ENDEAVOUR: Towards a flexible software-defined network ecosystem

Project name	ENDEAVOUR
Project ID	H2020-ICT-2014-1 Project No. 644960
Working Package Number	3
Deliverable Number	D3.3
Document title	Implementation of the Monitoring Platform
Document version	0.9
Editor in Chief	Castro, QMUL
Authors	Fernandes, Boettger, Antichi, Lapeyrade, Owezarski
Date	31/01/2017
Reviewer	Owezarski, LAAS CNRS
Date of Review	01/01/2017
Status	*Public*

Revision History

Date	Version	Description	Author
28/11/16	0.1	First draft	Antichi, Leao, Boettger, Castro
29/11/16	0.2	CNRS contribution	Lapeyrade, Owezarski
30/11/16	0.3	Demonstration of the monitoring platform	Fernandes
30/11/16	0.5	Add more information to the monitoring architecture	Fernandes
30/11/16	0.6	Description of demonstrator on anomaly detection	Lapeyrade
30/11/16	0.7	Pass on the complete draft	Antichi, Castro
05/12/16	0.8	Executive summary	Castro
05/12/16	0.9	1st review - typos correction	Owezarski

Executive Summary

This is the accompanying report of the demonstrator of Work Package 3 for month 24, where the implementation of the ENDEAVOUR monitoring platform is documented. In this report, we briefly discuss the organization of the code development, we then describe the implementation of the elements of the ENDEAVOUR monitoring platform, and finally, present and document the demonstrators.

Contents

1 Introduction

In this report, we document the implementation of the ENDEAVOUR monitoring platform. Further details of its architecture were presented in the previous deliverable [3]. We first discuss the organization of the project in terms of code location, content, and development process (Section 2), then we document the main elements of ENDEAVOUR monitoring platform involved in the demonstrator (Section 3), and a description of the workflow of the demonstrator (Section 4). Finally, Section 5, concludes this report.

2 Development Environment

The code of the ENDEAVOUR monitoring platform is versioned using the `git` system and hosted on GitHub (`https://github.com/h2020-endeavour`) where the rest of the ENDEAVOUR related code is also held. In particular the branches *"monitor"* and *"anomaly_detection_demo"* contain the relevant code to the monitoring platform. The code development process followed for the implementation of the monitoring architecture is similar to the one used in the rest of the ENDEAVOUR platform and reported in deliverable [2]. For this reason, we do not repeat here those details.

3 Implementation of the monitoring architecture

As presented in the deliverable [3], the monitoring architecture of the ENDEAVOUR platform relies both in data gathered from the Software Defined Networking (SDN) switches, and on data collected using the Open Source Network Tester (OSNT) enabled monitoring probes. In this section we present the implementation of such architecture and the elements that integrate it. The main elements include a monitoring database that stores the relevant per port and per flow statistics, which are retrieved by the platform manager from the SDN switches. The Monitoring Controller is then the element providing both Internet eXchange Point (IXP) members and operators access to the data stored in such database, using an Application Programming Interface (API). The monitoring platform also includes OSNT monitoring probes that run specific software to extend the monitoring capabilities of the ENDEAVOUR platform. We now describe the implementation of each of these elements.

3.1 The Monitoring Controller

In ENDEAVOUR, the Monitoring Controller is a central element that interacts both with the IXP members and operators and enables the aforementioned agents to access statistics collected from the Open Flow (OF) switches through an API.

The Monitoring Controller can receive monitoring flows on its initialization or through a REpresentational State Transfer (REST) interface, giving the options for IXP operators and members to install monitoring flows in the monitor table of the platform. The monitor table is always the initial table of the OF pipeline of the switches in the IXP fabric. A monitoring flow is received by the Monitoring Controller in JavaScript Object Notation (JSON) format as listed by Listing 1. In the example, the flow specified is used to monitor the incidence of broadcast traffic in the switches of the IXP.

```
{
    "monitor_flows": [
        {
            "dpids": [
                1,2,3,4
            ],
            "cookie": 2054,
            "cookie_mask": 1,
            "idle_timeout": 300,
            "hard_timeout": 300,
            "priority": 11111,
            "match": {
                "eth_type": 2054,
                "eth_dst": "ff:ff:ff:ff:ff:ff"
            }
        }
}
```

Listing 1: Example of monitoring flow configuration.

Furthermore, the software installed in the monitoring probes also interacts with the Monitoring Controller, that then informs the platform manager of what are the appropriate rules to install in the switches so as to implement the specific use case triggered by the monitoring efforts.

3.2 The Monitoring Database

The ENDEAVOUR monitoring platform retrieves statistical information from the SDN switches and stores it in a time series database named InfluxDB. The code to save the information in the database is adapted from [1]. A configuration file for every switch of the IXP fabric is required in order to monitor a switch port and flow statistics. Listing 2 shows an example of the YAML Ain't Markup Language (YAML) file required by the platform to monitor a switch. Each configuration file contains a similar description where the switch and the port names are specified as other fields related to sampling interval and the recording of information in the database.

```
---
# The id of the datapath to be controlled
dp_id: 0x0000000000000001
# The name of the datapath for use with logging
name: "Edge-1"
# Purely informational
description: "Edge 1"
hardware: "??"
# whether gauge should monitor stats for ports
monitor_ports: True
# The file to record ports statistics
monitor_ports_file: "ports.out"
# the polling interval for port stats in seconds
monitor_ports_interval: 10
# whether gauge should take periodic flow table dumps
monitor_flow_table: True
# the file to record flow table dumps
monitor_flow_table_file: "ft.out"
# the polling interval for flow table monitoring
monitor_flow_table_interval: 10
influxdb_stats: True
interfaces:
    # name for this port, used for logging/monitoring
    2:
        name: "core 2"
    1:
        name: "core 1"
    4:
        name: "core 4"
    3:
        name: "core 3"
    5:
```

```
    name: "A"
6:
    name: "Route Server"
7:
    name: "Arp Proxy"
```

Listing 2: Monitoring Configuration file.

The database is populated using native OF API in an application present in the Fabric Manager. The statistics retrieved from the switches include all possible per port/flow statistics. The Monitoring controller then is able to execute queries in the database and filter the relevant information for the IXP operator or member. This is a positive feature of the platform as the separation of the logic to populate the database and to retrieve information makes it easier to deploy independent applications that can have an easier access to the required data.

Table 1 presents all the statistics requested for the switches and written in the database.

Statistic	**Description**
flow_byte_count	Total number of bytes matched by a flow
flow_packet_count	Total number of packets matched by a flow
port_bytes_in	Total number of bytes received by a port
port_bytes_out	Total number of bytes sent through a port
port_dropped_in	Number of packet received dropped
port_dropped_out	Number of packets to be sent dropped out
port_errors_in	Number of errors in received packet in a port
port_packets_in	Number of packets received by a port
port_packets_out	Number of packets sent through a port
port_state_reason	Changes to port state

Table 1: Statistics from the switches available in the database

3.3 The Monitoring API

An API is the element bridging the information collected in the database with both IXP members and operators. When a member wants to retrieve some information, the member communicates with the Monitoring Controller, who, using the monitoring API, sends that information to the member requesting it. Slightly different is the case for IXP operators which can create applications directly using the function calls of the monitoring API. It enables the retrieval of statistics to be used in the decision process of some use cases such as load balancing and advanced blackholing.

The API methods are implemented inside a class name StatsCollector, which receives the database client when it is created as shown in Listing 3.

```
INFLUXDB_DB = "sdx"
INFLUXDB_HOST = "localhost"
INFLUXDB_PORT = 8086
INFLUXDB_USER = ""
INFLUXDB_PASS = ""
client = InfluxDBClient(
        host=INFLUXDB_HOST, port=INFLUXDB_PORT,
        username=INFLUXDB_USER, password=INFLUXDB_PASS,
        database=INFLUXDB_DB, timeout=10)
c = StatsCollector(client)
```

Listing 3: Creation of a StatsCollector with an InfluxDB client.

The current available methods to retrieve information from the database and implemented as methods from StatsCollector API are:

- **Port methods.** Port methods get the rate of variate port statistics. All the API calls related to ports receive the same three arguments, which help to improve the time to learn how to use the code. To get the port statistics a user should pass: the interval of the measurements, the name of the switch and the port name. In order to be valid, all these three fields should be present in the configuration files of the Fabric Manager application discussed in 3.2. The available methods are described below:

  `port_bytes_in(interval, dp_name, port_name)` - gets the rate of bytes that entered a port.

  `port_bytes_out(interval, dp_name, port_name)` - gets the rate of bytes sent through a port.

  `port_dropped_in(interval, dp_name, port_name)` - gets the rate of packets dropped by an input port.

  `port_dropped_out(interval, dp_name, port_name)` - get the rate of packet dropped by an output port.

`port_errors_in(interval, dp_name, port_name)` - gets the rate of errors in an input port.

`port_packets_in(interval, dp_name, port_name)` - gets the rate of packets in an input port.

`port_packets_out(interval, dp_name, port_name)` - gets the rate of packets sent through a port.

- **Flow methods.** Flow methods allow the obtention of the rate of packets that match a flow or a group of flows in the switch. There are two methods: one to retrieve the amount of packets per second and another for the number of bytes per second. The required arguments are: the interval of the measurements and the name of the switch. Optional arguments can be passed to improve the results: the value of the flow cookie identifier, the number of the table and the match fields of a flow. It is important to notice that if one of the values in the query does not exist in the database, the result returned will be empty. The available methods are as follow:

  `flow_bytes(self, interval, dp_name, cookie, table_id, flow)` - gets the rate of bytes of a flow or group of flows.

  `flow_packets(self, interval, dp_name, cookie, table_id, flow)` - gets the rate of packets of a flow or a group of flows.

An example of usage of the monitoring API can be seen in section 4.

3.4 The Monitoring probes

While the typical statistical information collected using the OF protocol is rich and wide, not all the use cases considered in ENDEAVOUR can be implemented. Use case with monitoring requirements not supported by the standard native OF API require specific software. While this software can be installed in the platform manager, it forcibly compromises its resources. For this reason, using monitoring probes allows the extension of the capabilities of the platform without compromising its resources, as it was argued in the previous deliverable [2]. The monitoring probes of the ENDEAVOUR monitoring platform are built upon OSNT.

3.4.1 Anomaly detection

In this section we describe how we extend the monitoring capabilities using OSNT monitoring probes, which we exemplifies with the anomaly detection use case. We provide here a brief description of the software used for the detection of anomalies, but the focus of this deliverable is on its monitoring capabilities and on its integration in the ENDEAVOUR monitoring platform using the OSNT monitoring probes. A short description of the anomaly detection software developed for that purpose can be found in the deliverable presenting the corresponding use case [4].

Anomalies (including attacks) are a moving target, as new anomalies and attacks arise every day. To detect them, traffic needs to be characterized and classified (as much as possible) in order to autonomously make decisions concerning the treatment to apply on the isolated traffic classes (legitimate or illegitimate). Relying on human network administrators for deciding whether a flow is legitimate, leads to very poor temporal performances. Unsupervised learning is thus a promising approach for the monitoring of traffic anomalities. An IXP could benefit from such capabilities. In this case we consider an online unsupervised learning algorithms based on clustering [5] to build a system able to detect anomalies autonomously and on real-time.

While this is a rather desirable use case for the ENDEAVOUR platform, it is not directly supported by the native OF API, as it depends on a specific software (i.e., Online and Real-time Unsupervised Network Anomaly Detection Algorithm (ORUNADA)). Whereas such software could run in the ENDEAVOUR fabric manager, it would consume the limited resources of the controller. As argued in previous deliverables [3], we can benefit from the OSNT monitoring probe. The OSNT ability of packet-stripping and accurate timestamping further helps in implementing these monitoring capabilities with a limited impact on the available resources. We now present how the fabric manager mirrors the traffic to the OSNT monitoring probe, and the anomaly detection software that monitors the traffic in search for anomalies. When such an anomaly is detected, the software sends an alert to the Monitoring Controller, which comunicates with the platform manager to ensure that the rules prevent the traffic from entering the fabric are installed.

Integration of the anomaly detection software with the OSNT monitoring probes

In order to fully exploit the capabilities of both OSNT and the anomaly detection software, the two components needed to be jointly revisited. The OSNT platform must be able to send packets (or the essential informations included in those packets) in a way allowing the anomaly detection software to read them. Similarly, the anomaly detection software has to adapt its traffic input procedure to handle packets at the speed provided by OSNT.

For the OSNT part, two major changes needed to be be done to fit the anomaly detection software requirements:

- Add a precise timestamp to each packet, which is a required information for the anomaly detection software. While the software could timestamp packets based on its system clock, it would lack accuracy as it would not timestamp at the time of the packet arrival at the network board. Moreover, it would impose a toll in terms of resource consumption. To deal with this, the hardware timestamp provided by the netFPGA card was directly included into each packed. More precisely, the timestamp was included into the Media Access Control (MAC) destination address slot of the Ethernet frame.
- Re-forge incoming packets in order to save resources and speed-up its processing. By stripping the packets from their payload and any other unnecessary fields, resource consumption is limited. Note that commercial SDN switches lack this capability.

Regarding the anomaly detection software, the main modification relates to the ability to handle the speed at which OSNT provides new packets. Due to the hardware nature of the monitoring probes, the speed at which it provides packets can only be handle by the software component after some modifications of this later. Consequently, we adapted the software in the following manner:

- the anomaly detection software now directly receives packets from the interface, hence reducing I/O operations for the machine;
- packets are separated into micro-slots upon reception, so they can wait for the anomaly detection software to have available resources to process those micro-slots in the right order.

4 Demonstration of the Monitoring Platform

In this section we describe a demonstration of the monitoring platform of ENDEAVOUR providing flow and port statistics, required for use cases such as TE, load balancing, or access control (see [2] for an examination of monitoring requirements).

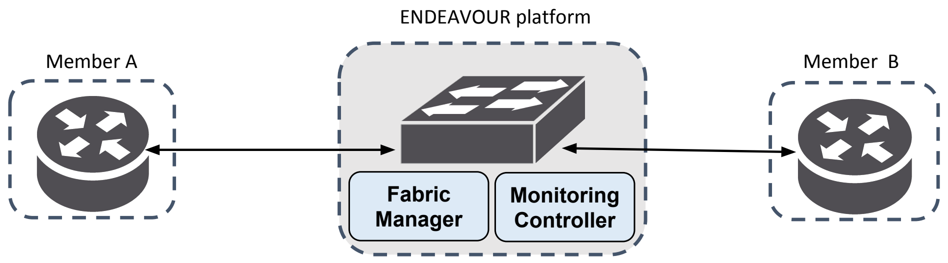


Figure 1: A simple topology for the demonstration.

4.1 Demonstrator description

The monitoring demonstrator is built upon a simple IXP topology with two peering members, depicted in Figure 1. Members A and B exchange Hypertext Transfer Protocol (HTTP) traffic on the Transport Control Protocol (TCP) port 80.

The steps of the demonstration are as follows:

1. The topology is started and the initial monitoring flow is installed in the fabric switches by the Monitoring Controller of the ENDEAVOUR platform. The flow has the goal to monitor the amount of traffic destined to the port 80 in one of the switches of the fabric. The flow is in JSON format, as shown in Listing 4:

```
{
    "monitor_flows": [
        {
            "dpids": [
                1
            ],
            "cookie": 1,
            "cookie_mask": 1,
            "idle_timeout": 500,
            "hard_timeout": 500,
            "priority": 11111,
            "match": {
                "eth_type": 2048,
                "ip_proto": 6,
                "tcp_dst": 80
            }
            }
    }
```

Listing 4: Description of monitor flow.

2. The Fabric Manager requests statistics from the switches, in the configured interval, e.g., every 10 seconds. The flow and port counters are saved as time series in the database. The Fabric Manager saves statistics for every flow and port of the IXP switches. Figure 2 shows the reply messages for the port and flow status requests.

3. Member A sends traffic to port B for the TCP destination port 80.

4. An application presented in Listing 5 is used to demonstrate the API. In the example, the application connects to the database and sends requests at the rate of bytes of the port named `A` in the configuration. Also, it requests the rate of packets destined to the MAC address `ca:fe:bo:ca:fa`. Figure 3 shows the output of the query.

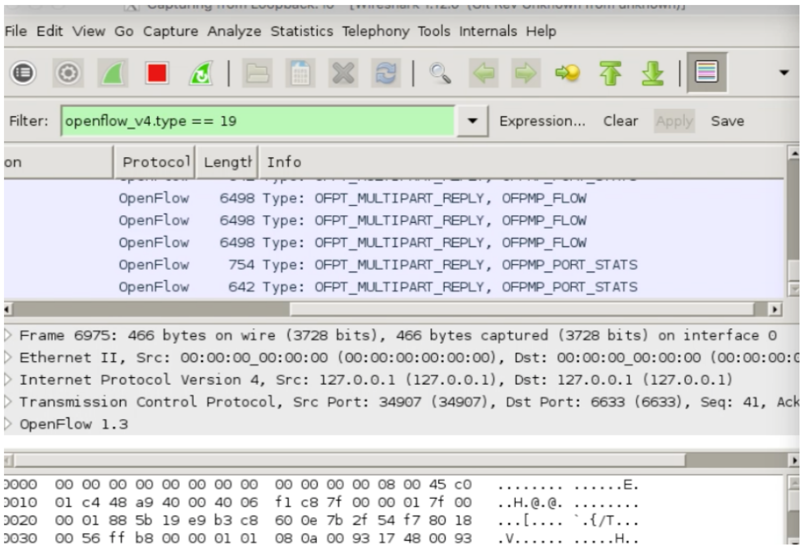
```
client = InfluxDBClient(
host=INFLUXDB_HOST, port=INFLUXDB_PORT,
username=INFLUXDB_USER, password=INFLUXDB_PASS,
database=INFLUXDB_DB, timeout=10)
c = StatsCollector(client)
# Get the rate of bytes going to port A on switch Edge-1
c.port_bytes_out(10, "Edge-1", "A")
flow = {"eth_dst": "ca:fe:bo:ca:fa"}
c.flow_packets(10, Edge-1, flow)
```

Listing 5: API usage in the demonstration.

The graph of the traffic resulting from the monitoring flow is also displayed in the demonstration and can be seen in Figure 4. This flexibility highlights the benefits of using a monitoring database to keep all the data.

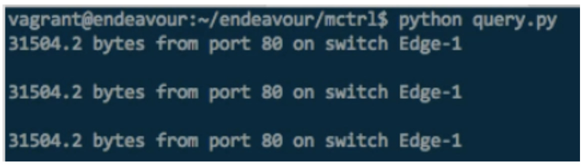
Another important aspect of the demonstration, is that the capabilities to retrive information can be used by the IXP members and operators to dynamically implement use cases. By integrating the monitoring API code, use cases such as load balancing or TE can dynamically change the network based on the current rate of traffic in a port. Moreover, the possibility to install monitoring flows allows fine grained monitoring that further leverages SDN capabilities.

A narrated video of this use case demonstration can be found at:



```
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: openflow_v4.type == 19          Expression... Clear Apply Save

on            Protocol Length Info
              OpenFlow   6498 Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
              OpenFlow   6498 Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
              OpenFlow   6498 Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
              OpenFlow    754 Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
              OpenFlow    642 Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS

▷ Frame 6975: 466 bytes on wire (3728 bits), 466 bytes captured (3728 bits) on interface 0
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:
▷ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▷ Transmission Control Protocol, Src Port: 34907 (34907), Dst Port: 6633 (6633), Seq: 41, Ack
▷ OpenFlow 1.3

000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 c0   ........ ......E.
010  01 c4 48 a9 40 00 40 06  f1 c8 7f 00 00 01 7f 00   ..H.@.@. ........
020  00 01 88 5b 19 e9 b3 c8  60 0e 7b 2f 54 f7 80 18   ...[.... `.{/T...
030  00 56 ff b8 00 00 01 01  08 0a 00 93 17 48 00 93   .V...... .....H..
```

Figure 2: OF stats messages requested by the Fabric Manager.



```
vagrant@endeavour:~/endeavour/mctrl$ python query.py
31504.2 bytes from port 80 on switch Edge-1

31504.2 bytes from port 80 on switch Edge-1

31504.2 bytes from port 80 on switch Edge-1
```

Figure 3: Result of the query for the packet rate of TCP traffic in the port 80.

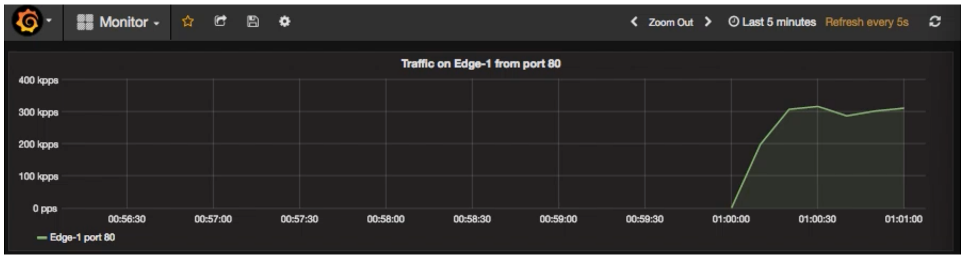


Figure 4: Result of the query for the packet rate of TCP traffic in the port 80.

- `https://www.youtube.com/watch?v=YiKYC2jaqSo`

Reproducing the monitoring demonstration. We used `torch` for orchestrating and scheduling the network events needed to showcase monitoring in the ENDEAVOUR platform. The specification file containing each network event is stored in `iSDX/test/specs/test1-mon.spec`. The monitoring rules of the demonstration can be found in the file `iSDX/test/specs/test1-mon-monitor_flows.cfg`.

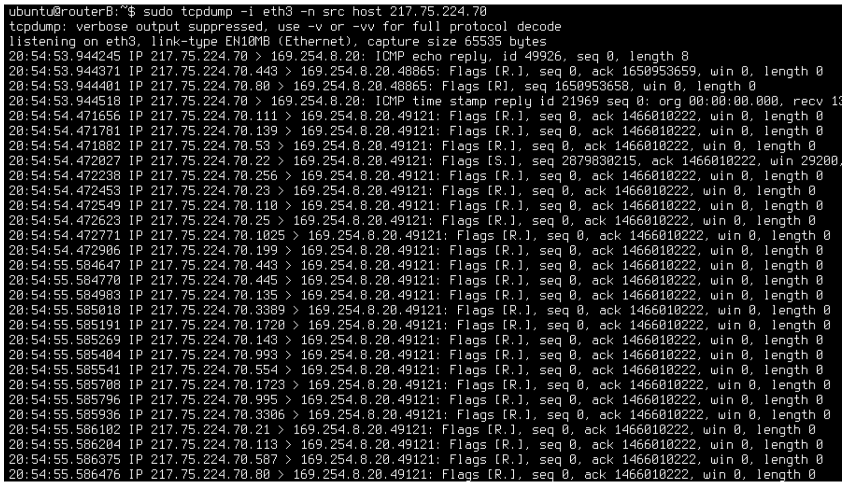
Grafana is the tool used to reproduce the flow of traffic through the IXP fabric. The web-based Grafana interface can be accessed via any browser installed on the Host VM machine by entering the following address `http://localhost:3000`.

4.1.1 Integration of monitoring probes

In this demonstrator we used the same topology described in Figure 1, where members A and B again exchange HTTP traffic.

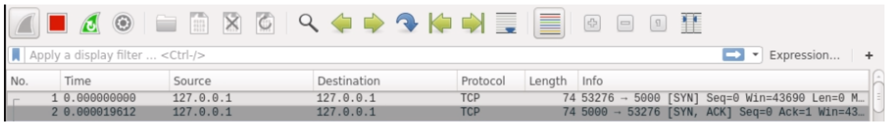
The steps of the demonstration are as follows:

1. The topology is started and IXP member A sends traffic to member B. Such traffic is initially normal, until we inject a TCP attack. Figure 5 show the tpcdump result once the attack traffic has been injected.



```
ubuntu@routerB:~$ sudo tcpdump -i eth3 -n src host 217.75.224.70
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
20:54:53.944245 IP 217.75.224.70 > 169.254.8.20: ICMP echo reply, id 49926, seq 0, length 8
20:54:53.944371 IP 217.75.224.70.443 > 169.254.8.20.48865: Flags [R.], seq 0, ack 1650953659, win 0, length 0
20:54:53.944401 IP 217.75.224.70.80 > 169.254.8.20.48865: Flags [R], seq 1650953658, win 0, length 0
20:54:53.944518 IP 217.75.224.70 > 169.254.8.20: ICMP time stamp reply id 21969 seq 0: org 00:00:00.000, recv 1
20:54:54.471656 IP 217.75.224.70.111 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.471781 IP 217.75.224.70.139 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.471882 IP 217.75.224.70.53 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.472027 IP 217.75.224.70.22 > 169.254.8.20.49121: Flags [S.], seq 2879830215, ack 1466010222, win 29200,
20:54:54.472238 IP 217.75.224.70.256 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.472453 IP 217.75.224.70.23 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.472549 IP 217.75.224.70.110 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.472623 IP 217.75.224.70.25 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.472771 IP 217.75.224.70.1025 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:54.472906 IP 217.75.224.70.199 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.584647 IP 217.75.224.70.443 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.584770 IP 217.75.224.70.445 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.584983 IP 217.75.224.70.135 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585018 IP 217.75.224.70.3389 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585191 IP 217.75.224.70.1720 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585269 IP 217.75.224.70.143 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585404 IP 217.75.224.70.993 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585541 IP 217.75.224.70.554 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585708 IP 217.75.224.70.1723 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585796 IP 217.75.224.70.995 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.585936 IP 217.75.224.70.3306 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.586102 IP 217.75.224.70.21 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.586204 IP 217.75.224.70.113 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.586375 IP 217.75.224.70.587 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
20:54:55.586476 IP 217.75.224.70.80 > 169.254.8.20.49121: Flags [R.], seq 0, ack 1466010222, win 0, length 0
```

Figure 5: Tcpdump trace of the attack sent to router B



```
No.  Time          Source      Destination  Protocol  Length  Info
   1 0.000000000   127.0.0.1   127.0.0.1    TCP           74  53276 → 5000 [SYN] Seq=0 Win=43690 Len=0 M…
   2 0.000019612   127.0.0.1   127.0.0.1    TCP           74  5000 → 53276 [SYN, ACK] Seq=0 Ack=1 Win=43…
```

Figure 6: Traffic capture over the anomaly detection software interface

2. The fabric manager mirrors the traffic to the OSNT monitoring probe.
3. The monitoring probes then strips the packets of the unnecessary fields and passes the packets with their own timestamp to the anomaly detection software, as described in 3.4.1. The software relies on a virtual network interface to check every packet.
4. The anomaly detection software stores every packet received. After initialization, the software engages in a calibration process of 15 seconds of duration.
5. When the software detects an attack, sends an HTTP message to the fabric manager, such as the one in Figure 6. This message lists all the attacks detected during a 1 second time window. The time window is of 1 second, i.e., if during this period there is at least one attack, the software will sent an HTTP message to the fabric manager with all the attacks which took place in the period. Listing 6 shows an example from the demonstrator with the content of such message.
6. This message can then be used to enforce specific filtering policies to prevent the traffic of the attack from reaching its destination. Deliverable [4] demonstrates such use case..

```
{
    "anomalies": [
        {
            "additional_informations": [
                {
                    "feature": "NB_RST",
                    "type": "975.705",
                    "value": "sup"
                },
                {
                    "feature": "PER_SRCPORTS_DEST",
                    "type": "928.2975",
                    "value": "sup"
                },
                {
                    "feature": "PER_NB_RST",
                    "type": "0.7582732156561781",
                    "value": "sup"
                },
                {
                    "feature": "NB_SRCPORTS",
                    "type": "931.825",
                    "value": "sup"
                }
            ],
            "anomaly_description": "RST_attack",
            "anomaly_id": 4,
            "flow_information": {
                "ip_dst": "172.254.30.20"
            },
            "key_type": "ipv4_src",
            "point": "222.175.4.70"
        }
    ],
    "switch": 1,
    "time": "2016-11-25T17:02:08.000Z"
}
```

Listing 6: Example of anomaly report

A narrated video of this use case demonstration can be found at:

- `https://www.youtube.com/watch?v=qSVF58htEIg`

5 Summary

In this report, we document the implementation of the ENDEAVOUR monitoring platform. We discuss the organization of the code development, then we document the elements of ENDEAVOUR monitoring platform and present and describe the demonstrator.

6 Acronyms

SDN	Software Defined Networking
IXP	Internet eXchange Point
TE	Traffic Engineering
VM	Virtual Machine
TCP	Transport Control Protocol
API	Application Programming Interface
MAC	Media Access Control
REST	REpresentational State Transfer
OF	Open Flow
OSNT	Open Source Network Tester
ORUNADA	Online and Real-time Unsupervised Network Anomaly Detection Algorithm
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
YAML	YAML Ain't Markup Language

References

[1] Josh Bailey and Stephen Stuart. Faucet: Deploying sdn in the enterprise. *ACM Queue*, October 2016.

[2] ENDEAVOUR consortium. D.2.3: Implementation of the sdn architecture. *ENDEAVOUR deliverable*, December 2016.

[3] ENDEAVOUR consortium. D.3.2: Design and final requirements of the monitoring platform. *ENDEAVOUR deliverable*, January 2016.

[4] ENDEAVOUR consortium. D.4.5: Software implementing selected use cases for ixp members. *ENDEAVOUR deliverable*, December 2016.

[5] J. Dromard, G. Roudière, and P. Owezarski. Orunada, an online and real-time unsupervised network anomaly detector. *IEEE Transaction on Network and System Management (TNSM)*, 2016.