

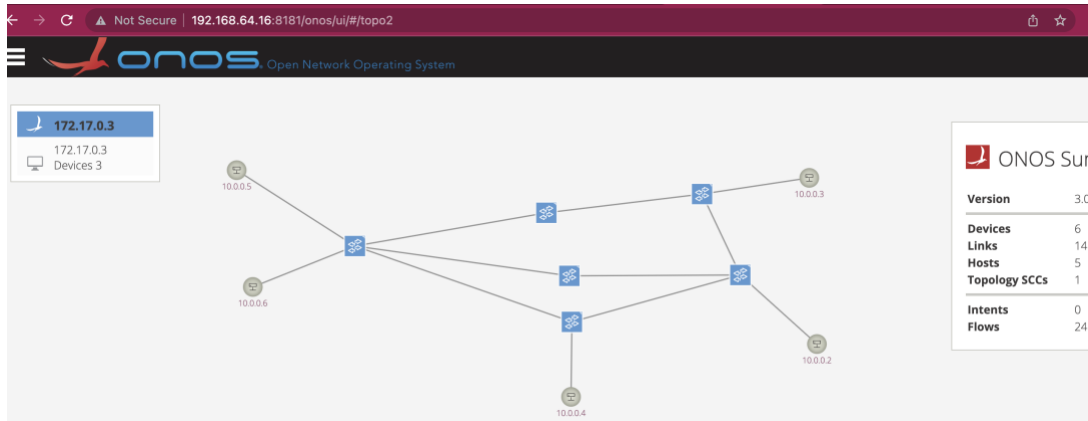
Table of Contents

<i>Demo 7: Using ONOS RESTful API to filter, mirror, and forward networking traffic based on ONOS's Intents framework.....</i>	<i>2</i>
3.2. Pre-Task.....	2
3.3. Tasks.....	3
• Create a series of point-to-point Intents, using Python-based codes and ONOS RESTful API, to allow communication between the "RED" network namespace and the "BLACK" network namespace. 3	
▪ Delete all the created Intents using Python-based code and ONOS RESTful API.....	5
• Create a Host-to-Host Intent to allow communication between the "RED" network namespace and the "BLACK" network namespace.....	5
• Is the Host-to-Host Intent an abstraction of the Point-to-Point Intents? Your answer must be provided with explanations.....	7
• What path is selected by the Host-to-Host Intent for enabling the communication between the "RED" network namespace and the "BLACK" network namespace?	7
• After specifying the path, you are asked to provide a hypothesis on how the Host- to-Host Intent selects paths.....	8
• Using Host-to-Host Intents, enable the communication between all network namespaces in the topology.	8
▪ Without deleting the current Intents, you are asked to create a Single-to-Multi point Intent to allow communication between the "RED" network namespace and "BLACK", "BLUE", "GREEN" network namespaces. Only communication on port 4009 should be allowed. Afer creating this Intent students are requested to verify using "nc" utility or any similar program.....	10
• Explain the benefits brought by the use of Intent-based networking compared to Open-Flow flow rules.	14

Demo 7: Using ONOS RESTful API to filter, mirror, and forward networking traffic based on ONOS's Intents framework

3.2. Pre-Task

A predefined topology created by using provided shell script, and below is the topology-



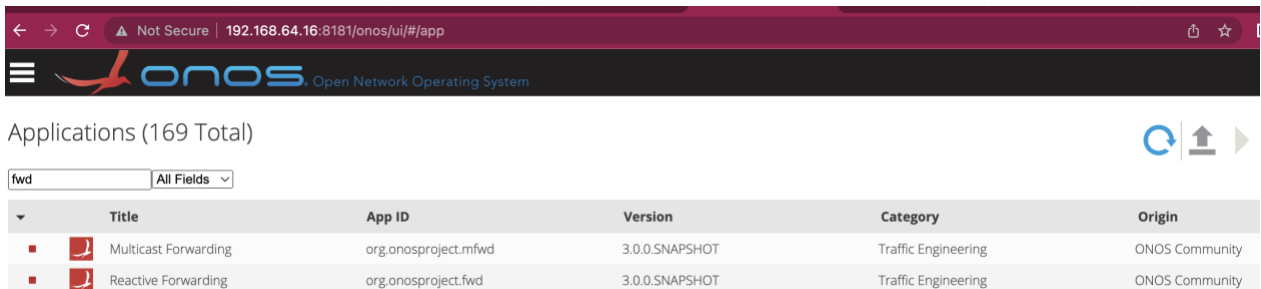
Used the python code created in the previous demonstration to access ONOS RESTful API interface and activated the required ONOS applications following the the list of required applications found in the following file “[start_applications.txt](#)” in demo1.

```
Users > eashin > Documents > sdn > activate-app.py > ...
1 import requests
2 from requests.auth import HTTPBasicAuth
3 import json
4
5 # Set url
6 url = "http://192.168.64.16:8181/onos/v1/applications/"
7
8 # list of apps to activate
9 apps = ["org.onosproject.hostprovider",
10         "org.onosproject.mobility",
11         "org.onosproject.lldpprovider",
12         "org.onosproject.ofagent",
13         "org.onosproject.openflow-base",
14         "org.onosproject.openflow",
15         "org.onosproject.roadm",
16         "org.onosproject.proxyarp",
17         "org.onosproject.fwd"]
18
19 # POST /applications/{app-name}/active
20 for app in apps:
21     myResponse = requests.post(url + app + "/active", auth=HTTPBasicAuth('onos', 'rocks'))
22     print(myResponse)
23     if myResponse.status_code == 200:
24         print("App " + app + " activated")
25
26 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
27 eashin@Eashins-MacBook-Pro ~/Documents/sdn python3 activate-app.py
28 <Response [200]>
29 App org.onosproject.hostprovider activated
30 <Response [200]>
31 App org.onosproject.mobility activated
32 <Response [200]>
33 App org.onosproject.lldpprovider activated
34 <Response [200]>
35 App org.onosproject.ofagent activated
36 <Response [200]>
37 App org.onosproject.openflow-base activated
38 <Response [200]>
39 App org.onosproject.openflow activated
40 <Response [200]>
41 App org.onosproject.roadm activated
42 <Response [200]>
43 App org.onosproject.proxyarp activated
44 <Response [200]>
45 App org.onosproject.fwd activated
46 eashin@Eashins-MacBook-Pro ~/Documents/sdn
```

3.3. Tasks

- Create a series of point-to-point Intents, using Python-based codes and ONOS RESTful API, to allow communication between the "RED" network namespace and the "BLACK" network namespace.

From the ONOS GUI deactivated Reactive Forwarding Application to stop the traffic forwarding to allover, so that our created intents only will do the traffic forwarding. As in below screenshot.



Applications (169 Total)

Search: fwd | All Fields

Title	App ID	Version	Category	Origin
Multicast Forwarding	org.onosproject.mfwd	3.0.0.SNAPSHOT	Traffic Engineering	ONOS Community
Reactive Forwarding	org.onosproject.fwd	3.0.0.SNAPSHOT	Traffic Engineering	ONOS Community

I have tested and verified using `ping` that it stooped to forward traffic from one host to another.

```
ubuntu@dev:~$ sudo ip netns exec red ping -c 1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.

--- 10.0.0.5 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$
```

Below is the python program to create a series of point-to-point intents between RED and BLACK namespace. So, it will create total 6 rules based on intents and 3 of them for RED to BLACK traffic and another 3 of them from BLACK to RED traffic forwarding.

```

Users > eashin > Documents > sdn > Demo_5_6_7_REST_API > demo7 > point-to-point-red-black.py > f
1  import requests
2  from requests.auth import HTTPBasicAuth
3  import json
4
5  url = "http://192.168.64.16:8181/onos/v1/intents"
6  auth = ('onos', 'rocks')
7
8  def add_intent(src_device, src_port, dst_device, dst_port):
9      # Set the payload for the request
10     payload = {
11         "type": "PointToPointIntent",
12         "appId": "org.onosproject.cli",
13         "priority": 111,
14         "selector": {
15             "criteria": [
16                 {
17                     "type": "ETH_TYPE",
18                     "ethType": "0x8000"
19                 }
20             ]
21         },
22         "ingressPoint": {
23             "device": src_device,
24             "port": src_port
25         },
26         "egressPoint": {
27             "device": dst_device,
28             "port": dst_port
29         }
30     }
31     # Send the request to add the intent
32     response = requests.post(url, auth=auth, data=json.dumps(payload))
33     print(response.text)
34
35     # Add the intents red-black : add_intent(src_device, src_port, dst_device, dst_port)
36     add_intent("of:00001ecd0832e94c", "4", "of:00001ecd0832e94c", "1")
37     add_intent("of:0000b6c236375e49", "1", "of:0000b6c236375e49", "2")
38     add_intent("of:0000166648ad8848", "2", "of:0000166648ad8848", "4")
39     # Add the intents black-red : add_intent(src_device, src_port, dst_device, dst_port)
40     add_intent("of:0000166648ad8848", "4", "of:0000166648ad8848", "2")
41     add_intent("of:0000b6c236375e49", "2", "of:0000b6c236375e49", "1")
42     add_intent("of:00001ecd0832e94c", "1", "of:00001ecd0832e94c", "4")

```

It can be verified by using RESTful API that all our python script worked and all intents are in INSTALLED state without any failure, which show in below screenshot.

```

← → ↻ ⚠ Not Secure | 192.168.64.16:8181/onos/v1/intents
{"intents":[{"type":"PointToPointIntent","id":"0x61","key":"0x61","appId":"org.onosproject.cli","resources":[],"state":"INSTALLED"},
{"type":"PointToPointIntent","id":"0x6c","key":"0x6c","appId":"org.onosproject.cli","resources":[],"state":"INSTALLED"},
{"type":"PointToPointIntent","id":"0x5f","key":"0x5f","appId":"org.onosproject.cli","resources":[],"state":"INSTALLED"},
{"type":"PointToPointIntent","id":"0x69","key":"0x69","appId":"org.onosproject.cli","resources":[],"state":"INSTALLED"},
{"type":"PointToPointIntent","id":"0x66","key":"0x66","appId":"org.onosproject.cli","resources":[],"state":"INSTALLED"},
{"type":"PointToPointIntent","id":"0x5d","key":"0x5d","appId":"org.onosproject.cli","resources":[],"state":"INSTALLED"}]}

```

Test and verified using **ping** that traffic is going through as expected between RED and BLACK.

```

ubuntu@dev:~$ sudo ip netns exec red ping -c 1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=1.17 ms

--- 10.0.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.168/1.168/1.168/0.000 ms
ubuntu@dev:~$ sudo ip netns exec black ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.40 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.396/1.396/1.396/0.000 ms
ubuntu@dev:~$

```

- Delete all the created Intents using Python-based code and ONOS RESTful API.

Below is the python script to delete all the existing intents available. The script is first getting all available intents and it loop through all intents and delete one by one.

```
Users > eashin > Documents > sdn > Demo_5_6_7_REST_API > demo7 > delete-intents.py > ...
1  import requests
2  from requests.auth import HTTPBasicAuth
3
4  url = "http://192.168.64.16:8181/onos/v1/intents"
5  auth = HTTPBasicAuth('onos', 'rocks')
6
7  # Get all the intents
8  response = requests.get(url, auth=auth)
9  if response.status_code != 200:
10     print("Error: Failed to get the intents")
11     exit()
12
13  intents = response.json()["intents"]
14
15  # Delete all the intents
16  for intent in intents:
17     app_id = intent["appId"]
18     key = intent["key"]
19     response = requests.delete(url + "/" + app_id + "/" + key, auth=auth)
20     if response.status_code != 204:
21         print("Error: Failed to delete the intent with KEY {}".format(key))
22         exit()
23
24  print("All the intents have been deleted successfully!")
25
```

It is verified by using RESTful API that there are no intents available now after executing the script successfully.



- Create a Host-to-Host Intent to allow communication between the “RED” network namespace and the “BLACK” network namespace.

This time again, before creating host to host intents testing that all traffics are blocked as the Reactive Forwarding Application stopped and all intents are deleted. It is verified by using **ping** as an example below-

```
ubuntu@dev:~$ sudo ip netns exec red ping -c 1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.

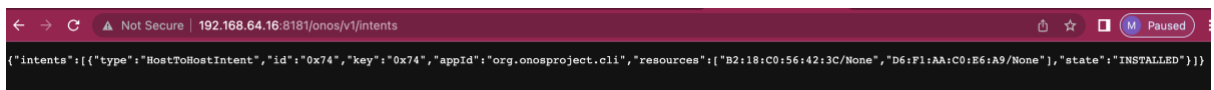
--- 10.0.0.5 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$
```

Below is the python script to create host to host intent. To allow communication between RED and BLACK namespace. The script using hosts Mac Address as the host details, also specified with '/-1' that without any vlan it should use the MAC address to create the intent.

```
Users > eashin > Documents > sdn > Demo_5_6_7_REST_API > demo7 > host-to-host-red-black.py > ..
1  import requests
2  import json
3
4  # Set the ONOS controller details
5  url = "http://192.168.64.16:8181/onos/v1/intents"
6  auth = ('onos', 'rocks')
7
8  # Set the host details
9  host_red = "B2:18:C0:56:42:3C/-1"
10 host_black = "D6:F1:AA:C0:E6:A9/-1"
11
12 # Set the intent parameters
13 payload = {
14     "type": "HostToHostIntent",
15     "appId": "org.onosproject.cli",
16     "priority": 103,
17     "one": host_red,
18     "two": host_black
19 }
20
21 # Send the request to create the intent
22 response = requests.post(url, auth=auth, data=json.dumps(payload))
23 print(response.text)
24
```

It is verified by the REST endpoints from ONOS GUI and it's in INSTALLED state successfully.



```
{
  "intents": [
    {
      "type": "HostToHostIntent",
      "id": "0x74",
      "key": "0x74",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "D6:F1:AA:C0:E6:A9/None"
        ]
      ],
      "state": "INSTALLED"
    }
  ]
}
```

It can be tested using `ping` as below and check that traffic is flowing between RED and BLACK.

```
ubuntu@dev:~$ sudo ip netns exec red ping -c 1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.508 ms

--- 10.0.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.508/0.508/0.508/0.000 ms
ubuntu@dev:~$ sudo ip netns exec black ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.58 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.584/1.584/1.584/0.000 ms
ubuntu@dev:~$
```

It is also possible to check from ONOS GUI that created flow rules based on our intents and that has packets we transmitted by using [ping](#)

Flows for Device of:00001ecd0832e94c (5 Total)

Search All Fields ⌵

STATE ⌵	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	2	147	103	0	IN_PORT:1, ETH_DST:B2:18:C0:56:42:3C, ETH_SRC:D6:F1:AA:C0:E6:A9	imm[OUTPUT:4], cleared:false	*net.intent
Added	2	147	103	0	IN_PORT:4, ETH_DST:D6:F1:AA:C0:E6:A9, ETH_SRC:B2:18:C0:56:42:3C	imm[OUTPUT:1], cleared:false	*net.intent

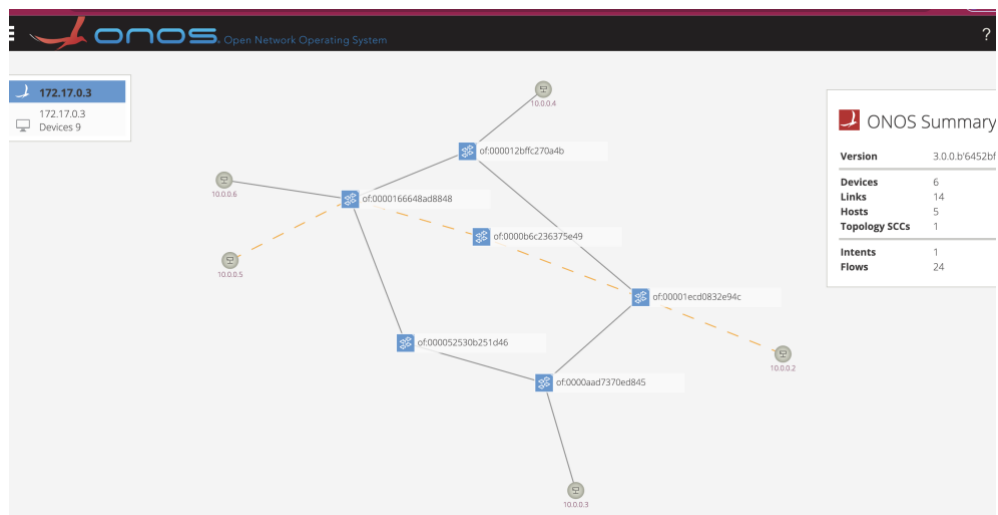
Flows for Device of:0000166648ad8848 (5 Total)

Search All Fields ⌵

STATE ⌵	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	2	193	103	0	IN_PORT:2, ETH_DST:D6:F1:AA:C0:E6:A9, ETH_SRC:B2:18:C0:56:42:3C	imm[OUTPUT:4], cleared:false	*net.intent
Added	2	193	103	0	IN_PORT:4, ETH_DST:B2:18:C0:56:42:3C, ETH_SRC:D6:F1:AA:C0:E6:A9	imm[OUTPUT:2], cleared:false	*net.intent

- [Is the Host-to-Host Intent an abstraction of the Point-to-Point Intents? Your answer must be provided with explanations.](#)
 - Yes, the Host-to-Host Intent can be seen as an abstraction of the Point-to-Point Intents.
 - By using a Point-to-Point Intent, two distinct ports on same or different devices are connected, enabling communication between them. Contrarily, a Host-to-Host Intent is used to establish a connection between two particular hosts, regardless of the ports and connecting hardware.
 - Host-to-Host Intent is regarded as an abstraction of the Point-to-Point Intent because it makes it simpler to establish a connection between two hosts without being aware of the specifics of the network. In essence, a Host-to-Host Intent can be seen as a higher-level idea that removes the specifics of the Point-to-Point Intents, making it simpler to build networks in a more abstract and natural fashion.
- [What path is selected by the Host-to-Host Intent for enabling the communication between the "RED" network namespace and the "BLACK" network namespace?](#)

In below screenshot it shows (yellow marked) the path selected based on the Host-to-Host intent for enabling communication between RED and BLACK namespace.



- After specifying the path, you are asked to provide a hypothesis on how the Host- to-Host Intent selects paths.
 - The ONOS controller will compute the best path between the two hosts based on the configured routing algorithm using its understanding of the network architecture, including switch and link information.
 - Any current flows or intents in the network will likewise influence the Host-to-Host Intent's decision-making process. The Host-to-Host Intent may choose a less-than-optimal path or fail to install if there are competing intentions or flows that prevent the ONOS controller from finding an ideal path.
 - In conclusion, the Host-to-Host Intent will decide which route to take depending on the network architecture and routing algorithms employed by the ONOS controller, as well as the presence of any competing flows or intentions in the network.
- Using Host-to-Host Intents, enable the communication between all network namespaces in the topology.

Here again tested using `ping` that communication between all namespaces and blocked as the Reactive Forwarding Application is stopped on ONOS.

```
ubuntu@dev:~$ sudo ip netns exec black ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$ sudo ip netns exec black ping -c 1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```



```

ubuntu@dev:~$ sudo ip netns exec black ping -c 1 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.

--- 10.0.0.4 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$ sudo ip netns exec black ping -c 1 10.0.0.6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.

--- 10.0.0.6 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

```

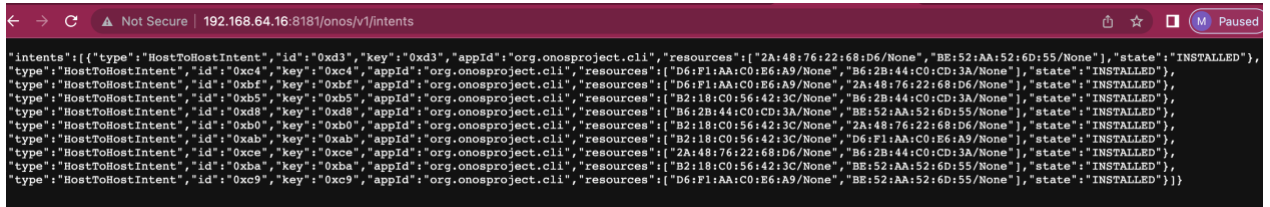
Below is python script that creates intents between all pairs of namespaces in the network topology we have. In this script also used host MAC Addresses and specified that without vlan it should create intents.

```

Users > eashin > Documents > sdn > Demo_5_6_7_REST_API > demo7 > host-to-host-all.py > host_to_host_intent
1  import requests
2  import json
3
4  def host_to_host_intent(one, two):
5      # Set the ONOS controller details
6      url = "http://192.168.64.16:8181/onos/v1/intents"
7      auth = ('onos', 'rocks')
8
9      # Set the intent parameters
10     payload = {
11         "type": "HostToHostIntent",
12         "appId": "org.onosproject.cli",
13         "priority": 104,
14         "one": one,
15         "two": two
16     }
17
18     # Send the request to create the intent
19     response = requests.post(url, auth=auth, data=json.dumps(payload))
20     print(response.text)
21
22 def create_intents(namespaces):
23     # Iterate over all pairs of namespaces
24     for i in range(len(namespaces)):
25         for j in range(i+1, len(namespaces)):
26             ns1 = namespaces[i]
27             ns2 = namespaces[j]
28             # Create the host-to-host intent between the hosts
29             host_to_host_intent(ns1, ns2)
30
31 # Set the host mac addresses details
32 red = "B2:18:C0:56:42:3C/-1"
33 black = "D6:F1:AA:C0:E6:A9/-1"
34 green = "2A:48:76:22:68:D6/-1"
35 blue = "B6:2B:44:C0:CD:3A/-1"
36 yellow = "BE:52:AA:52:6D:55/-1"
37 # Call the function with the list of namespace mac addresses details
38 create_intents([red, black, green, blue, yellow])
39

```

It can verify using ONOS API that all the expected pairs of namespaces having the intents which all together 10 intents and all in INSTALLED state successfully.

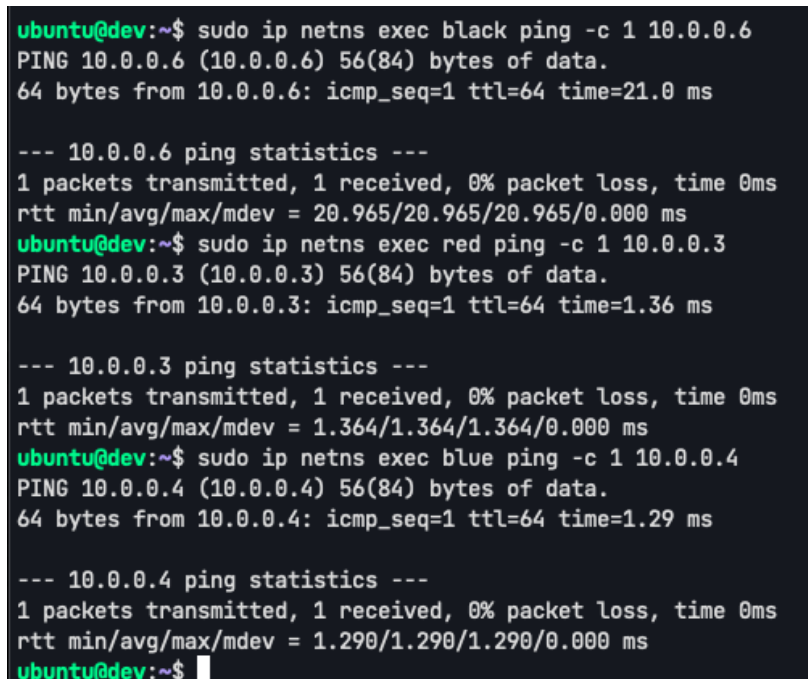


```

{
  "intents": [
    {
      "type": "HostToHostIntent",
      "id": "0xd3",
      "key": "0xd3",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:B6:A9/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xc4",
      "key": "0xc4",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:B6:A9/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xb1",
      "key": "0xb1",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:B6:A9/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xb5",
      "key": "0xb5",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xd8",
      "key": "0xd8",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B6:2B:44:C0:CD:3A/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xb0",
      "key": "0xb0",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xab",
      "key": "0xab",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "D6:F1:AA:C0:B6:A9/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xce",
      "key": "0xce",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:B6:A9/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xba",
      "key": "0xba",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xc9",
      "key": "0xc9",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:B6:A9/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      ],
      "state": "INSTALLED"
    }
  ]
}

```

It tested and verified using [ping](#) that all namespaces can communicate among them which are based on host to host intents we installed.



```

ubuntu@dev:~$ sudo ip netns exec black ping -c 1 10.0.0.6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=21.0 ms

--- 10.0.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 20.965/20.965/20.965/0.000 ms
ubuntu@dev:~$ sudo ip netns exec red ping -c 1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.36 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.364/1.364/1.364/0.000 ms
ubuntu@dev:~$ sudo ip netns exec blue ping -c 1 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=1.29 ms

--- 10.0.0.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.290/1.290/1.290/0.000 ms
ubuntu@dev:~$

```

- Without deleting the current Intents, you are asked to create a Single-to-Multi point Intent to allow communication between the "RED" network namespace and "BLACK", "BLUE", "GREEN" network namespaces. Only communication on port 4009 should be allowed. After creating this Intent students are requested to verify using "nc" utility or any similar program.

Using below script, created Single-to-Multi point intent to allow communication between RED and BLACK, BLUE, GREEN namespaces which only allowed on port 4009. So, the script creates 1 intent which block all traffic in a way that it is a point-to-point intent and sends all traffic to a non-existing port on the device effectively DROP traffics. And the script also create a Single-to-Multi point intents as required.

```

users > eashin > Documents > sdn > Demo_5_6_7_REST_API > demo7 > single-multi-host.py > ...
1
2 import requests
3 import json
4
5 # Set the ONOS controller details
6 url = "http://192.168.64.16:8181/onos/v1/intents"
7 auth = ('onos', 'rocks')
8
9 payload = {
10     "type": "SinglePointToMultiPointIntent",
11     "appId": "org.onosproject.cli",
12     "priority": 110,
13     "selector": {
14         "criteria": [
15             {
16                 "type": "ETH_TYPE",
17                 "ethType": "0x0800"
18             },
19             {
20                 "type": "IP_PROTO",
21                 "protocol": 6,
22             },
23             {
24                 "type": "TCP_DST",
25                 "tcpPort": 4009
26             }
27         ]
28     },
29     "ingressPoint": {
30         "device": "of:00001ecd0832e94c",
31         "port": "4"
32     },
33     "egressPoint": [
34         {
35             "device": "of:0000aad7370ed845",
36             "port": "3"
37         },
38         {
39             "device": "of:000012bffc270a4b",
40             "port": "2"
41         },
42         {
43             "device": "of:0000166648ad8848",
44             "port": "4"
45         }
46     ]
47 }
48 # Send the request to create the single to multi host intent for allowing matching traffic
49 response = requests.post(url, auth=auth, data=json.dumps(payload))
50 print(response.text)
51 # Set the payload for the request to block all traffic
52 block_traffic_payload = {
53     "type": "PointToPointIntent",
54     "appId": "org.onosproject.cli",
55     "priority": 110,
56     "selector": {
57         "criteria": [
58             {
59                 "type": "ETH_TYPE",
60                 "ethType": "0x800"
61             }
62         ]
63     },
64     "ingressPoint": {
65         "device": "of:00001ecd0832e94c",
66         "port": "4"
67     },
68     "egressPoint": {
69         "device": "of:00001ecd0832e94c",
70         "port": "9999" # This is a non-existent port
71     }
72 }
73 # Send the request to add the intent
74 response = requests.post(url, auth=auth, data=json.dumps(block_traffic_payload))
75 print(response.text)
76

```

Similar way as previously it is verified using ONOS API that without deleting previous all host-to-host intents it INSTALLED 2 more intents as our requirement as expected by the execution of script.



```
{
  "intents": [
    {
      "type": "HostToHostIntent",
      "id": "0xd3",
      "key": "0xd3",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "2A:48:76:22:68:D6/None",
          "BE:52:AA:52:6D:55/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xc4",
      "key": "0xc4",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:E6:A9/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xbf",
      "key": "0xbf",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:E6:A9/None",
          "2A:48:76:22:68:D6/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xb5",
      "key": "0xb5",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xd8",
      "key": "0xd8",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B6:2B:44:C0:CD:3A/None",
          "BE:52:AA:52:6D:55/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xb0",
      "key": "0xb0",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "2A:48:76:22:68:D6/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "PointToPointIntent",
      "id": "0x27e",
      "key": "0x27e",
      "appId": "org.onosproject.cli",
      "resources": [
        {}
      ],
      "state": "INSTALLED"
    },
    {
      "type": "SinglePointToMultiPointIntent",
      "id": "0x27d",
      "key": "0x27d",
      "appId": "org.onosproject.cli",
      "resources": [
        {}
      ],
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xab",
      "key": "0xab",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "D6:F1:AA:C0:E6:A9/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xce",
      "key": "0xce",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "2A:48:76:22:68:D6/None",
          "B6:2B:44:C0:CD:3A/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xba",
      "key": "0xba",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "B2:18:C0:56:42:3C/None",
          "BE:52:AA:52:6D:55/None"
        ]
      },
      "state": "INSTALLED"
    },
    {
      "type": "HostToHostIntent",
      "id": "0xc9",
      "key": "0xc9",
      "appId": "org.onosproject.cli",
      "resources": [
        {
          "D6:F1:AA:C0:E6:A9/None",
          "BE:52:AA:52:6D:55/None"
        ]
      },
      "state": "INSTALLED"
    }
  ]
}
```

It is tested using **ping** that traffics are blocked which do not have a destination port 4009.

```
ubuntu@dev:~$ sudo ip netns exec red ping -c 1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$
```

```
ubuntu@dev:~$ sudo ip netns exec red ping -c 1 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.

--- 10.0.0.4 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

It is also tested using **nc** that traffics are blocked which do not have a destination port 4009.

```
ubuntu@dev:~$ sudo ip netns exec black nc -l 4008
[ ]
```

```
ubuntu@dev: ~ (multipass)
```

```
ubuntu@dev:~$ sudo ip netns exec red nc 10.0.0.5 4008
hii
```

On the other hands it tested and verified using `nc` that traffic has destination port 4009 successfully able to communicate each other.

```
ubuntu@dev:~$ sudo ip netns exec black nc -l 4009
hello from red to black
```

```
█
```

```
✕ ubuntu@dev: ~ (multipass)
```

```
ubuntu@dev:~$ sudo ip netns exec red nc 10.0.0.5 4009
hello from red to black
```

```
█
```

```
ubuntu@dev:~$ sudo ip netns exec blue nc -l 4009
hi from red to blue
```

```
█
```

```
✕ ubuntu@dev: ~ (multipass)
```

```
ubuntu@dev:~$ sudo ip netns exec red nc 10.0.0.3 4009
hi from red to blue
```

```
█
```

```
ubuntu@dev:~$ sudo ip netns exec green nc -l 4009
hi from red to green
```

```
█
```

```
✕ ubuntu@dev: ~ (multipass)
```

```
ubuntu@dev:~$ sudo ip netns exec red nc 10.0.0.4 4009
hi from red to green
```

```
█
```

```
ubuntu@dev:~$ sudo ip netns exec green nc -l 3000
```

```
█
```

```
✕ ubuntu@dev: ~ (multipass)
```

```
ubuntu@dev:~$ sudo ip netns exec red nc 10.0.0.4 3000
hello green
```

```
█
```

- Explain the benefits brought by the use of Intent-based networking compared to OpenFlow flow rules.

Over OpenFlow flow rules, intent-based networking (IBN) offers several advantages, some of which are as follows:

- By establishing high-level intents, IBN encapsulates the underlying network infrastructure. On the other hand, OpenFlow flow rules demand a thorough knowledge of the network topology and the capabilities of the network devices.
- IBN systems can check to see if the intended purpose is being carried out correctly in the network and will take corrective action if it is not. These automatic techniques for verification and correction are not present in OpenFlow flow rules.
- IBN offers a centralized view of the network that may be used to coordinate the realization of an intent throughout the whole network. Orchestration is made more challenging by the fact that OpenFlow flow rules are defined and managed on individual network devices.
- Unlike OpenFlow flow rules, which must be manually updated when the network changes, IBN systems may react to network changes in real-time by dynamically modifying intentions.
- IBN offers abstractions at the higher levels, including security limitations, service-level agreements (SLAs), and policies. Complex policies and SLAs that are challenging to achieve with OpenFlow flow rules.