=tcl_dependencies)
ixia  = IxiaHlt(ixiatcl, use_legacy_api = 1)

Note: If python version > 3.4, call IxiaTcl with Tcl 8.6
path. Example: tcl_dependencies = ['/path/to/tcl8.6'];
ixiatcl = IxiaTcl(tcl_autopath=tcl_dependencies)
ixiatcl = IxiaTcl(); ixia = IxiaHlt(ixiaTcl)
```

3.2 Add Chassis and Reserve Ports

::ixia::connect - Connects to the Ixia Chassis, takes ownership of selected ports and optionally loads a configuration on the chassis or resets the targeted ports to factory defaults.

TCL

```
> set connect_result [::ixia::connect          \
   -device 10.39.64.132                        \
   -port_list {1/7 1/8}                        \
   -reset 1                                    \
   -ixnetwork_tcl_server 10.154.161.223:8009   \
   -tcl_server 10.154.161.223]
> set ports [keylget connect_result vport_list]
> set port1 [lindex $ports 0]
> set port2 [lindex $ports 1]
```

PYTHON

```
> connect_result = ixia.connect (              \
   device = 10.39.64.132,                      \
   port_list = "1/7 1/8",                      \
   reset = 1                                   \
   ixnetwork_tcl_server = "10.154.161.223:8009", \
   tcl_server    = 10.154.161.223)
> ports = connect_result['vport_list'].split()
> port1 =  ports[0]
> port2 =  ports[1]
```

**Note: High Level API's are highlighted in Red and all other handles are highlighted in Green.*

Now chassis are connected and reserved ports (1/7 & 1/8). Please refer Fig 10.1

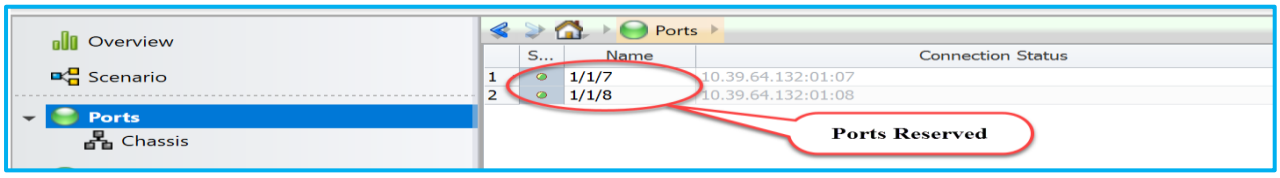


Fig 10.1 Connected to chassis and reserved ports

3.3 Configure Ports

::ixia::interface_config - Configures an interface and accommodates addressing schemes such as IPv4, IPv6, MAC and VLAN.

TCL

```
> set intf_cfg_stats1 [ ::ixia::interface_config  \
                        -port_handle $port1       \
                        -intf_ip_addr 198.168.1.2 \
                        -gateway 198.168.1.1      \
                        -arp_send_req 1           \
                        -autonegotiation 1        \
                        -phy_mode fiber]

> set intf_cfg_stats2 [ ::ixia::interface_config  \
                        -port_handle $port2       \
                        -intf_ip_addr 198.168.1.1 \
                        -gateway 198.168.1.2      \
                        -arp_send_req 1           \
                        -autonegotiation 1        \
                        -phy_mode fiber]
```

PYTHON

```
> intf_cfg_stats1 = ixia.interface_config (  \
                    port_handle = port1,     \
                    intf_ip_addr = 198.168.1.2, \
                    gateway = 198.168.1.1,   \
                    arp_send_req = 1,        \
                    autonegotiation = 1,     \
                    phy_mode = 'fiber')

> intf_cfg_stats2 = ixia.interface_config (  \
                    port_handle = port2,     \
                    intf_ip_addr = 198.168.1.1, \
                    gateway = 198.168.1.2,   \
                    arp_send_req = 1,        \
                    autonegotiation = 1,     \
                    phy_mode = 'fiber')
```

Fig 11.1 Configure ports attributes using HLAPI

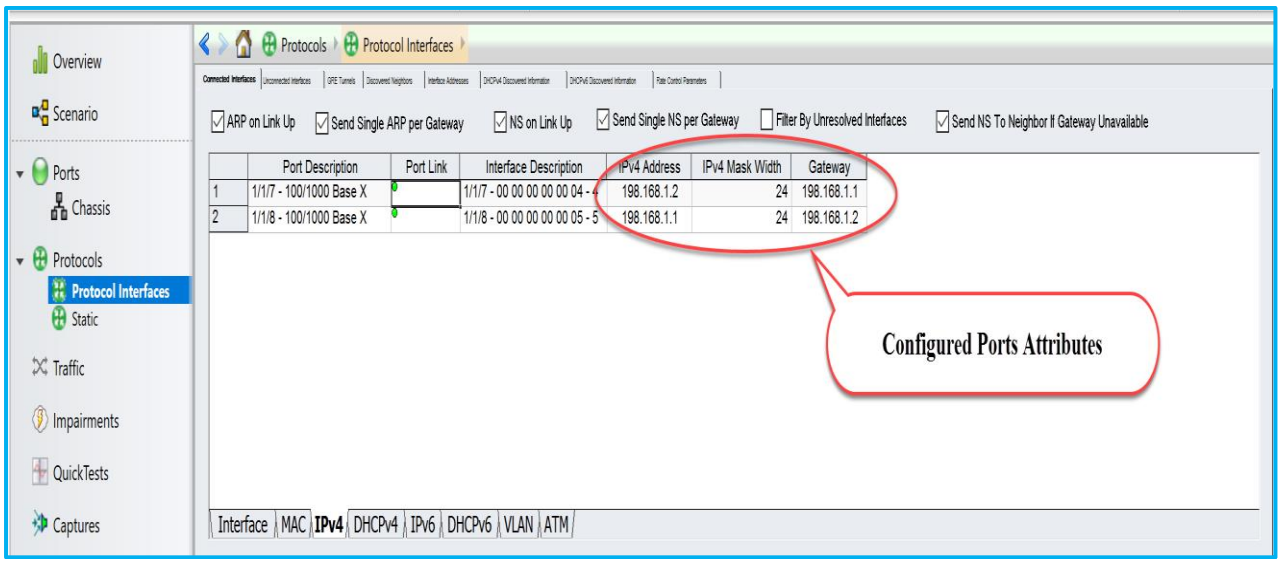


Fig 11.2 View configured attributes in GUI

3.4 Create OSPFv2

::ixia::emulation_ospf_config - Add ospf router(s) to a port and configure OSPFv2 attributes. Disable 'discard learned lsa' to view learned lsa's on peer router.

TCL

```
> set ospf_emul_res1   \
[::ixia::emulation_ospf_config                  \
          -port_handle $port1                   \
          -mode create                          \
          -session_type ospfv2                  \
          -area_id 0.0.0.1                      \
          -router_id 190.29.0.1                 \
          -network_type ptop                    \
          -lsa_discard_mode 0                   \
          -intf_ip_addr 198.168.1.2             \
          -neighbor_intf_ip_addr 192.168.1.1]
> set ospf_handle1 [keylget ospf_emul_res1   \
handle]

> set ospf_emul_res2   \
[::ixia::emulation_ospf_config                  \
          -port_handle $port2                   \
          -mode create                          \
          -session_type ospfv2                  \
          -area_id 0.0.0.1                      \
          -router_id 190.30.0.1                 \
          -network_type ptop                    \
          -lsa_discard_mode 0                   \
          -intf_ip_addr 198.168.1.1             \
          -neighbor_intf_ip_addr 192.168.1.2]
> set ospf_handle2 [keylget ospf_emul_res2   \
handle]
```

PYTHON

```
> ospf_emul_res1 = \
ixia.emulation_ospf_config (                    \
          port_handle = port1,                  \
          mode = create,                        \
          session_type = 'ospfv2',              \
          area_id = 0.0.0.1,                    \
          router_id = 190.29.0.1,               \
          network_type = 'ptop',                \
          lsa_discard_mode = 0,                 \
          intf_ip_addr = 198.168.1.2,           \
          neighbor_intf_ip_addr = 192.168.1.1)
> ospf_handle1 = ospf_emul_res1['handle']


> ospf_emul_res2 = \
ixia.emulation_ospf_config (                    \
          port_handle = port2,                  \
          mode = create,                        \
          session_type = 'ospfv2',              \
          area_id = 0.0.0.1,                    \
          router_id = 190.30.0.1,               \
          network_type = 'ptop',                \
          lsa_discard_mode = 0 ,                \
          intf_ip_addr = 198.168.1.1,           \
          neighbor_intf_ip_addr = 192.168.1.2)
> ospf_handle2 = ospf_emul_res2['handle']
```

Fig 12.1 Configure OSPFv2 attributes using HLAPI

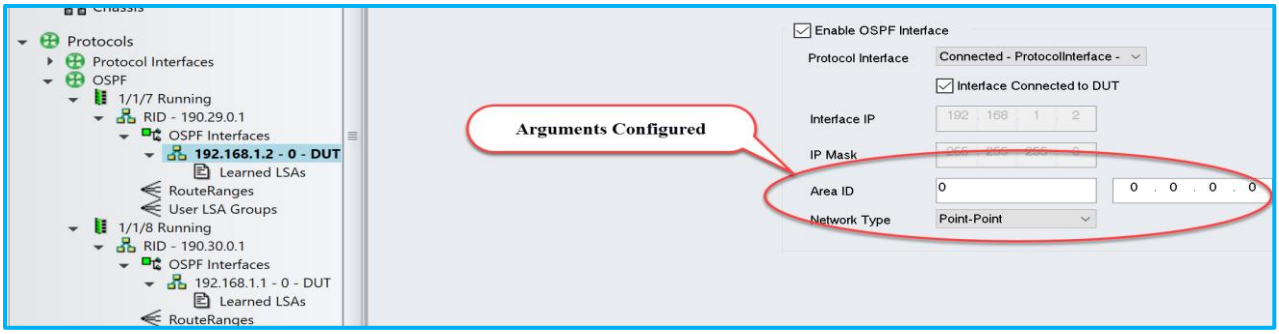


Fig 12.2 View configured ospfv2 attributes on 1/1/7 in GUI

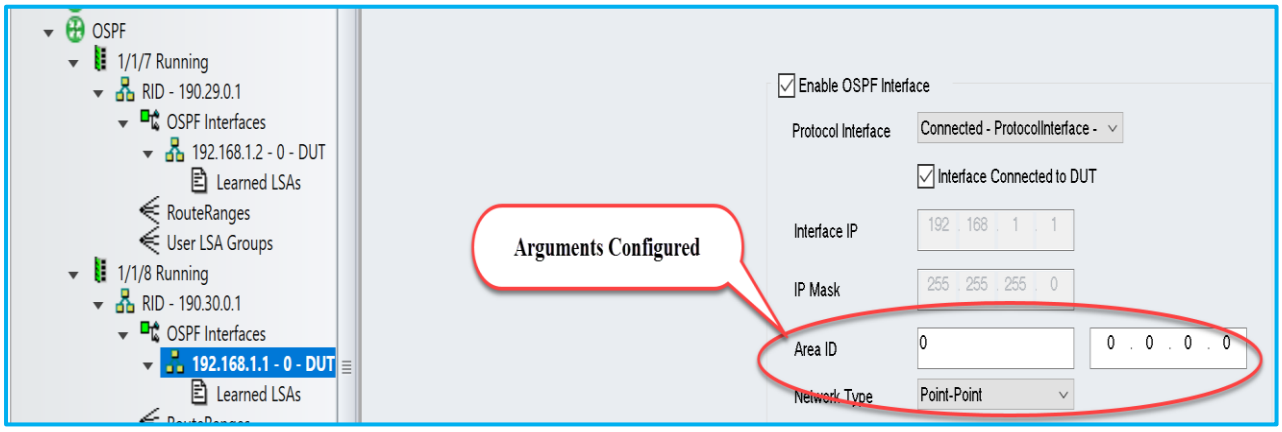


Fig 12.3 View configured ospfv3 attributes on 1/1/8 in GUI

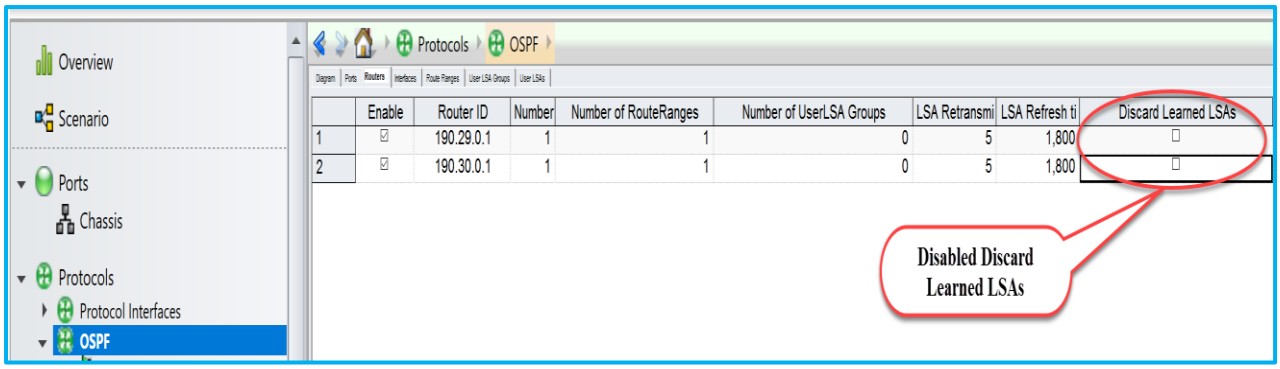


Fig 12.4 View Discard Learned LSAs disbaled in GUI

3.5 Create OSPFv2 Route Ranges

::ixia::emulation_ospf_topology_route_config - Add OSPFv2 route(s) to a simulated OSPFv2 router interface.

TCL	PYTHON
*> set ospf_routes * *[::ixia::emulation_ospf_topology_route_config * *-mode create * *-handle $ospf_handle1 * *-type ext_routes * *-external_number_of_prefix 1 * *-external_prefix_length 24 * *-external_prefix_start 5.1.1.0 * *-external_prefix_step 1]*	*> ospf_routes = * *ixia.emulation_ospf_topology_route_config (* *mode = 'create', * *handle = ospf_handle1, * *type = ext_routes, * *external_number_of_prefix = 1, * *external_prefix_length = 24, * *external_prefix_start = 5.1.1.0, * *external_prefix_step 1)*

Fig 13.1 Configure OSPFv2 Route Ranges on 1/1/7 ospf handle using HLAPI

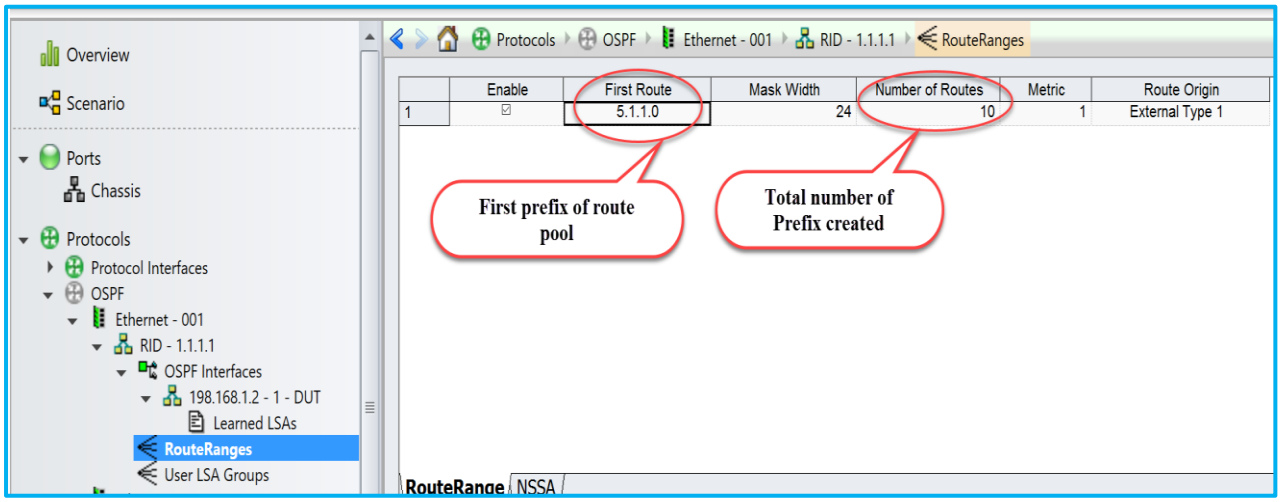


Fig 13.2 View configured route ranges in 1/1/7 (Ethernet – 001)

3.6 Start Protocols

::ixia::emulation_ospf_control - Start OSPF protocol.

TCL	PYTHON
*> set result1 [::ixia::emulation_ospf_control * *-handle $ospf_handle1 * *-port_handle $port1 * *-mode start]* *> set result2 [::ixia::emulation_ospf_control * *-handle $ospf_handle2 * *-port_handle $port2 * *-mode start]* *> puts "Wait for some time for the protocols to converge"*	*> result1 = ixia.emulation_ospf_control (* *handle = ospf_handle1, * *port_handle = port1, * *mode = 'start')* *> result2 = ixia.emulation_ospf_control (* *handle = ospf_handle2, * *port_handle = port2, * *mode = 'start')* *> print """Wait for some time for the protocols to converge"""*

Fig 14.1 Start OSPFv2 protocols using HLAPI

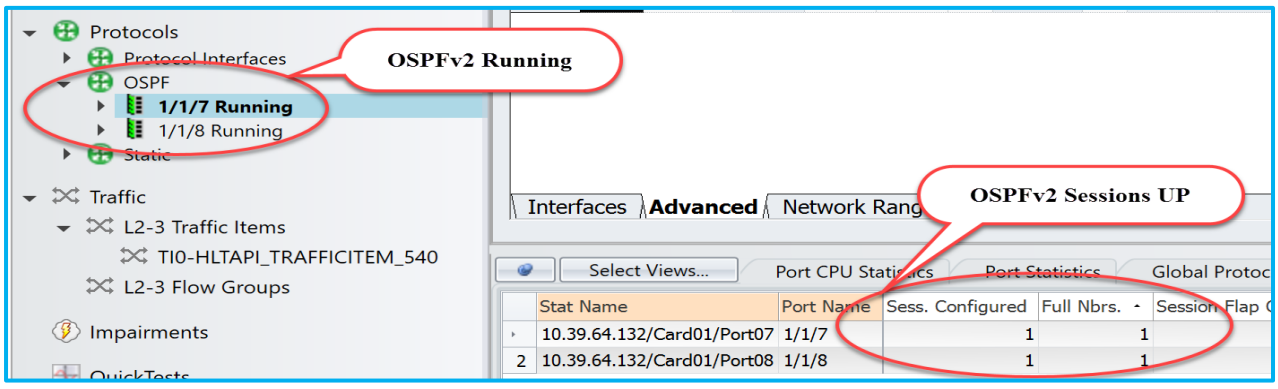


Fig 14.2 View ospfv2 session states details in GUI

3.7 Check Learned LSAs

::ixia::emulation_ospf_info - Retrieves information about the OSPF sessions. Please refer Fig 15.2

TCL

```
> set learned_info [::ixia::emulation_ospf_info      \
                                -mode learned_info      \
                               -handle $ospf_handle2]
> set sessions [keylget learned_info
$ospf_handle2.$ $ospf_handle2/interface:1]
> puts "Learned LSAs: $sessions"
        1:
          adv_router_id: 190.29.0.1
          age: 30
          link_state_id: 190.29.0.1
          lsa_type: router
          seq_number: 80000002
          prefix_v4_address: ::ixNet::OK
          prefix_v6_address: ::ixNet::OK
          prefix_length: ::ixNet::OK
        2:
          adv_router_id: 190.29.0.1
          age: 41
          link_state_id: 5.1.1.0
          lsa_type: external
          seq_number: 80000001
          prefix_v4_address: ::ixNet::OK
          prefix_v6_address: ::ixNet::OK
          prefix_length: ::ixNet::OK
        3:
          adv_router_id: 190.29.0.1
          age: 41
          link_state_id: 5.1.2.0
          lsa_type: external
          seq_number: 80000001
          prefix_v4_address: ::ixNet::OK

         ------ Stripped off Long output ---------

        9:
          adv_router_id: 190.29.0.1
          age: 41
          link_state_id: 5.1.8.0
          lsa_type: external
          seq_number: 80000001
          prefix_v4_address: ::ixNet::OK
          prefix_v6_address: ::ixNet::OK
          prefix_length: ::ixNet::OK
```

PYTHON

```
> learned_info = ixia.emulation_ospf_info (      \
                                mode = 'learned_info',      \
                               handle = ospf_handle2)
> sessions =
learned_info[ospf_handle2][ospf_handle2+
'/interface:1']
> print "Learned LSAs: %s" % sessions
1:
          adv_router_id: 190.29.0.1
          age: 30
          link_state_id: 190.29.0.1
          lsa_type: router
          seq_number: 80000002
          prefix_v4_address: ::ixNet::OK
          prefix_v6_address: ::ixNet::OK
          prefix_length: ::ixNet::OK
        2:
          adv_router_id: 190.29.0.1
          age: 41
          link_state_id: 5.1.1.0
          lsa_type: external
          seq_number: 80000001
          prefix_v4_address: ::ixNet::OK
          prefix_v6_address: ::ixNet::OK
          prefix_length: ::ixNet::OK
        3:
          adv_router_id: 190.29.0.1
          age: 41
          link_state_id: 5.1.2.0
          lsa_type: external
          seq_number: 80000001
          prefix_v4_address: ::ixNet::OK

         ------ Stripped off Long output ---------
        9:
          adv_router_id: 190.29.0.1
          age: 41
          link_state_id: 5.1.8.0
          lsa_type: external
          seq_number: 80000001
          prefix_v4_address: ::ixNet::OK
          prefix_v6_address: ::ixNet::OK
          prefix_length: ::ixNet::OK
```

10: *adv_router_id: 190.29.0.1* *age: 41* *link_state_id: 5.1.9.0* *lsa_type: external* *seq_number: 80000001* *prefix_v4_address: ::ixNet::OK* *prefix_v6_address: ::ixNet::OK* *prefix_length: ::ixNet::OK* *11:* *adv_router_id: 190.29.0.1* *age: 41* *link_state_id: 5.1.10.0* *lsa_type: external* *seq_number: 80000001* *prefix_v4_address: ::ixNet::OK* *prefix_v6_address: ::ixNet::OK* *prefix_length: ::ixNet::OK* *status: 1* *log:*	*10:* *adv_router_id: 190.29.0.1* *age: 41* *link_state_id: 5.1.9.0* *lsa_type: external* *seq_number: 80000001* *prefix_v4_address: ::ixNet::OK* *prefix_v6_address: ::ixNet::OK* *prefix_length: ::ixNet::OK* *11:* *adv_router_id: 190.29.0.1* *age: 41* *link_state_id: 5.1.10.0* *lsa_type: external* *seq_number: 80000001* *prefix_v4_address: ::ixNet::OK* *prefix_v6_address: ::ixNet::OK* *prefix_length: ::ixNet::OK* *status: 1* *log:*

Fig 15.1 Retrieve learned LSA info using HLAPI

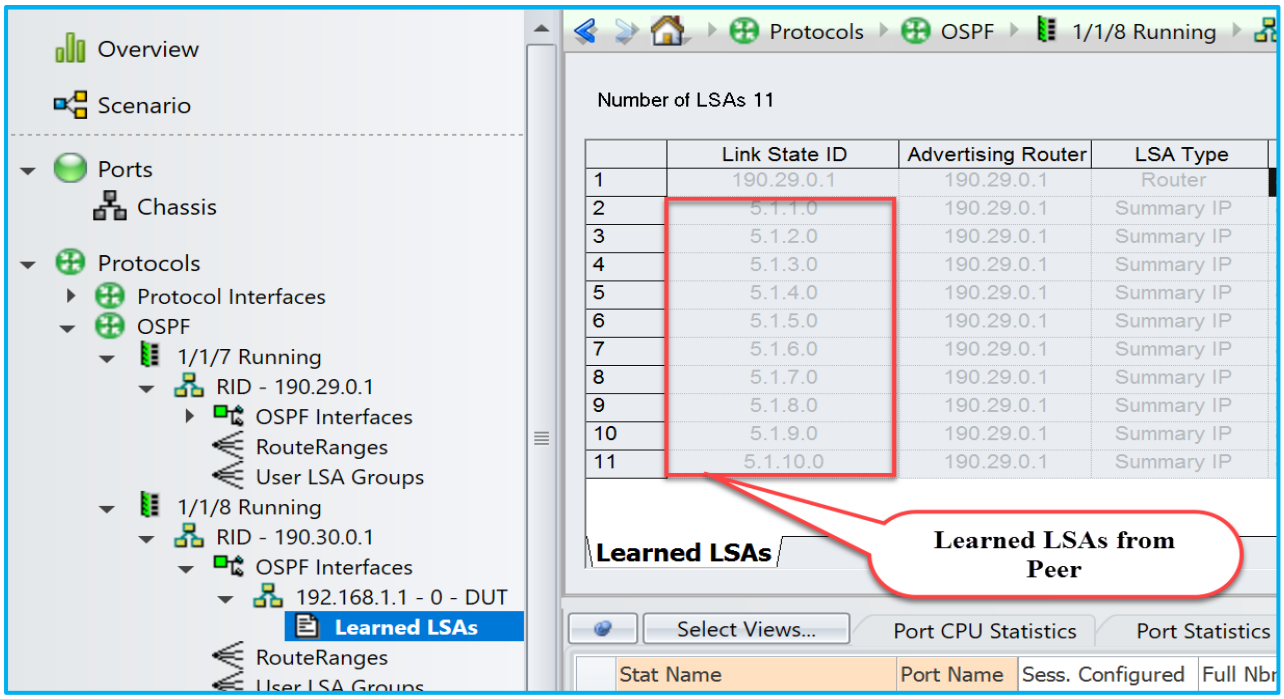


Fig 15.2: View learned LSAs info in GUI

3.8 Configure Traffic

::ixia::traffic_config - Configures traffic streams on the specified ports with the specified options. Here created the traffic flow from 1/1/8 -> 1/1/7.

TCL

```
> set traffic_res [::ixia::traffic_config      \
                -mode create                   \
                -transmit_mode continuous      \
                -track_by {traffic_item}       \
                -rate_pps 1000                 \
                -port_handle $port2            \
                -port_handle2 $port1           \
                -l3_protocol ipv4              \
                -ip_src_addr 192.168.1.1       \
                -ip_dst_addr 192.168.1.2       \
                -mac_dst 00:00:19:d5:54:74     \
                -mac_src 00:00:19:d5:54:75     \
                -l3_length 64]
```

PYTHON

```
> traffic_res = ixia.traffic_config (          \
                mode = 'create',               \
                transmit_mode = 'continuos',   \
                track_by = 'traffic_item',     \
                rate_pps = 1000,               \
                port_handle = port2,           \
                port_handle2 = port1,          \
                l3_protocol = 'ipv4',          \
                ip_src_addr = 192.168.1.1,     \
                ip_dst_addr = 192.168.1.2,     \
                mac_dst = 00:00:19:d5:54:74,   \
                mac_src = 00:00:19:d5:54:75,   \
                l3_length = 64)
```

Fig 16.1: Create traffic stream from 1/1/8 to 1/1/7 port handle using HLAPI

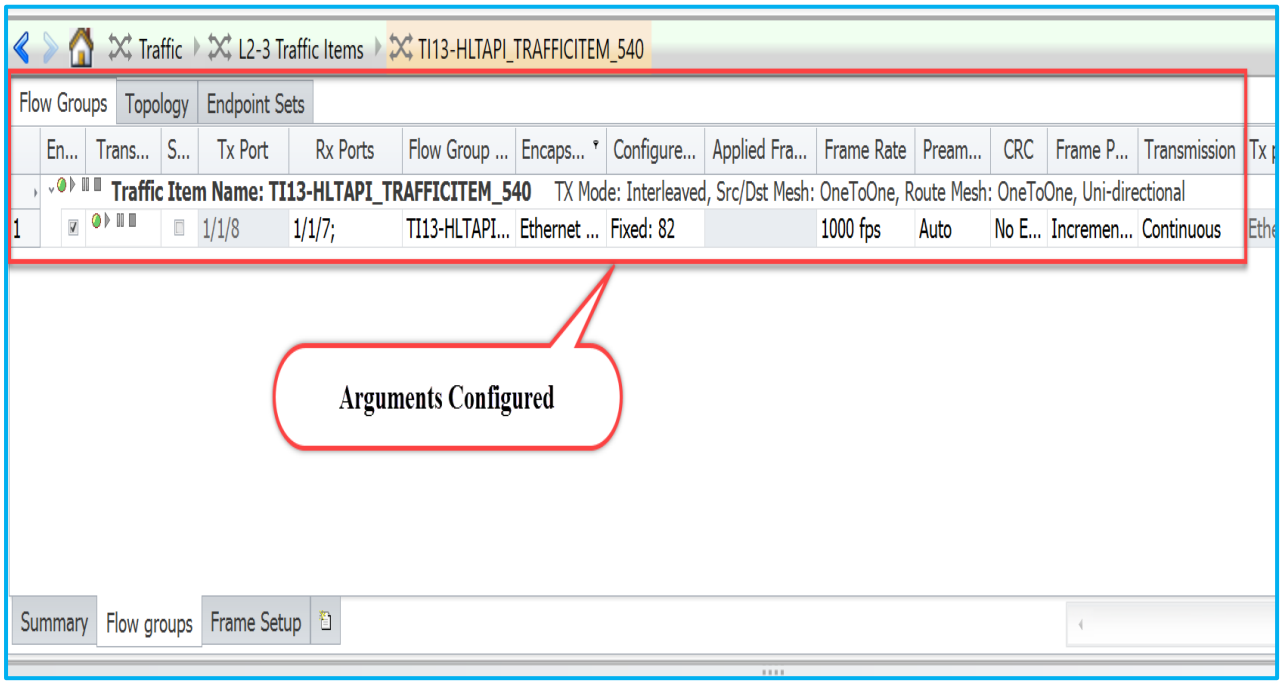


Fig 16.2: View traffic item configuration in GUI

3.9 Start and Stop Traffic

::ixia::traffic_control - Starts or Stops traffic on a given port list. As per below code, start the traffic, wait for some time for the traffic to flow and stop the traffic.

TCL	PYTHON
> *::ixia::traffic_control -action run* > *puts "Wait for some time for the traffic to flow"* > *::ixia::traffic_control -action stop*	> *ixia.traffic_control (action = 'run')* > *print "Wait for some time for the traffic to flow"* > *ixia.traffic_control (action = 'stop')*

Fig 17.1: Start and stop the traffic using HLAPI

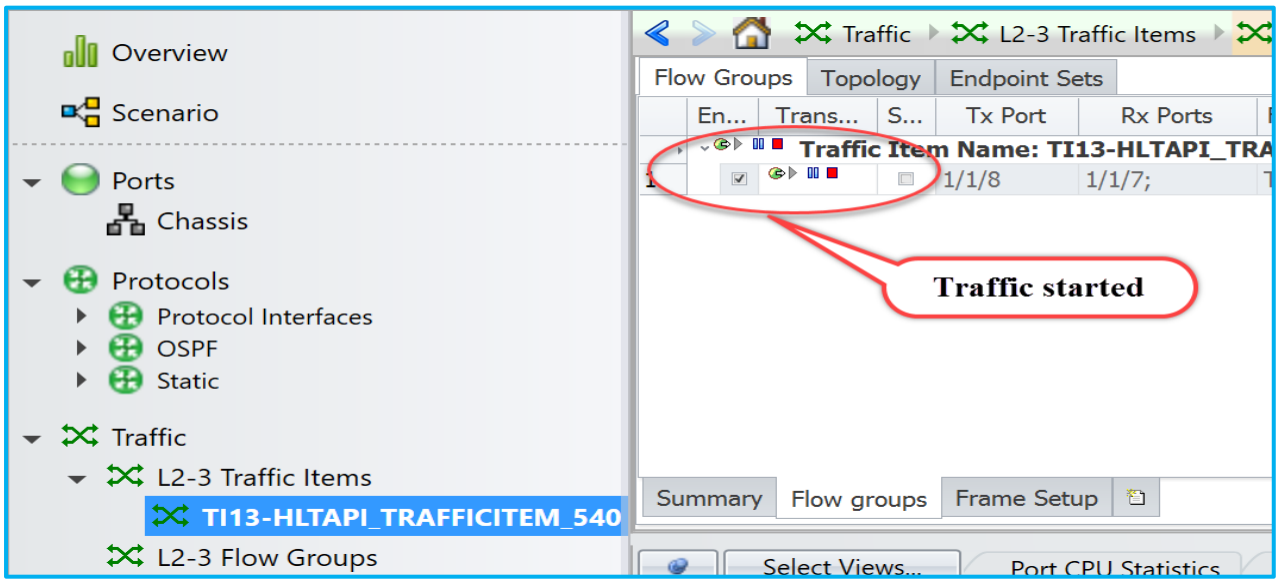


Fig 17.2: View traffic start status in GUI

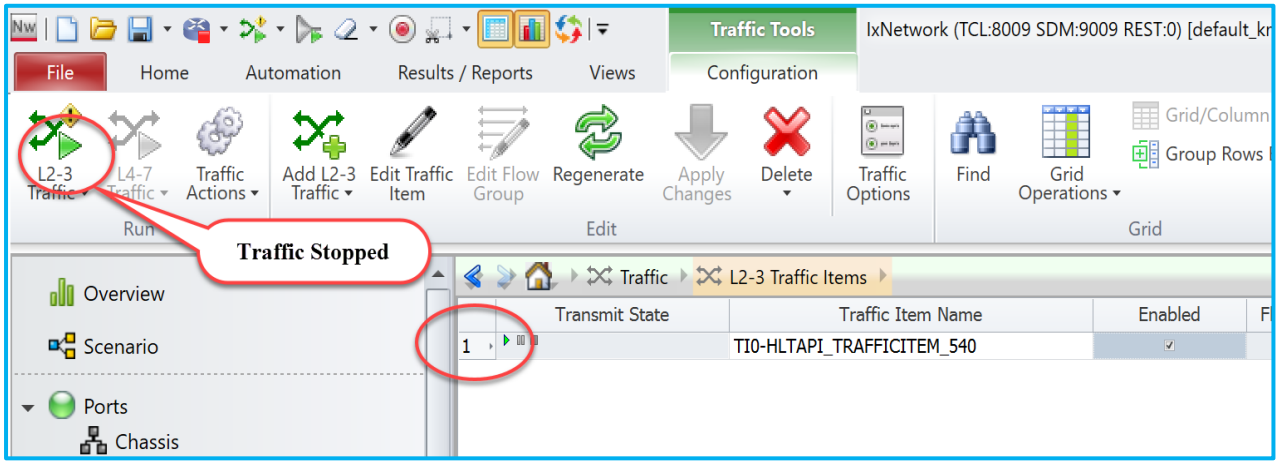


Fig 17.3: View traffic stop status in GUI

3.10 Get Statistics

::ixia::traffic_stats - Gathers statistics depends on the mode suchss as per_port_flows, session, stream, streams, all etc., Below code used mods as 'streams'

TCL

```
> set stats [::ixia::traffic_stats                    \
        -mode streams                                 \
        -streams TI13-HLTAPI_TRAFFICITEM_540]
> puts "Traffic Stats: $stats"
status: 1
measure_mode: mixed
waiting_for_stats: 0
1/1/7:
  stream:
    TI13-HLTAPI_TRAFFICITEM_540:
      rx:
        total_pkt_rate: 0.000
        total_pkt_byte_rate: 0.000
        loss_percent: 0.000
        small_error: N/A
        total_pkts_bytes: 150153856
        expected_pkts: N/A
        pkt_loss_duration: N/A
        last_tstamp: 00:00:16.334
        total_pkts: 368497
        reverse_error: N/A
        ---------- Stripped off long output -----
1/1/8:
  stream:
    TI13-HLTAPI_TRAFFICITEM_540:
      tx:
        total_pkts: 368497
        total_pkt_rate: 0.000
```

PYTHON

```
> stats = ixia.traffic_stats (                        \
    mode = streams,                                   \
    streams = TI13-HLTAPI_TRAFFICITEM_540)
> print "Traffic stats: %s" % stats
status: 1
measure_mode: mixed
waiting_for_stats: 0
1/1/7:
  stream:
    TI13-HLTAPI_TRAFFICITEM_540:
      rx:
        total_pkt_rate: 0.000
        total_pkt_byte_rate: 0.000
        loss_percent: 0.000
        small_error: N/A
        total_pkts_bytes: 150153856
        expected_pkts: N/A
        pkt_loss_duration: N/A
        last_tstamp: 00:00:16.334
        total_pkts: 368497
        reverse_error: N/A
        ---------- Stripped off long output -----
1/1/8:
  stream:
    TI13-HLTAPI_TRAFFICITEM_540:
      tx:
        total_pkts: 368497
        total_pkt_rate: 0.000
```

Fig 18.1: Retrieve traffic stats using HLAPI

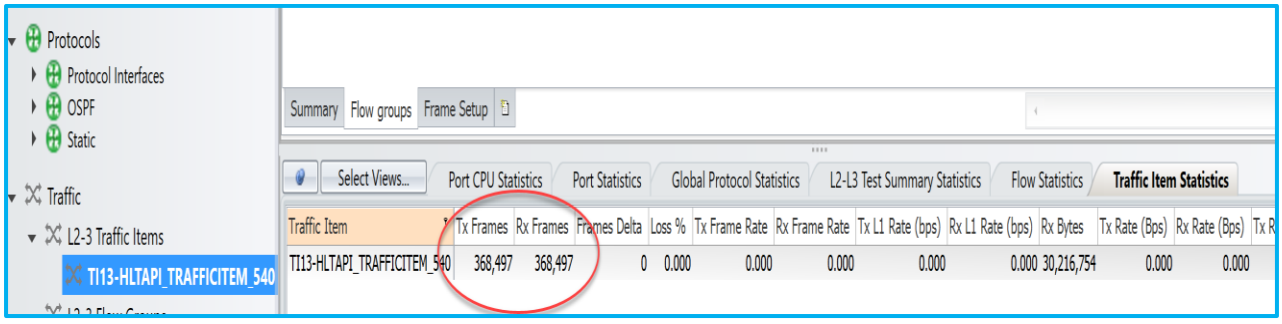


Fig 18.3: Verify Tx Frames and Rx Frames in GUI

4. Other Utilities

4.1 IxNetwork API Documentation Browser

- The main feature of this application is the ability to browse the API meta data in a hierarchical format. Access each level of the hierarchy with a view of siblings, attributes, execs, errors, and children by on clicking on BROWSE.

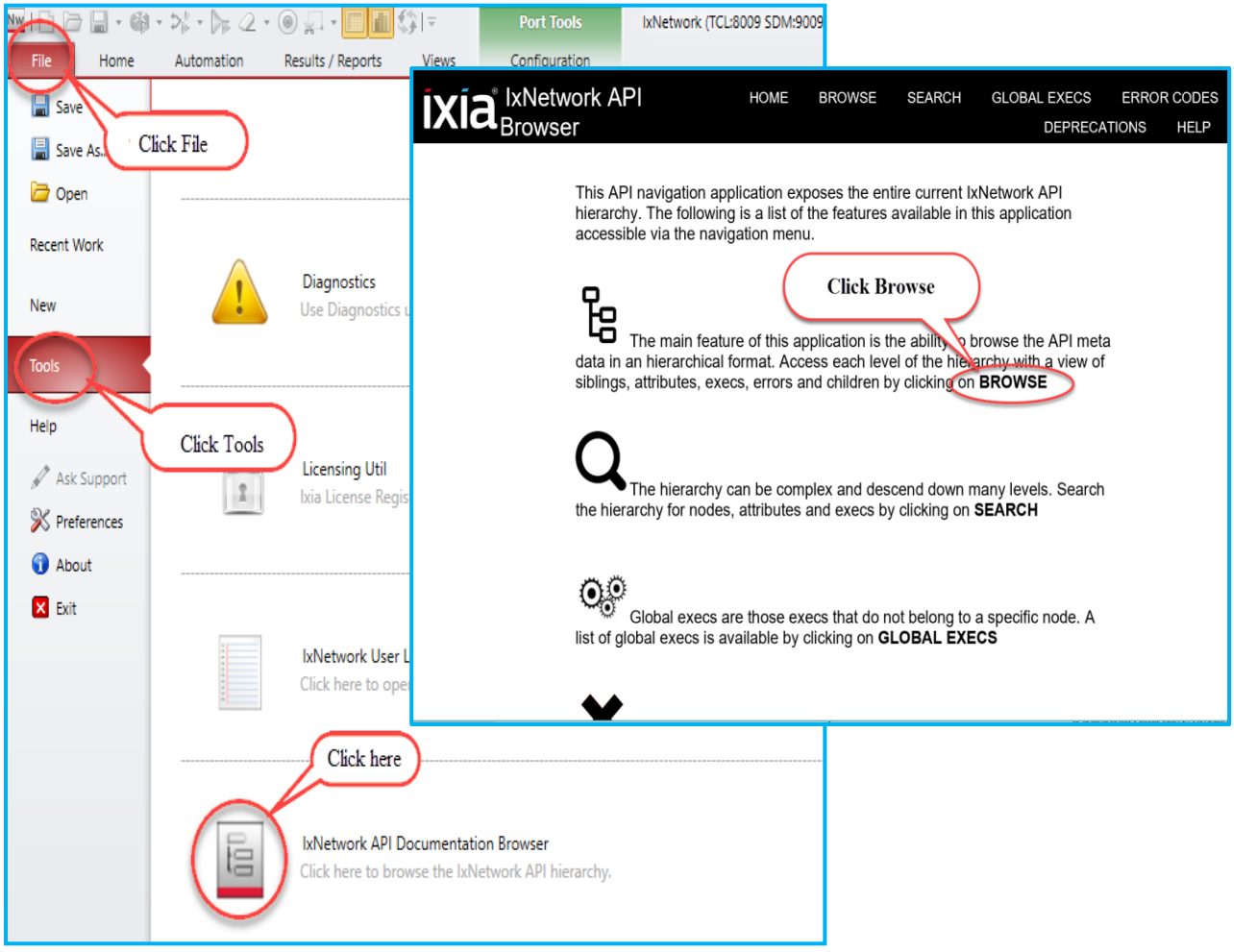


Fig 19.1: IxNetwork API Documentation Browser

4.2 Script Gen

- For complex configuration use SCRIPTGEN. Reverse-engineer the scriptgen scripts as per the requirement.

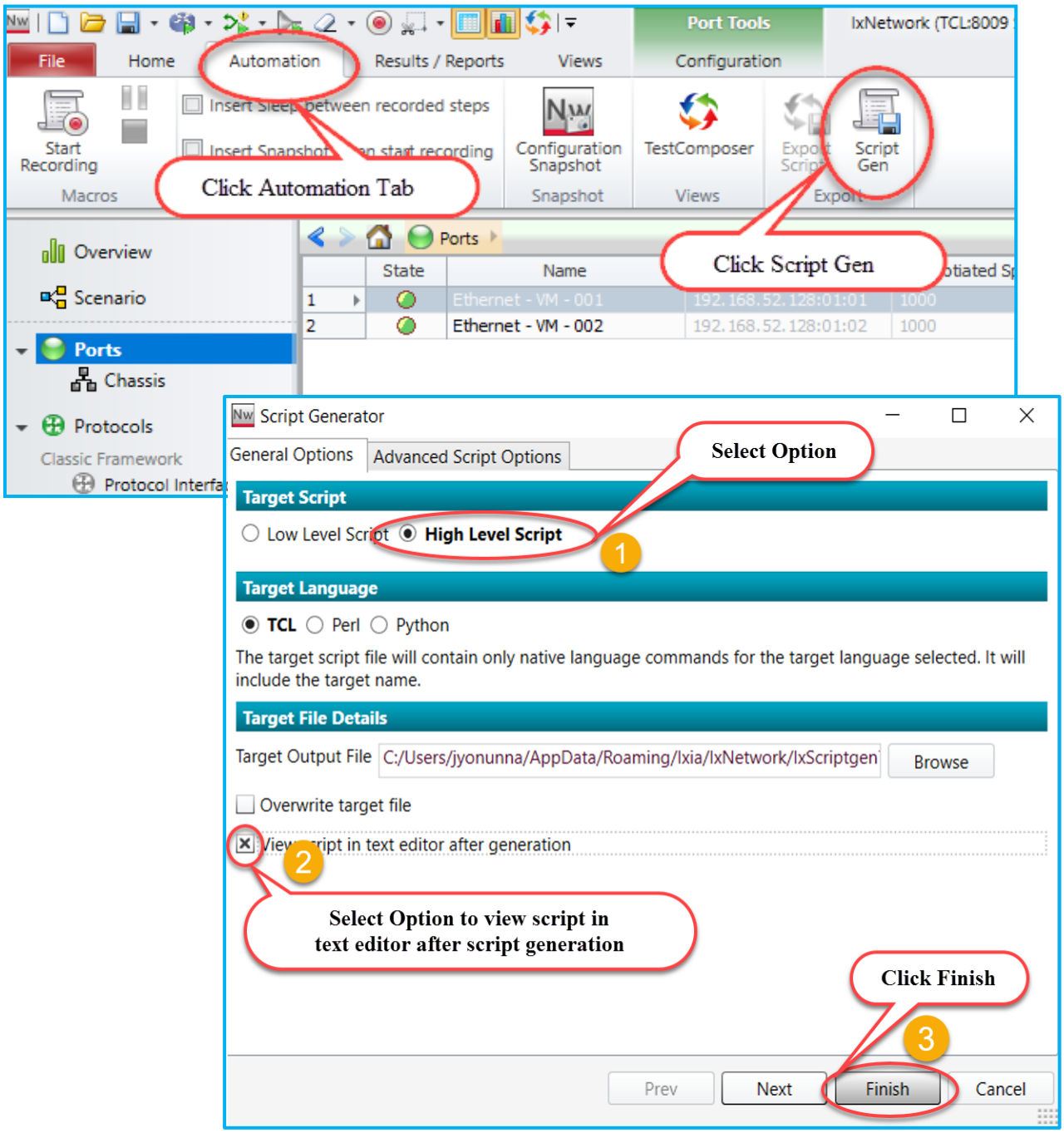


Fig 20.1: Tool to generate script gen

4.3 F1 Option

- Move the mouse pointer over any field in the GUI, and then press F1 to get more information about the field. From Classic Protocols section, users can explore all the fields of all protocols. OSPFv2 protocol tree structure shown below.

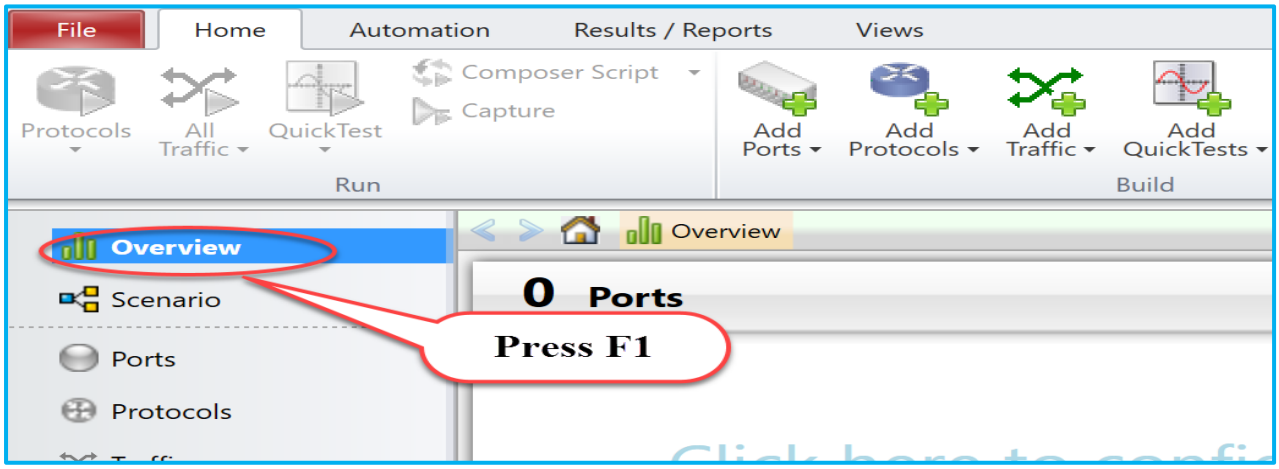


Fig 20.1 Press F1 on Overview field

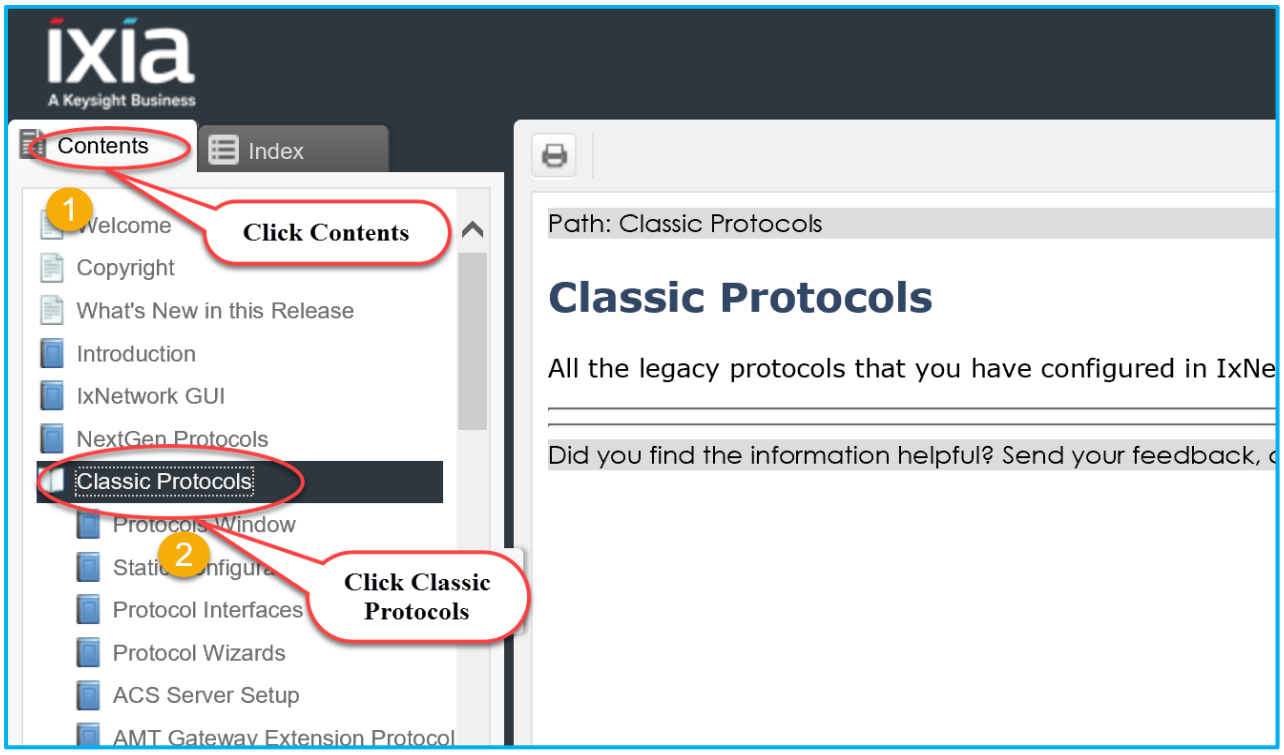


Fig 20.2 Help page to explain about each field in GUI

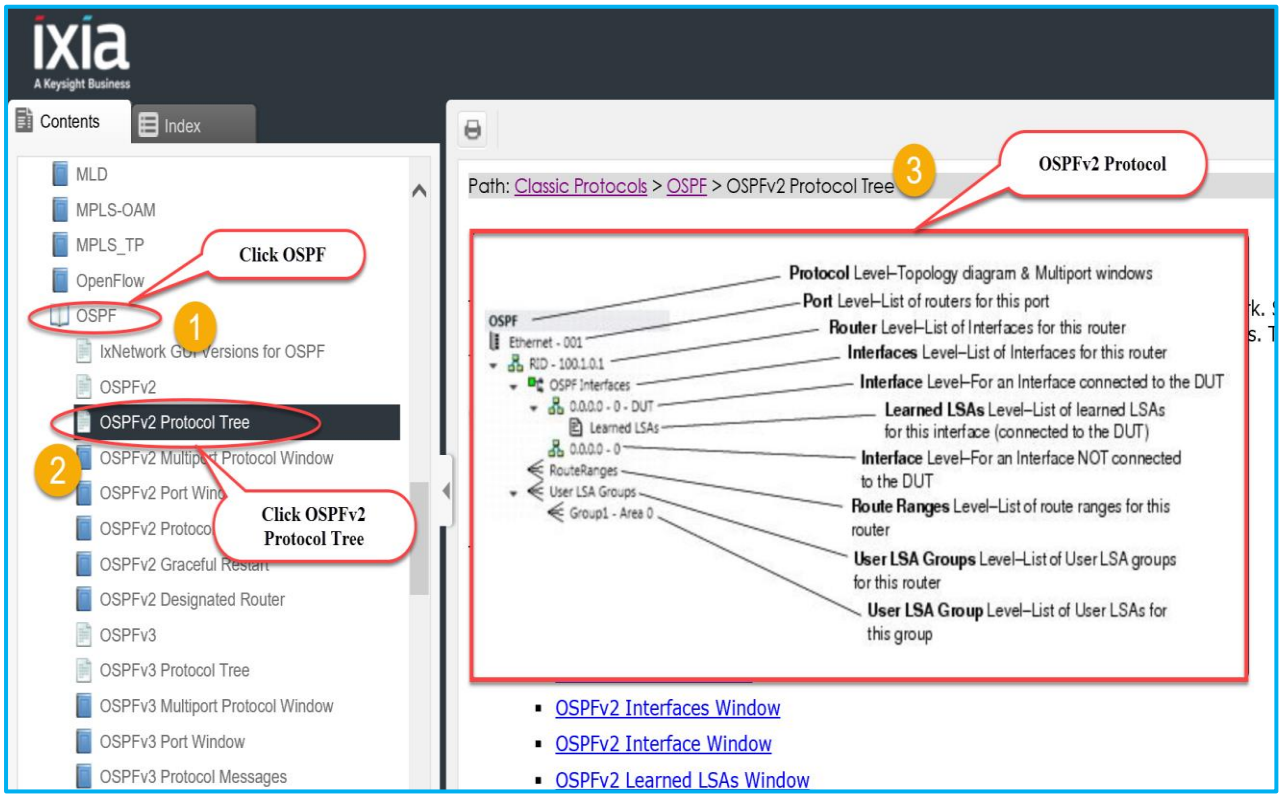


Fig 20.2.1 Shows the contents of OSPFv2 Protocol Tree

5. To Know More on IxNetwork Classic

https://www.youtube.com/watch?v=gWjgFndvSAI
http://openixia.com/sampleScripts//IxNetwork/HighLevelApi/Classic/Tcl

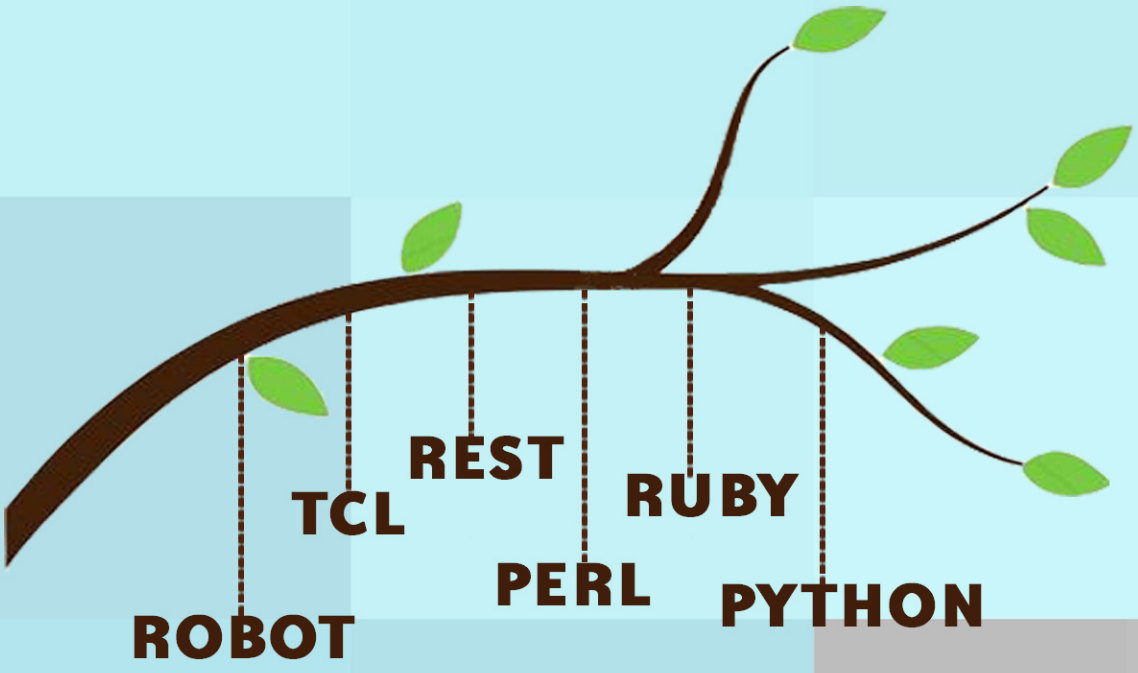
6. Support

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

ixia | A Keysight Business

AUTOMATION

#WeMakeItEasier



https://github.com/openixia

For queries : support.ix@keysight.com