ENDEAVOUR: Towards a flexible software-defined network ecosystem

Project name	ENDEAVOUR
Project ID	H2020-ICT-2014-1 Project No. 644960
Working Package Number	2
Deliverable Number	2.3
Document title	Implementation of the SDN architecture
Document version	0.9
Editor in Chief	Chiesa, UCLO
Authors	Canini, Chiesa, Dietzel, Fernandes
Date	15/12/2016
Reviewer	Dietzel, DE-CIX
Date of Review	13/12/2016
Status	*Public*

Revision History

Date	Version	Description	Author
21/11/16	0.1	First skeleton draft	Canini, Chiesa
23/11/16	0.2	Demonstration of the architecture	Canini, Chiesa
24/11/16	0.3	Description of flows to handle ARP	Fernandes
30/11/16	0.4	Demonstration of APIs	Dietzel
5/12/16	0.5	Executive abstract and corrections	Canini
5/12/16	0.6	Revision	Chiesa
6/12/16	0.7	Description of access control table	Chiesa
13/12/16	0.8	Review	Dietzel, Kopp
13/12/16	0.9	Revision of contents based on internal review	Canini, Chiesa

Executive Summary

This deliverable demonstrates the implementation of the SDN-enabled ENDEAVOUR SDN architecture for IXP fabrics, which is released as open source software. The ENDEAVOUR SDN-enabled IXP architecture is designed to support the use cases defined in Deliverables 4.2 and 4.3 via the specification of arbitrarily fine-grained policies by the IXP participants by means of high-level constructs and APIs. To demonstrate the feasibility of this approach, in this document we focus on illustrating that our implementation is capable of transforming the high-level objectives and policies established by the selected ENDEAVOUR use cases into low-level flow table entries that customize the forwarding behavior of the IXP fabric.

Contents

1 Introduction

In this report, we document the implementation of the ENDEAVOUR SDN architecture. We refer the reader to Deliverable 2.2 for a complete description of the overall architecture. We discuss the organization of the project implementation in terms of code location, contents, and development process in Section 2 and the workflow documentation in Section 3, which demonstrates the ENDEAVOUR SDN architecture prototype.

2 Overview of the Implementation

We first describe the location and development process adopted by the different partners of the ENDEAVOUR project for the prototyping phase of the ENDEAVOUR's SDN architecture demonstrator.

2.1 Git repositories

The ENDEAVOUR code is versioned using the `git` software and hosted on GitHub [1].

To avoid typical pitfalls of team programming, we decided to adhere to the following development guidelines[2]:

- The master branch is **always** deployable.
- Before starting to work on a new feature or fixing a bug, create a descriptively named branch of the master one.
- While working in a branch, keep it updated with the master. This makes it simpler to integrate any future pull request.
- Commit to that branch locally and regularly push the local work to the same named branch on the server.
- When feedback or help is needed or a branch is ready for merging, a pull request should be opened.
- After someone else has reviewed a feature, ask the rest of the team for a green light so that the working branch can be merged with the master one.

[1] `https://github.com/h2020-endeavour`
[2] Based on `http://scottchacon.com/2011/08/31/github-flow.html`

2.2 Development organization

Activities and issues are tracked and visualized with Waffle, a web-based Kanban-like [1] board system that can easily be synchronized with GitHub.

The Waffle board of the ENDEAVOUR project[3] is divided into four columns: `Backlog`, which contains features to be implemented, bugs, and other questions that need to be discussed and are not yet ready to be addressed. `Ready`, which contains those tasks that have been discussed and are ready to be implemented. `In-Progress`, which contains all the tasks that are being implemented at a specific time, and `Done`, which contains those feature that have been implemented and need to be discussed before being archived. Each task or activity is described by a card entity, which contains a brief description of the task, a comment section to append more detailed information for discussions with other team members, a set of labels for classifying the type of a task, and one or more people assigned to that task.

It follows a description of some labels that we added on top of Waffle to better categorize a task:

- **Help Wanted:** To be used whenever some external help is needed to complete a task.
- **Question:** To be used to get information about a relevant topic or aspect of the project.
- **Request for Comments:** To be used to gather feedback from the other collaborators.
- **Bug:** Strongly recommended label to describe an issue that needs to be fixed or, at least, documented for the users.

The ENDEAVOUR team holds a weekly one-hour group call for discussing and updating each other on the status of the current activities with the help of the Waffle board. Each task from all the columns is discussed.

2.3 Code Review

We committed to review code written by other members to the best of our possibilities. The team member that wishes to merge a branch should have its code reviewed by some other team members that have knowledge about the affected part of the platform.

[3] `https://waffle.io/h2020-endeavour/endeavour`

2.4 Tests

Tests are crucial for guaranteeing software reliability and a sound evolution. While unit tests may be implemented according to the developer's sense, behavior tests **must** always be created and performed.

2.5 Virtual Machine Development

We decided to create a Virtual Machine (VM) that can be easily distributed and deployed. Hence, we leverage the Vagrant tool.[4]

3 Demonstration of the SDN architecture

In this section, we demonstrate the functionality of the ENDEAVOUR platform in a tutorial-like style. The reader will be able to repeat all the steps on its own machine. We aim to provide a closer look at the ENDEAVOUR forwarding-level by showing how ENDEAVOUR supports fine-grained routing policies, multi-hop forwarding (with Umbrella), and load-balancing of traffic. We refer the reader to Deliverable 2.2 for a complete description of the overall architecture. Due to the low-level, technical nature of this demonstration, we focus on a tutorial-style workflow instead of providing a video demonstration. The videos within Deliverables 4.4 and 4.5 complement this workflow by presenting a high-level perspective with concrete usages of the platform.

3.1 Getting started

Please refer to the up-to-date setup instructions described on the GitHub repository to provision a VM with the ENDEAVOUR platform and install the necessary software for running the ENDEAVOUR platform. Let `$HOME` be the root folder the ENDEAVOUR repository will be located.

```
$ cd $HOME
$ git clone git@github.com:h2020-endeavour/endeavour.git
```

Note that the remainder of this section is based on the following Git commit ID: ef34b6b. This commit can be retrieved by issuing the following command from the `endeavour` folder:

```
$ git checkout ef34b6b
```

[4] https://www.vagrantup.com/

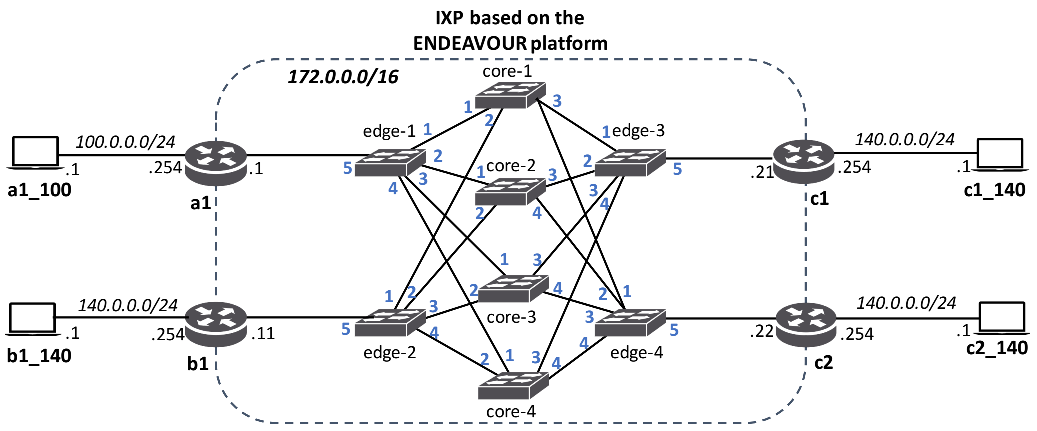

Figure 1: Example IXP network.

Vagrant is used to provision a readily-usable VM with the ENDEAVOUR platform as follows:

```
$ cd endeavour && vagrant up
```

3.2 Emulation environment

The topology will be created with the MiniNet[5] emulation tool, which is installed by the above VM provisioning operation. Each router in MiniNet runs the zebra and bgpd daemons, part of the Quagga routing engine.[6]

3.3 Network topology and BGP/iSDX configuration

We use the network topology depicted in Figure 1 to demonstrate the ENDEAVOUR platform. Three IXP members `A`, `B`, and `C` connect to an ENDEAVOUR IXP fabric, which consists of a Core-Edge topology with four core switches and four edge switches. Member `A` owns a BGP border router `a1` connected to `edge1`, member `B` owns a BGP border router `b1` connected to `edge2`, and Member `C` owns two BGP border routers `c1` and `c2` connected to `edge3` and `edge4`, respectively. A host `a1_100` with IP address `100.0.0.1` is connected to member `A`'s router and three hosts `b1_140`, `c1_140`, and `c2_140`

[5]http://mininet.org/
[6]http://www.nongnu.org/quagga/

with the same IP address `140.0.0.1` are connected to routers `b1`, `c1`, and `c2`, respectively. The IXP IP subnet is `172.0.0.0/16` and the exact address of each member's border router is depicted in the figure close to the member's router. Member `A` announces `100.0.0.0/24` while members `B` and `C` both announce `140.0.0.0/24`. The port numbers used within the IXP fabric are highlighted in blue.

The MAC addresses of the members' border routers assigned to the interfaces connected to the IXP fabric are:

border-router	**MAC-address**
a1	08:00:bb:bb:01:00
b1	08:00:bb:bb:02:00
c1	08:00:bb:bb:03:00
c2	08:00:bb:bb:03:01

The virtual MAC addresses are used by the ENDEAVOUR platform to transmit BGP information to the hosts. Its bits are used to mark whether a member announced or not an IP subnet. The most-significant bit is used to notify that a packet has to be processed by the iSDX pipeline. The mapping of bits to members (in this example) is as follows:

member	**most-significat-bit**
A	2
B	4
C	3

The iSDX pipeline also uses additional identifiers for each member entity:

member	**iSDX-MAC-identifier**
A	**:**:**:**:00:01
B	**:**:**:**:00:02
C	**:**:**:**:00:03

and for each members' border routers:

border-router	**iSDX-MAC-identifier**
a1	**:**:**:**:00:01
b1	**:**:**:**:00:02
c1	**:**:**:00:00:03
c2	**:**:**:01:00:03

3.4 SDN fine-grained policy specification

Member `A`'s outbound routing policies:

```
match(dstport=80) >> fwd(B)
match(dstport=4321) >> fwd(C)
match(dstport=4322) >> fwd(C)
```

HTTP traffic (port 80) is forwarded by member `A` towards member `B` while the traffic sent by member `A` destined to ports `4321` and `4322` is forwarded through member `C`. Member `C`'s inbound routing policies:

```
match(dstport=4321) >> fwd(C1)
match(dstport=4322) >> fwd(C2)
```

Member `C` configure its routing policies so as to steer traffic directed to port `4321` and `4322` to its routers `C1` and `C2`, respectively.

3.5 Configuring the setup

Log into the VM machine from the `endeavour` folder and launch the ENDEAVOUR platform using the following commands:

```
$ vagrant ssh
$ cd iSDX/test
$ bash buildall.sh
$ sudo bash startup.sh -i test1-mh-architecture
```

That's it! Your ENDEAVOUR platform is now ready to handle the fine-grained routing policies specified by your IXP members.

3.6 Testing the setup

We check the status of the network and send some flows of traffic that match the policies specified in the beginning of this tutorial.

We first check if the route server has correctly advertised the routes by dumping the IP routing table of member `A`'s router:

```
> exec a1 ip route
100.0.0.0/24 dev a1-eth1  proto kernel  scope link  src 100.0.0.254
140.0.0.0/24 via 172.0.1.1 dev a1-eth0  proto zebra
172.0.0.0/16 dev a1-eth0  proto kernel  scope link  src 172.0.0.1
```

We can see that `A` is directly connected to both `100.0.0.0/24` and `172.0.0.0/16` and it received `140.0.0.0/24` via the `zebra` routing daemon.

We can verify that the border routers are connected with each other by executing the following full-mesh of `ping` commands:

```
> exec a1 ping 172.0.0.11 -c 1
> exec a1 ping 172.0.0.21 -c 1
> exec a1 ping 172.0.0.22 -c 1
> exec b1 ping 172.0.0.1 -c 1
> exec b1 ping 172.0.0.21 -c 1
> exec b1 ping 172.0.0.22 -c 1
> exec c1 ping 172.0.0.1 -c 1
> exec c1 ping 172.0.0.11 -c 1
> exec c1 ping 172.0.0.22 -c 1
> exec c2 ping 172.0.0.1 -c 1
> exec c2 ping 172.0.0.11 -c 1
> exec c2 ping 172.0.0.21 -c 1
```

3.7 Testing the SDN policies

We now want to check that the members' routing policies are correctly implemented within the fabric. As such, we want to start sending from a host inside `A`'s network several flows towards IP `140.0.0.1`. `A`'s outbound specifies that HTTP traffic should be sent to `B` and traffic towards ports `4321` and `4322` will be routed to `C`. Moreover, `C`'s inbound policy states that traffic towards Transport Control Protocol (TCP) port `4321` must enter its network at router `C1`, and traffic towards `4322` must enter its network at router `C2`. We will use `iperf`[7] to verify that ENDEAVOUR is implementing these policies correctly.

Verify HTTP traffic by sending a 10-second flow towards `b1_140` with these commands:

```
> exec b1_140 iperf -s -B 140.0.0.1 -p 80 &IPERF_SERVER1
> exec a1_100 iperf -c 140.0.0.1 -B 100.0.0.1 -p 80 -t 10
```

Successful `iperf` connections should look like this:

```
> exec b1_140 iperf -s -B 140.0.0.1 -p 80 &IPERF_SERVER1
MM:b1_140 REXEC: iperf -s -B 140.0.0.1 -p 80 &IPERF_SERVER1
MM:b1_140 REXEC: output =
Process (24283) started
> exec a1_100 iperf -c 140.0.0.1 -B 100.0.0.1 -p 80 -t 10
MM:a1_100 REXEC: iperf -c 140.0.0.1 -B 100.0.0.1 -p 80 -t 10
MM:a1_100 REXEC: output =
```

[7] `https://iperf.fr/`

```
------------------------------------------------------------
Client connecting to 140.0.0.1, TCP port 80
Binding to local address 100.0.0.1
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 100.0.0.1 port 49594 connected with 140.0.0.1 port 80
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  3.83 GBytes  3.29 Gbits/sec
```

Kill the server with `> killp b1_140 IPERF_SERVER1`.

Verify traffic destined to port `4321` by sending a 10-second flow towards `c1_140` with these commands:

```
> exec c1_140 iperf -s -B 140.0.0.1 -p 4321 &IPERF_SERVER2
> exec a1_100 iperf -c 140.0.0.1 -B 100.0.0.1 -p 4321 -t 10
```

Kill the server with `> killp c1_140 IPERF_SERVER2`.

Verify traffic destined to port `4322` by sending a 10-second flow towards `c1_140` with these commands:

```
> exec c2_140 iperf -s -B 140.0.0.1 -p 4322 &IPERF_SERVER3
> exec a1_100 iperf -c 140.0.0.1 -B 100.0.0.1 -p 4322 -t 10
```

Kill the server with `> killp c2_140 IPERF_SERVER3`.

The `iperf` has always the same destination IP address, but reaches different hosts (b1_140, c1_140, c2_140) due to the more specific SDN policies.

3.8 Check the ARP tables

We now check the state of `a1` ARP table:

```
> exec a1 arp
Address         HWtype  HWaddress           Flags Mask    Iface
172.0.255.254   ether   08:00:27:89:3b:ff   C             a1—eth0
172.0.1.1       ether   b0:00:00:00:00:02   C             a1-eth0
100.0.0.1       ether   0e:34:4e:d0:db:9a   C             a1—eth1
```

Recall from Section 3.6 that `a1` received a route towards `140.0.0.0/24` via `172.0.0.1`, which is mapped to `b0:00:00:00:00:02`. This MAC address is a virtual MAC address whose first octet contains the BGP reachability information of `140.0.0.0/24`. Namely, `b0=(1011 0000)`, which means that both members `B` and `C` are announcing a route towards that prefix since the 3'rd and 4'th most-significant bits are set to 1.

Observe that if we withdraw `140.0.0.0/24` from member `B` using the following command:

```
> withdraw b1 140.0.0.0/24
```

We check again the ARP table of `a1`:

```
> exec a1 arp
Address          HWtype  HWaddress           Flags Mask    Iface
172.0.255.254    ether   08:00:27:89:3b:ff   C             a1—eth0
172.0.1.1        ether   a0:00:00:00:00:03   C             a1-eth0
100.0.0.1        ether   0e:34:4e:d0:db:9a   C             a1—eth1
```

We can see that the most-significant octet is now `a0=(1010 0000)`, which means that only member `C` is announcing a route towards `140.0.0.0/24`.

We announce again the `140.0.0.0/24` IP subnet from `b1` using the following command:

```
> announce b1 140.0.0.0/24
```

3.9 Check the content of the OpenFlow tables

3.9.1 Dump of `edge-1` switch table

We dump the forwarding table of `edge-1` using the following command:

```
> local ovs-ofctl -O OpenFlow13 dump-flows edge-1
```

We observe that there are eight different tables:

Table 0: **Monitoring**: This table contains the monitoring rules that are used to count the amount of traffic for some specific flows. We refer the reader to Deliverable 3.3 for details regarding the monitoring capabilities.

Table 1: **Main-In**: This table verifies if a packet should be processed by the iSDX pipeline (moving it to the next table) and it handles ARP traffic for the iSDX component. The packet is otherwise forwarded to Table 5.

Table 2: **Outbound iSDX**. This table matches iSDX packets to the configured outbound policies.

Table 3: **Inbound iSDX**. This table matches iSDX packets to the configured inbound policies and it applies the blackholing policies.

Table 4: **Main-Out**. This table translates iSDX encodings into real MAC addresses.

Table 5: **Access Control**. This table contains the access control rules that are used to control what traffic is allowed to traverse the IXP fabric. We refer the reader to Deliverable 4.2 and Deliverable 4.4 for details regarding the need for this capability and its demonstrator.

Table 6: **Load-Balancing**: This table assigns a core switch to the forwarded traffic flows so as to optimize network performance.

Table 7: **Umbrella**. This table modifies the destination MAC address of a packet so as to forwarded a packet to the IXP egress port traversing the previously computed core switch (if any). It leverages the Umbrella encoding. It also handles ARP traffic within the fabric.

Next, we analyze the content of Table 0 (i.e., the Monitoring Table). Rules with a greater number have a higher priority. For the sake of simplicity, we do not discuss rules used to handle traffic towards the controller.

```
1: cookie=0x3, duration=1568.845s, table=0, n_packets=0,
   n_bytes=0, priority=11111,tcp,tp_dst=4322 actions=goto_table:1
2: cookie=0x2, duration=1568.845s, table=0, n_packets=1642101,
   n_bytes=20480637514, priority=11111,tcp,tp_dst=4321
   actions=goto_table:1
3: cookie=0x1, duration=1568.851s, table=0, n_packets=554880,
   n_bytes=4146908960, priority=11111,tcp,tp_dst=80
   actions=goto_table:1
4: cookie=0x0, duration=1569.359s, table=0, n_packets=2171082,
   n_bytes=143368849, priority=0 actions=goto_table:1
```

Rules 1,2, and 3 are used to count the amount of packets that match on port 4322, 4321, 80, respectively. Row 4 matches any other traffic. The action is always to go to the next table, i.e., Table 1.

We now analyze the content of Table 1 (i.e., the Main-In Table):

```
 1: cookie=0x34, duration=1569.021s, table=1, n_packets=2166791,
    n_bytes=143048867, priority=1 actions=goto_table:4
 2: cookie=0x0, duration=1569.359s, table=1, n_packets=2,
    n_bytes=180, priority=0 actions=CONTROLLER:65535
 3: cookie=0x33, duration=1569.021s, table=1, n_packets=0,
    n_bytes=0, priority=2,dl_dst=80:00:00:00:00:00/80:00:00:00:00:00
    actions=goto_table:2
 4: cookie=0x1, duration=1569.022s, table=1, n_packets=2196985,
    n_bytes=24627546762, priority=5,in_port=5
    actions=set_field:08:00:bb:bb:01:00->eth_src,goto_table:2
 5: cookie=0x32, duration=1569.021s, table=1, n_packets=2, n_bytes=84,
    priority=9,arp,dl_dst=00:00:00:01:00:03/00:00:03:ff:ff:ff
    actions=set_field:08:00:bb:bb:03:01->eth_dst,goto_table:5
 6: cookie=0x2f, duration=1569.022s, table=1, n_packets=3, n_bytes=126,
    priority=9,arp,dl_dst=00:00:00:00:00:01/00:00:03:ff:ff:ff
    actions=set_field:08:00:bb:bb:01:00->eth_dst,goto_table:5
 7: cookie=0x31, duration=1569.021s, table=1, n_packets=2, n_bytes=84,
    priority=9,arp,dl_dst=00:00:00:00:00:03/00:00:03:ff:ff:ff
    actions=set_field:08:00:bb:bb:03:00->eth_dst,goto_table:5
 8: cookie=0x30, duration=1569.022s, table=1, n_packets=6, n_bytes=252,
    priority=9,arp,dl_dst=00:00:00:00:00:02/00:00:03:ff:ff:ff
    actions=set_field:08:00:bb:bb:02:00->eth_dst,goto_table:5
 9: cookie=0x0, duration=1569.359s, table=1, n_packets=4146,
    n_bytes=313676, priority=8, tcp,tp_src=179 actions=goto_table:5
10: cookie=0x35, duration=1569.020s, table=1, n_packets=4, n_bytes=168,
    priority=8,arp,arp_tpa=172.0.1.0/24
    actions=set_field:08:00:27:89:33:ff->eth_dst,goto_table:5
11: cookie=0x0, duration=1569.359s, table=1, n_packets=122, n_bytes=5124,
    priority=7,arp actions=goto_table:5
```

Rule 3 is used to match packets that has to be processed by the iSDX pipeline for whereas the sending member has not configured any iSDX outbound policy. These packets can be easily detected by having their most-significant bit set to 1.

Rule 4 enforces the source MAC address of a packet to be a unique MAC address for all the traffic generate by a member that configured an iSDX oubound policy. It helps to reduce the number of flows in case of a member with more than one port at the IXP.

Rules 5 to 8 rewrite the virtual destination MAC of ARP replies issued by the ARP proxy. To avoid broadcast messages the destination MAC address of the packet is set to be the real MAC address. It then moves it to the Umbrella table.

Rule 10 is meant to forward ARP requests for a Virtual Next Hop directly to the ARP Proxy. It sets the packet destination MAC address to be the ARP Proxy one and moves the packet to the Umbrella table. Rule 11 is also used to handle ARP traffic, however, rule 11 is intended to carefully enable the correct exchange of requests and replies between the route server and the members during the establishment of BGP sessions.

We now analyze the content of Table 2 (i.e., the Outbound Table):

```
1: cookie=0x10001, duration=1560.484s, table=2, n_packets=554880,
   n_bytes=4146908960, priority=2, tcp,dl_src=08:00:bb:bb:01:00,
   dl_dst=10:00:00:00:00:00/50:00:00:00:00:00, tp_dst=80
   actions=set_field:80:00:00:00:00:02->eth_dst, goto_table:3
2: cookie=0x38, duration=1569.019s, table=2, n_packets=4, n_bytes=288,
   priority=1 actions=goto_table:3
3: cookie=0x0, duration=1569.359s, table=2, n_packets=0, n_bytes=0,
   priority=0 actions=CONTROLLER:65535
4: cookie=0x10003, duration=1560.539s, table=2, n_packets=0, n_bytes=0,
   priority=2,tcp,dl_src=08:00:bb:bb:01:00,
   dl_dst=20:00:00:00:00:00/60:00:00:00:00:00,tp_dst=4322
   actions=set_field:80:00:00:00:00:03->eth_dst,goto_table:3
5: cookie=0x10002, duration=1560.540s, table=2, n_packets=1642101,
   n_bytes=20480637514, priority=2, tcp,dl_src=08:00:bb:bb:01:00,
   dl_dst=20:00:00:00:00:00/60:00:00:00:00:00, tp_dst=4321
   actions=set_field:80:00:00:00:00:03->eth_dst, goto_table:3
```

Rule 1 applies member `A`'s outbound policies by matching the source MAC address of `a1`' (i.e., `08:00:bb:bb:01:00`), verifying that B is announcing a route towards `140.0.0.0/24` by matching the 4'th most-significant bit of the destination MAC address (i.e. `dl_dst=10:00:00:00:00:00/50:00:00:00:00:00`), the TCP port to be 80. If a packet is matched by this rule, its destination MAC is rewritten with `B`'s iSDX-identifier (i.e., `80:00:00:00:00:02`) and the packet is moved to the Inbound Table.

Rule $4-5$ are used to propagate packets destined to ports `4321` and `4322` to the Inbound Table only if member `C`'s announced a route towards `140.0.0.0/24`, where member `C`'s BGP reachability information is encoded in the 3'rd most-significant bit of the destination MAC address (i.e. `dl_dst=20:00:00:00:00:00/60:00:00:00:00:00`). If a packet is matched by these rules, its destination MAC is rewritten with `C`'s iSDX-identifier (i.e., `80:00:00:00:00:03`) and the packet is moved to the Inbound Table.

We now analyze the content of Table 3 (i.e., the Inbound Table):

```
1: cookie=0x37, duration=1569.019s, table=3, n_packets=554884,
   n_bytes=4146909248, priority=1 actions=goto_table:4
```

```
2: cookie=0x0, duration=1569.359s, table=3, n_packets=0, n_bytes=0,
   priority=0 actions=CONTROLLER:65535
3: cookie=0x30004, duration=1567.875s, table=3, n_packets=1642101,
   n_bytes=20480637514, priority=4,tcp,
   dl_dst=00:00:00:00:00:03/00:00:00:00:ff:ff,tp_dst=4321
   actions=set_field:00:00:00:00:00:03->eth_dst,goto_table:4
4: cookie=0x30005, duration=1567.874s, table=3, n_packets=0, n_bytes=0,
   priority=4,tcp,dl_dst=00:00:00:00:00:03/00:00:00:00:ff:ff,tp_dst=4322
   actions=set_field:00:00:00:01:00:03->eth_dst,goto_table:4
5: cookie=0x36, duration=1569.020s, table=3, n_packets=0, n_bytes=0,
   priority=3,dl_dst=00:00:00:00:00:03/00:00:00:00:ff:ff
   actions=set_field:00:00:00:00:00:03—>eth_dst,goto_table:4
6: cookie=0x31001, duration=40.018s, table=3, n_packets=134474,
   n_bytes=203324688, priority=5, udp, dl_src=08:00:bb:bb:01:00,
   dl_dst=00:00:00:00:00:03/00:00:00:00:ff:ff,nw_dst=140.0.0.1,
   tp_dst=53 actions=drop
```

Rules 3 and 4 are used to send packets destined to ports `4321` and `4322` towards `c1` and `c2`, respectively. These rules first match on the destination MAC address to verify that a packet is directed towards member `C` (i.e., identified by `00:00:00:00:00:03`). If a packet is matched by rule 3 its destination MAC address is rewritten with the iSDX-identifier of `c1` (i.e., `00:00:00:00:00:03`). The same applies for rule 4 but the destination MAC address is rewritten with the iSDX-identifier of `c2` (i.e., `00:00:00:01:00:03`). Rule 6 is an advanced blackholing rule installed by member `C` for dropping all UDP packets towards IP `140.0.0.1` destined to port `53` that are forwarded by member `A` (i.e., (`dl_src=08:00:bb:bb:01:00`) towards member `C` (i.e., `dl_dst=00:00:00:00:00:03/00:00:00:00:ff:ff`).

We now analyze the content of Table 4 (i.e., the Main-in Table):

```
1: cookie=0x3d, duration=1569.019s, table=4, n_packets=2166781,
   n_bytes=143048175, priority=1 actions=goto_table:5
2: cookie=0x0, duration=1569.359s, table=4, n_packets=0, n_bytes=0,
   priority=0 actions=CONTROLLER:65535
3: cookie=0x39, duration=1569.019s, table=4, n_packets=1642101,
   n_bytes=20480637514,
   priority=4,dl_dst=00:00:00:00:00:03/00:00:03:ff:ff:ff
   actions=set_field:08:00:bb:bb:03:00->eth_dst,goto_table:5
4: cookie=0x3a, duration=1569.019s, table=4, n_packets=0,
   n_bytes=0,
   priority=4,dl_dst=00:00:00:01:00:03/00:00:03:ff:ff:ff
   actions=set_field:08:00:bb:bb:03:01->eth_dst,goto_table:5
5: cookie=0x3b, duration=1569.019s, table=4, n_packets=0,
   n_bytes=0,
   priority=4,dl_dst=00:00:00:00:00:01/00:00:00:00:ff:ff
   actions=set_field:08:00:bb:bb:01:00->eth_dst,goto_table:5
6: cookie=0x3c, duration=1569.019s, table=4, n_packets=554894,
   n_bytes=4146909940,
   priority=4,dl_dst=00:00:00:00:00:02/00:00:00:00:ff:ff
   actions=set_field:08:00:bb:bb:02:00->eth_dst,goto_table:5
```

Rules 3 – 6 are used to rewrite the destination MAC of a packet with the real MAC address of the selected egress interface. For instance, rule 3 replaces `c1`'s iSDX-identifier, i.e., `00:00:00:00:00:03`, with `c1`'s real MAC address, i.e., `08:00:bb:bb:03:00`. All packets are forwarded to table 5.

We now analyze the content of Table 5 (i.e., the Access Control Table):

```
1: cookie=0x1, duration=175.640s, table=1, n_packets=267,
   n_bytes=22934, priority=1000, tcp, nw_src=172.0.0.0/16,
   nw_dst=172.0.0.0/16, tp_dst=179 actions=goto_table:6
2: cookie=0x2, duration=175.639s, table=1, n_packets=400,
   n_bytes=21600, priority=900,tcp,tp_dst=179 actions=drop
3: cookie=0x3, duration=175.639s, table=1, n_packets=400,
   n_bytes=28000, priority=850,ip,nw_proto=89 actions=drop
4: cookie=0x0, duration=176.390s, table=1, n_packets=525,
   n_bytes=38850, priority=0 actions=goto_table:6
```

Rules 1 has the highest priority and allows any BGP packet between any two BGP border routers within the IXP fabric to communicate (`nw_src=172.0.0.0/16`, `nw_dst=172.0.0.0/16`). Rule 2 has a lower priority and drops any BGP traffic from/to external entities. Rule 3 drops any OSPF traffic (`nw_proto=89`) from/to any external entity. Rule 4 is used to forward any other type traffic to the next table.

We now analyze the content of Table 6 (i.e., the Load-Balancing Table). Note that any packet traversing the IXP fabric will traverse this table.

```
1: cookie=0x0, duration=1569.359s, table=6, n_packets=14, n_bytes=980,
   priority=0 actions=CONTROLLER:65535
2: cookie=0x81, duration=1568.432s, table=6, n_packets=17, n_bytes=714,
   priority=10,arp actions=write_metadata:0x40/0xffffffffff,goto_table:7
3: cookie=0x7f, duration=1568.432s, table=6, n_packets=1858,
   n_bytes=142803, priority=10,ip,nw_src=0.0.0.0/0.0.0.1,
   nw_dst=0.0.0.0/0.0.0.1  actions=write_metadata:0x30/0xffffffff,
   goto_table:7
4: cookie=0x7e, duration=1568.432s, table=6, n_packets=3110,
   n_bytes=235410, priority=10,ip,nw_src=0.0.0.1/0.0.0.1,
   nw_dst=0.0.0.0/0.0.0.1  actions=write_metadata:0x20/0xffffffff,
   goto_table:7
5: cookie=0x80, duration=1568.432s, table=6, n_packets=4360473,
   n_bytes=24770336962, priority=10,ip,nw_src=0.0.0.1/0.0.0.1,
   nw_dst=0.0.0.1/0.0.0.1  actions=write_metadata:0x40/0xffffffff,
   goto_table:7
6: cookie=0x7d, duration=1568.432s, table=6, n_packets=2467,
   n_bytes=193150, priority=10,ip,nw_src=0.0.0.0/0.0.0.1,
   nw_dst=0.0.0.1/0.0.0.1  actions=write_metadata:0x10/0xffffffff,
   goto_table:7
```

Rules $3-6$ match packets based on the least significants bits of the source and destination IP addresses and assign one of the four core switches based on the result. The core switches `core-1`, `core-2`, `core-3`, and `core-4` are assigned an identifier 10, 20, 30, and 40, respectively. For instance, rule 3 matches all packets where the least significant bit of the source and destination IP addresses is 0 and writes 40 in the metadata of the packets to signal the intent that this packet will have to traverse `core-4`.

We now analyze the content of Table 7 (i.e., the Umbrella Table):

```
1: cookie=0x2a, duration=1568.616s, table=7, n_packets=822,
   n_bytes=64377,priority=4,metadata=0x30,dl_dst=08:00:bb:bb:03:01
   actions=set_field:04:05:00:00:00:00->eth_dst,output:3
2: cookie=0x24, duration=1568.636s, table=7, n_packets=0,
   n_bytes=0,priority=4,metadata=0x20,dl_dst=08:00:bb:bb:03:00
   actions=set_field:03:05:00:00:00:00->eth_dst,output:2
3: cookie=0x29, duration=1568.616s, table=7, n_packets=2,
   n_bytes=84,priority=4,metadata=0x40,dl_dst=08:00:bb:bb:03:01
   actions=set_field:04:05:00:00:00:00->eth_dst,output:4
4: cookie=0x1f, duration=1568.640s, table=7, n_packets=823,
   n_bytes=64443,priority=4,metadata=0x10,dl_dst=08:00:bb:bb:02:00
   actions=set_field:02:05:00:00:00:00->eth_dst,output:1
5: cookie=0x21, duration=1568.639s, table=7, n_packets=554886,
   n_bytes=4146909212,priority=4,metadata=0x40,dl_dst=08:00:bb:bb:02:00
   actions=set_field:02:05:00:00:00:00->eth_dst,output:4
6: cookie=0x28, duration=1568.617s, table=7, n_packets=0,
    n_bytes=0,priority=4,metadata=0x20,dl_dst=08:00:bb:bb:03:01
    actions=set_field:04:05:00:00:00:00->eth_dst,output:2
 7: cookie=0x20, duration=1568.640s, table=7, n_packets=0,
    n_bytes=0,priority=4,metadata=0x20,dl_dst=08:00:bb:bb:02:00
    actions=set_field:02:05:00:00:00:00->eth_dst,output:2
 8: cookie=0x22, duration=1568.639s, table=7, n_packets=0,
    n_bytes=0,priority=4,metadata=0x30,dl_dst=08:00:bb:bb:02:00
    actions=set_field:02:05:00:00:00:00->eth_dst,output:3
 9: cookie=0x27, duration=1568.619s, table=7, n_packets=0,
    n_bytes=0,priority=4,metadata=0x10,dl_dst=08:00:bb:bb:03:01
    actions=set_field:04:05:00:00:00:00->eth_dst,output:1
10: cookie=0x23, duration=1568.639s, table=7, n_packets=823,
    n_bytes=64443,priority=4,metadata=0x10,dl_dst=08:00:bb:bb:03:00
    actions=set_field:03:05:00:00:00:00->eth_dst,output:1
11: cookie=0x26, duration=1568.636s, table=7, n_packets=0,
    n_bytes=0,priority=4,metadata=0x30,dl_dst=08:00:bb:bb:03:00
    actions=set_field:03:05:00:00:00:00->eth_dst,output:3
12: cookie=0x25, duration=1568.636s, table=7, n_packets=1642103,
    n_bytes=20480637598,priority=4,metadata=0x40,dl_dst=08:00:bb:bb:03:00
    actions=set_field:03:05:00:00:00:00->eth_dst,output:4
13: cookie=0x0, duration=1569.359s, table=7, n_packets=0, n_bytes=0,
    priority=0 actions=CONTROLLER:65535
14: cookie=0x3, duration=1568.642s, table=7, n_packets=0, n_bytes=0,
    priority=8,arp,arp_tpa=172.0.255.253
    actions=set_field:08:00:27:89:33:ff->eth_dst,output:7
15: cookie=0x2, duration=1568.643s, table=7, n_packets=59, n_bytes=2478,
    priority=8,arp,arp_tpa=172.0.255.254
    actions=set_field:08:00:27:89:3b:ff->eth_dst,output:6
16: cookie=0x9, duration=1568.641s, table=7, n_packets=1, n_bytes=42,
    priority=8,arp,arp_tpa=172.0.0.22
    actions=set_field:04:05:00:00:00:00->eth_dst,output:3
17: cookie=0x8, duration=1568.641s, table=7, n_packets=16, n_bytes=672
    priority=8,arp,arp_tpa=172.0.0.21
    actions=set_field:03:05:00:00:00:00->eth_dst,output:3
18: cookie=0x1, duration=1568.646s, table=7, n_packets=28, n_bytes=1176,
    priority=8,arp,arp_tpa=172.0.0.1
    actions=set_field:08:00:bb:bb:01:00->eth_dst,output:5
19: cookie=0x7, duration=1568.642s, table=7, n_packets=18, n_bytes=756,
    priority=8,arp,arp_tpa=172.0.0.11
    actions=set_field:02:05:00:00:00:00->eth_dst,output:3
```

```
20: cookie=0x79, duration=1568.432s, table=7, n_packets=0, n_bytes=0,
    priority=4,dl_dst=00:07:00:00:00:00/00:ff:00:00:00:00
    actions=set_field:08:00:27:89:33:ff->eth_dst,output:7
21: cookie=0x78, duration=1568.432s, table=7, n_packets=3110,
    n_bytes=235410, priority=4,dl_dst=00:06:00:00:00:00/00:ff:00:00:00:00
    actions=set_field:08:00:27:89:3b:ff->eth_dst,output:6
22: cookie=0x77, duration=1568.432s, table=7, n_packets=2163492,
    n_bytes=142790488,priority=4,dl_dst=00:05:00:00:00:00/00:ff:00:00:00:00
    actions=set_field:08:00:bb:bb:01:00->eth_dst,output:5
23: cookie=0x1b, duration=1568.640s, table=7, n_packets=4, n_bytes=168,
    priority=4,dl_dst=08:00:27:89:33:ff actions=output:7
24: cookie=0x19, duration=1568.641s, table=7, n_packets=824,
    n_bytes=64390, priority=4,dl_dst=08:00:bb:bb:01:00 actions=output:5
25: cookie=0x1a, duration=1568.641s, table=7, n_packets=1036,
    n_bytes=78426, priority=4,dl_dst=08:00:27:89:3b:ff actions=output:6
```

Rules 1, 3, 6, and 9 are used to forward a packet to `c2` (i.e., `dl_dst=08:00:bb:bb:03:01`) through the core switch `core-3`, `core-4`, `core-2`, and `core-1`, respectively. Take, for instance, rule 1. Any packet that matches this rule will have its destination MAC address rewritten with the Umbrella encoding of the sequence of outgoing port that will traverse `core-3` and `edge-3` (i.e., `04:05:00:00:00:00`). The first outgoing port, which is not encoded in the packet, but is used in the action of the rule, i.e., `output: 3`. Rules 2, 10, 11, and 12 are used to forward a packet to `c1` (i.e., `dl_dst=08:00:bb:bb:03:00`) through the core switch `core-2`, `core-1`, `core-3`, and `core-4`, respectively. Rules 4, 5, 7, and 8 are used to forward a packet to `b1` (i.e., `dl_dst=08:00:bb:bb:02:00`) through the core switch `core-1`, `core-4`, `core-2`, and `core-3`, respectively. Clearly, packets towards `a1` are not sent through the core switches since `a1` is directly connected to `edge-1`.

Rules $14-19$ deal with ARP traffic. To limit broadcast of ARP requests, the ENDEAVOUR fabric carefully steers the ARP messages towards the intended receivers. Rules 14, 15, and 18 steer ARP requests towards ARP Proxy, the route server, and `a1`, respectively. All these devices are directly connected to `edge-1` so the action corresponding to these rules is simply forwarding a packet to the correct outgoing port. Rules 16, 17 and 19 steer ARP requests towards `c1`, `c2`, and `b1`, respectively. All these devices are *not* directly connected to `edge-1` so the action corresponding to these rules consists of rewriting the destination MAC address with the Umbrella encoded path towards the intended destination.

Rule 22 is part of the Umbrella pipeline that checks whether the second most-significant octet is 05 and, in that case, forwards it to the outgoing port identified by 05. In our case, this port corresponds to the interface that connects `edge-1` to `a1`. In fact, the action of the rule is to rewrite the real MAC address of `a1` in the destination MAC address of the packet.

3.9.2 Dump of `core-1` switch table

A core switch consists of only monitoring and Umbrella tables.

```
1: cookie=0x0, duration=186.742s, table=0, n_packets=252, n_bytes=19410,
   priority=0 actions=goto_table:1
2: cookie=0x0, duration=186.742s, table=1, n_packets=8, n_bytes=560,
   priority=0 actions=CONTROLLER:65535
3: cookie=0x69, duration=186.173s, table=1, n_packets=121, n_bytes=9383,
   priority=4,dl_dst=03:00:00:00:00:00/ff:00:00:00:00:00  actions=output:3
4: cookie=0x67, duration=186.173s, table=1, n_packets=2, n_bytes=84,
   priority=4,dl_dst=01:00:00:00:00:00/ff:00:00:00:00:00  actions=output:1
5: cookie=0x6a, duration=186.173s, table=1, n_packets=0, n_bytes=0,
   priority=4,dl_dst=04:00:00:00:00:00/ff:00:00:00:00:00  actions=output:4
6: cookie=0x68, duration=186.173s, table=1, n_packets=121, n_bytes=9383,
   priority=4,dl_dst=02:00:00:00:00:00/ff:00:00:00:00:00  actions=output:2
```

Rules 3 – 6 are part of the Umbrella pipeline. The first most-significant octet is used to determine the outgoing port.

3.10 Example API: Blackholing REST API

This section briefly describes the implemented REST API to control advanced blackholing. Similar APIs are used to install monitoring and access control rules. Blackholing rules can be created with the following commands:

The ENDEAVOUR environment allows to install rules through a REST API. We extended that Application Programming Interface (API), which sits within each participant controller, so as to handle blackholing policies. To call the API service the following command can be used (e.g., http://localhost:5553/bh/):

```
COMMAND: curl –X GET http://ip_participant_controller:port_api/service/
```

The API returns the installed blackholing rules. To install a blackholing rule, a user can post a specific rule (e.g. at url/schema) with the following command.

```
COMMAND: curl –X POST ––header ’Content–Type: application/json’
```

```
–d '{"inbound": [{"action": {"drop": 0}, "cookie": 4097,
"match": {"eth_src": "08:00:bb:bb:01:00", "udp_dst": 53,
"ipv4_dst": "140.0.0.1"}}]}' http://localhost:5553/bh/
```

To delete a specific blackholing rule, a user needs to know the cookie of the rule and he can then call the API with the following command:

```
COMMAND: curl –X DELETE http://localhost:5553/bh/inbound/4097
```

For every GET command, a user can find additional information in the return header and the HTTP status code. (option `-v` can be used to increase verbosity).

The following example command issues a call for requesting all the blackholing rules:

```
COMMAND: curl –vX GET http://localhost:5553/bh/

OUTPUT:
< HTTP/1.0 200 OK
< Content–location: /bh/inbound
```

In addition, this command returns all the blackholing flow rules in the JSON format.

The following example command requests the inbound blackholing rules:

```
COMMAND: curl –vX GET http://localhost:5553/bh/inbound/

OUTPUT:
< HTTP/1.0 200 OK
< Content–location: /bh/inbound/4097
```

In addition, it returns the inbound blackholing flow rules in the JSON format.

4 Summary

In this report, we described the implemented ENDEAVOUR SDN architecture. We presented the project implementation in terms of code location, contents, and development process. We complemented the use cases videos introduced in Deliverables 4.4 and 4.5 with a complete tutorial-style workflow documentation. This demonstrates the ENDEAVOUR SDN architecture prototype and showcases its ability to support a range of use cases by supporting programming of fine-grained SDN policies in the IXP fabric.

5 Acronyms

SDN	Software Defined Networking
BGP	Border Gateway Protocol
IXP	Internet eXchange Point
IP	Internet Protocol
OSPF	Open Shortest Path First
VM	Virtual Machine
TCP	Transport Control Protocol
UDP	User Datagram Protocol
ARP	Address Resolution Protocol
API	Application Programming Interface
MAC	Media Access Control
REST	REpresentational State Transfer
HTTP	HyperText Transfer Protocol
REST	Representational State Transfer
JSON	JavaScript Object Notation
iSDX	Industrial Software Defined eXchange

References

[1] Kanban board. `https://en.wikipedia.org/wiki/Kanban_board`.

6 Acknowledgement

We would like to acknowledge Yazmin Alanis and Aaron K. Kim from the Noun Project for the Router and Mac Book Air images used in Figure 1, respectively.