



UNIVERSITY OF PISA

SCHOOL OF ENGINEERING

MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE  
AND DATA ENGINEERING

ADVANCE NETWORK ARCHITECTURES AND WIRELESS SYSTEMS

---

# Malicious flow quarantine

## SDN-based

---

*Student:*  
Tommaso BALDI

ACADEMIC YEAR 2022-2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Design</b>	<b>2</b>
2.1	Quarantine buffer . . . . .	2
2.2	Traffic Flow . . . . .	2
2.3	Rerouting . . . . .	2
2.4	REST Interface . . . . .	4
2.4.1	Endpoint Flows . . . . .	4
2.4.2	Endpoint Quarantine . . . . .	5
<b>3</b>	<b>Implementation</b>	<b>6</b>
3.1	MaliciousFlowsQuarantine.java . . . . .	6
<b>4</b>	<b>Testing</b>	<b>7</b>
4.1	Mark and Unmark a flow . . . . .	7
4.2	Refresh of the rules . . . . .	8
4.3	Change the buffer size . . . . .	8

# 1 Introduction

The objective of this project is to design and develop a *Floodlight* module that implements a mechanism to reroute traffic flows that are marked as malicious towards a dedicated switch, which will forward the traffic to the controller in order to put it in quarantine. The system shall expose a *RESTful* interface that allows a user to configure and query the module providing the following functionalities:

- allow a user to mark a traffic flow, identified by a client-IP and a server-IP, as malicious;
- allow a user to change the size of the quarantine buffer associated to a malicious flow;
- allow a user to unmark a malicious flow, specifying the buffer releasing mode (e.g., flush);
- allow a user to retrieve the total number of buffered packets associated to a malicious flow.

This document has the following organization: the second section presents the design of the module, describing the traffic flows and the quarantine buffer, it also shows the process of marking/unmarking a new malicious flow and the management information of the REST interface; the third section shows the software implementation of the module; the fourth reports the testing results in different scenarios of the system.

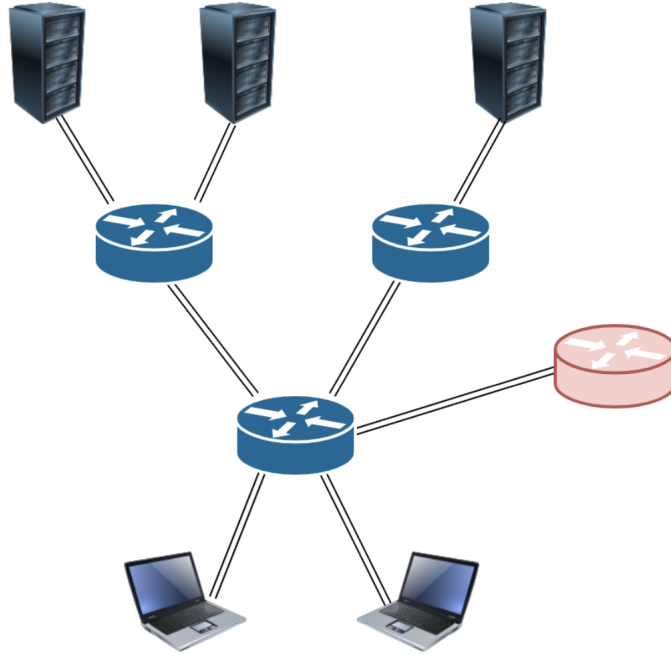


Figure 1: Reference topology, in red the dedicated switch.

## 2 Design

### 2.1 Quarantine buffer

To each marked flow is associated a **quarantine buffer** where are stored its malicious packets. The buffer has a default size of 20, but it can be changed by the user thanks to a REST API. It is implemented as a *circular buffer*, so in case of overflow the new packet will overwrite the oldest packet. The same logic is applied if the user decides to reduce the size of the buffer and there is no enough space for the already buffered packets, also in this case the oldest packet will be dropped.

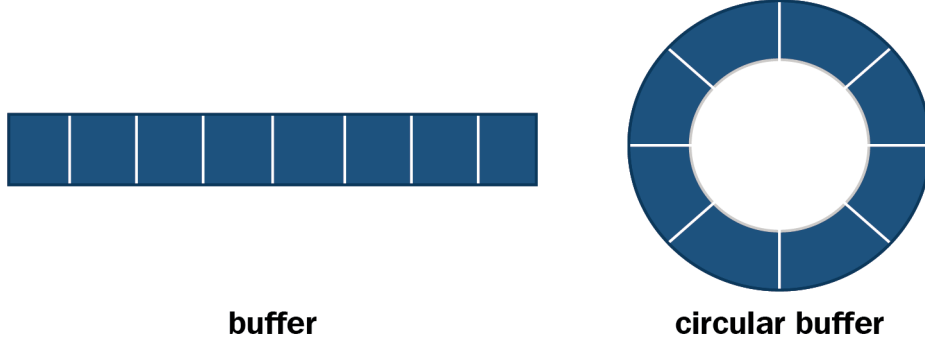


Figure 2: Circular buffer illustration.

### 2.2 Traffic Flow

A marked flow is identified by a *client-IP* and a *server-IP* specified by the user at creation time, later it is also bound to a flow ID used to simplify the REST APIs calls. Flow's data structure includes also a quarantine buffer, previously introduced, and a map where are recorded the active rules for each switch in the network. This map is useful because when a flow is unmarked, we need to remove from all the switches' flow table the rule used to reroute the packets towards the dedicated switch and, in order to do that, we need a reference to messages used to add those rules. We see it more in detail in the next section. The controller record all the marked flows in an array and it uses the single flow data structure to configure the buffer and store useful information.

### 2.3 Rerouting

When a flow is marked, the controller has to reroute all the traffic of the that flow toward a designed router, which will send the packet to the controller in order to be put in quarantine. The rerouting is achieved by setting up a specific flow entry in all the switches of the network, those entries have to instruct the switch about how to forward the packets in order to reach the dedicated switch. To set up a flow entry in the **Flow Table** of a switch the controller must send a `OFPT.FLOW_MOD` message. In this case the message is configured in the following way:

- **Match fields**, used to understand if the rule must be triggered or not, are EtherType, IPv4 destination address and IPv4 source address. The first is a field of the Ethernet frame used to specify which protocol is encapsulated in its

payload, it is set to IPv4. The last two match fields are the client-IP and the server-IP of the marked flow;

- The **priority** is set to the maximum to overcome all the other rules that the switch could have learned so far;
- **Idle and Hard timeout**, they are used to let the rule expire after a certain amount time, they are respectively set to 30 and 60 seconds. It is a good practice to give a life time to the flow entries, because if we consider a real-case scenario where a switch lost the connection with the controller and it does not receive the OFPT\_FLOW\_MOD message where the flow entry is removed at least we are sure that the rule will eventually expire;
- Set a **flag** called SEND\_FLOW\_REM, it is used to tell to the switch to advertise the controller with a OFPT\_FLOW\_REMOVED message when the Flow Table entry is deleted or it is expired;
- In the **Action** field is specified the out-port to use to forward the packet. By default it is set to 1, due to the topology of the network, but we also scan the links of the switch, thanks to the Link Service, in order to check if there is a direct link to the dedicated switch. If a direct link is found we set it as an out-port. In the special case of the dedicated switch the out-port is set with a special value which identifies the controller.

```
OpenFlow 1.3
Version: 1.3 (0x04)
Type: OFPT_FLOW_MOD (14)
Length: 104
Transaction ID: 2104
Cookie: 0x0000000000000000
Cookie mask: 0x0000000000000000
Table ID: 0
Command: OFPFC_ADD (0)
Idle timeout: 30
Hard timeout: 60
Priority: 32768
Buffer ID: OFP_NO_BUFFER (4294967295)
Out port: OFPP_ANY (4294967295)
Out group: OFPG_ANY (4294967295)
Flags: 0x0001
  .... .1 = Send flow removed: True
  .... .0 = Check overlap: False
  .... .0 = Reset counts: False
  .... 0... = Don't count packets: False
  .... 0... = Don't count bytes: False
Pad: 0000
Match
  Type: OFPMT_OXM (1)
  Length: 26
  OXM field
    Class: OFPXM_OPENFLOW_BASIC (0x8000)
    0000 101. = Field: OFPXM_OF_ETH_TYPE (5)
    .... 0 = Has mask: False
    Length: 2
    Value: IPv4 (0x0800)
  OXM field
    Class: OFPXM_OPENFLOW_BASIC (0x8000)
    0001 011. = Field: OFPXM_OF_IPV4_SRC (11)
    .... 0 = Has mask: False
    Length: 4
    Value: 10.0.0.2
  OXM field
    Class: OFPXM_OPENFLOW_BASIC (0x8000)
    0001 100. = Field: OFPXM_OF_IPV4_DST (12)
    .... 0 = Has mask: False
    Length: 4
    Value: 10.0.0.3
  Pad: 000000000000
```

Figure 3: Example of OFPT\_FLOW\_MOD message take from Wireshark.

The controller sends the OFPT\_FLOW\_MOD message only in three specific cases:

- When an user marks a flow with the REST API. In this case, the controller uses the Switch Service to iterate over all the active switches and sets up the rule one by one;
- When the Flow entry expires, the switch advertises the controller, which will check if the rule is still active and, if it is, it will refresh the flow entry;

- When the switch contact the controller to have information to deal with a certain frame through a OFPT\_PACKET\_IN message. In this case, the controller will first check if that packet belongs to a marked flow and, if it is, the controller will setup the flow entry.

Pay attention, this last case could seem useless, because if we setup immediately all the active switch and then we renewed the flow entries ones expired there is no chance to receive a OFPT\_PACKET\_IN message asking for information to the controller about a marked flow. But if we take into account a real-case scenario, different from the reference topology. What happen if at a certain point a new switch join the network? Or if for any reason a switch stop being active and then rejoin the network later? Thanks to the third case, also those scenarios are covered.

When a user unmark a flow, the controller has to send to all the active switches a OFTP\_FLOW\_MOD message to remove the flow entry relative that specific flow. In the flow data structure we have record the active flow entry for each switch, therefore the controller has to iterate over this record and delete each rule one by one. When a flow is unmarked, the user has to specify what the controller should do with the packets in quarantine, there are two options:

- *clear*: drop all the packets;
- *flush*: send the packets in the buffer towards their destination.

In flush mode, the controller builds a OFTP\_PACKET\_OUT message for each buffered packet and sends it to the dedicated switch, which will forward them to the their destinations.

## 2.4 REST Interface

A **RESTful interface** is provided to allow the user (or, for instance, a malicious flows detection system) to manage this module. The controller exposes a set of endpoint that are reachable at the following URLs:

- *http://CONTROLLER-IP:8080/mfq/flows/json*
- *http://CONTROLLER-IP:8080/mfq/quarantine/FLOW-ID/json*

### 2.4.1 Endpoint Flows

This endpoint allows a user to mark/unmark a flow as malicious and retrieve information about the marked flows.

- **GET**: return the list of the marked flows. The information showed for each flow are: the flow ID, client-IP address, server-IP address and the size of the buffer.
- **POST**: this method is used to both mark and unmark a flow. The distinction between the two operations is made by looking at the parameter specified in the payload, if only the client-IP address and the server-IP address are specified it is a mark operation, if it is specified also the mode (e.g., flush) it is an unmark operation.

```

{
  "id": "299d127a-9cce-40fc-98e7-d7cdb871517d",
  "clientIP": "10.0.0.2",
  "serverIP": "10.0.0.4",
  "bufferSize": 20
},
{
  "id": "fea96989-780c-46c2-90b8-ffdc034d51d8",
  "clientIP": "10.0.0.1",
  "serverIP": "10.0.0.3",
  "bufferSize": 20
}

```

Figure 4: Response to GET request.

### 2.4.2 Endpoint Quarantine

This endpoint allows the user to retrieve the number of buffered for a specified flow and it is also possible to change the size of the buffer. In the URL must be specified the flow id.

- **GET:** return the number of buffered packets.
- **POST:** change the size of the buffer with the new size specified in the payload of the request.

```

{
  "client": "10.0.0.1",
  "server": "10.0.0.3",
  "size": "40"
}

```

Figure 5: Payload of the POST request.

### 3 Implementation

The implementation of the solutions presented in this project are inside a single package:

- `net.floodlightcontroller.unipi.maliciousflows`

In the next section, the description of the methods and classes related to the REST interface is omitted for simplicity, anyway the source code contains comments about each part.



Figure 6: Directory structure.

#### 3.1 MaliciousFlowsQuarantine.java

```
private void redirectMaliciousFlow(IOFSwitch sw, Flow flow, OFPacketIn pi)
```

This method is used to send an OFPT\_FLOW\_MOD message to the switch specified in the arguments. The message will instruct the switch about how to set up the new flow entry as described in the previous sections. When the message is ready, it is also stored in the specific flow data structure to record all the active rules in the network.

```
public String markFlow(IPv4Address clientIP, IPv4Address serverIP)
```

This method is used to mark a flow. First, we check if the flow is already marked, then, if it is not the case, we create the flow instance and we add to an array, so the controller can keep track of it. Finally, all the active switches in the network about the new rule.

```
public String unmarkFlow(IPv4Address clientIP, IPv4Address serverIP,
                        String mode)
```

This method is used to unmark a flow. First, we check if the flow is actually marked, if it is the case, we remove the flow from the list of marked flows of the controller, then we retrieve all the packets in its quarantine buffer, if there are any, and then we check the mode specified by the user. If we are working in flush mode, the controller builds a OFTP\_PACKET\_OUT message for each buffered packet and sends it to the dedicated switch, which will forward them to their destinations.

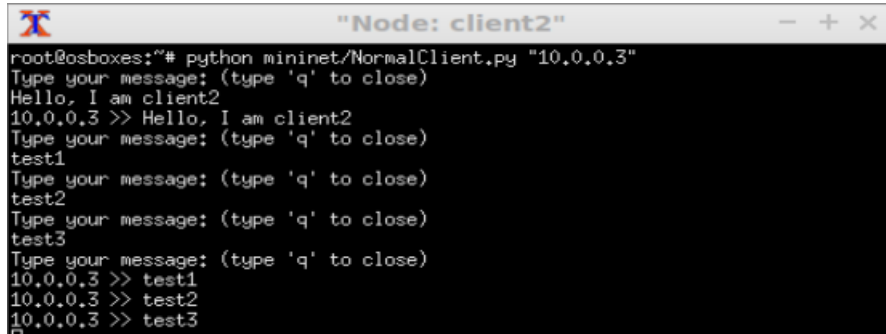


## 4 Testing

In the following section will be described the different tests that aim to demonstrate all the functionalities of the **Malicious Flows Quarantine** module. All the tests have been performed on the reference topology, by simulating a simple *client-server* application. On each of the serves of the network is running an application which is listening to UDP messages and send back an echo reply. On the clients are running two kinds of application: a good one which just interact with the server, in a sort of chat system, and a malicious one which will try to overload the server by sending a long series of messages. The user can test the *RESTful* interface of the module thanks to a demo application, where the user can send HTTP request easily from a user-friendly CLI.

### 4.1 Mark and Unmark a flow

First of all, we will check if after marking a flow the traffic is reroute to the dedicated switch and then the packets are buffered. As we can see in Figure, the first message



```
root@osboxes:~# python mininet/NormalClient.py "10.0.0.3"
Type your message: (type 'q' to close)
Hello, I am client2
10.0.0.3 >> Hello, I am client2
Type your message: (type 'q' to close)
test1
Type your message: (type 'q' to close)
test2
Type your message: (type 'q' to close)
test3
Type your message: (type 'q' to close)
10.0.0.3 >> test1
10.0.0.3 >> test2
10.0.0.3 >> test3
```

Figure 7: Echo replay after marking the flow.

receives immediately an echo reply, then we mark the flow as malicious and the next three messages were pending until we unmark the flow in flush mode. Now let's take a look into how the flow table is different on each switch.

i Flow Table									
Show	10	entries		Search:					
Table No	Pkt.Count	Byte	Duration(s)	Priority	IdleTimeoutSec	HardTimeoutSec	Flags	Instructions	
0x0	0	0	25	32768	30	60	SEND_FLOW_REM	output=1	
0x0	0	0	5	32768	30	60	SEND_FLOW_REM	output=1	

Figure 8: Flow table of a switch after marking a flow as malicious.

i Flow Table									
Show	10	entries		Search:					
Table No	Pkt.Count	Byte	Duration(s)	Priority	IdleTimeoutSec	HardTimeoutSec	Flags	Instructions	
0x0	0	0	25	32768	30	60	SEND_FLOW_REM	output=1	
0x0	0	0	5	32768	30	60	SEND_FLOW_REM	output=1	

Figure 9: Flow table of the dedicated switch after marking a flow as malicious.

As you can see, all the fields are set as previously discussed and the instructions change based on the position of the switch in the network.

## 4.2 Refresh of the rules

This functionality can be easily test thanks to **Wireshark**. We can see that periodically the software tracks OFPT\_FLOW\_REMOVED messages, which are sent by the switch when a rule is removed or it is expired, followed by OFPT\_FLOW\_MOD messages. If we reduce or increase the lifetime of the flow entries, we can notice that the period tracked change too.

```

OpenFll... 146 Type: OFPT FLOW REMOVED
TCP        66 6653 -> 49490 [ACK] Seq=196 Ack=278
OpenFll... 146 Type: OFPT FLOW REMOVED
TCP        66 6653 -> 49262 [ACK] Seq=902 Ack=512
OpenFll... 146 Type: OFPT FLOW REMOVED
TCP        66 6653 -> 49482 [ACK] Seq=434 Ack=278
OpenFll... 146 Type: OFPT FLOW REMOVED
OpenFll... 170 Type: OFPT FLOW MOD
TCP        66 6653 -> 49342 [ACK] Seq=672 Ack=278
OpenFll... 170 Type: OFPT FLOW MOD
OpenFll... 170 Type: OFPT FLOW MOD
OpenFll... 170 Type: OFPT FLOW MOD

```

Figure 10: Refresh of the flow entries tracked by Wireshark.

### 4.3 Change the buffer size

The user can use the REST API to change the size of the quarantine buffer of any marked flow. In this test we have used the malicious client to send 50 packets on the marked flow. In this picture we can see that after unmarking the flow in flush

```
#
#                                     QUARANTINE
#-----
0.      [ 10.0.0.1 -> 10.0.0.3] size = 20
1.      [ 10.0.0.2 -> 10.0.0.4] size = 20
2.      [ 10.0.0.1 -> 10.0.0.5] size = 20

Select the flow (index):

>> 0

>> Buffer = 19/20
```

Figure 11: Retrieve the number of buffered packets with the REST API.

```
root@osboxes:~# python mininet/MaliciousClient.py "10.0.0.3"
type "stop" to terminate the attack!
10.0.0.3> attack - 40 (1/50)
10.0.0.3> attack - 41 (2/50)
10.0.0.3> attack - 42 (3/50)
10.0.0.3> attack - 43 (4/50)
10.0.0.3> attack - 44 (5/50)
10.0.0.3> attack - 45 (6/50)
10.0.0.3> attack - 46 (7/50)
10.0.0.3> attack - 47 (8/50)
10.0.0.3> attack - 48 (9/50)
10.0.0.3> attack - 49 (10/50)
10.0.0.3> attack - 50 (11/50)
10.0.0.3> attack - 31 (12/50)
10.0.0.3> attack - 32 (13/50)
10.0.0.3> attack - 33 (14/50)
10.0.0.3> attack - 34 (15/50)
10.0.0.3> attack - 35 (16/50)
10.0.0.3> attack - 36 (17/50)
10.0.0.3> attack - 37 (18/50)
10.0.0.3> attack - 38 (19/50)
10.0.0.3> attack - 39 (20/50)
```

Figure 12: Attack messages received back.

mode, the malicious client has received back only the 20 most recent attack messages.

Therefore, the quarantine has worked as expected. Let's try to increase the size up to 70, then we will start another attack.

```
#-----#
# QUARANTINE #
#-----#

0.      [ 10.0.0.2 -> 10.0.0.4] size = 20
1.      [ 10.0.0.1 -> 10.0.0.3] size = 70

Select the flow (index):

>> 1

>> Buffer = 50/70
```

Figure 13: Increase the size of the quarantine buffer.

Now the buffer is not filled yet, but if we reduce the size back to 30 we will see that, again, the 20 oldest packets will be lost.

```
type "stop" to terminate the attack:
10.0.0.3 >> attack - 31 (1/50)
10.0.0.3 >> attack - 32 (2/50)
10.0.0.3 >> attack - 33 (3/50)
10.0.0.3 >> attack - 34 (4/50)
10.0.0.3 >> attack - 35 (5/50)
10.0.0.3 >> attack - 36 (6/50)
10.0.0.3 >> attack - 37 (7/50)
10.0.0.3 >> attack - 38 (8/50)
10.0.0.3 >> attack - 39 (9/50)
10.0.0.3 >> attack - 40 (10/50)
10.0.0.3 >> attack - 41 (11/50)
10.0.0.3 >> attack - 42 (12/50)
10.0.0.3 >> attack - 43 (13/50)
10.0.0.3 >> attack - 44 (14/50)
10.0.0.3 >> attack - 45 (15/50)
10.0.0.3 >> attack - 46 (16/50)
10.0.0.3 >> attack - 47 (17/50)
10.0.0.3 >> attack - 48 (18/50)
10.0.0.3 >> attack - 49 (19/50)
10.0.0.3 >> attack - 50 (20/50)
10.0.0.3 >> attack - 21 (21/50)
10.0.0.3 >> attack - 22 (22/50)
10.0.0.3 >> attack - 23 (23/50)
10.0.0.3 >> attack - 24 (24/50)
10.0.0.3 >> attack - 25 (25/50)
10.0.0.3 >> attack - 26 (26/50)
10.0.0.3 >> attack - 27 (27/50)
10.0.0.3 >> attack - 28 (28/50)
10.0.0.3 >> attack - 29 (29/50)
10.0.0.3 >> attack - 30 (30/50)
```

Figure 14: Attack messages received back after changing size of the buffer.