UNIVERSITÀ DI PISA

MSc Computer Engineering
Advanced Network Architectures and Wireless Systems

Flow reservation in Data centers

Federica Perrone

Veronica Torraca

A.Y. 2022-2023

Summary

1. Introduction

The aim of this project is to implement a Floodlight controller module for the Flow reservation in Data Centers.

The system is based on a Software Defined Network environment. In particular, in the proposed system, a leaf-and-spine topology is considered, like the one in Figure 1, and was created using Mininet.

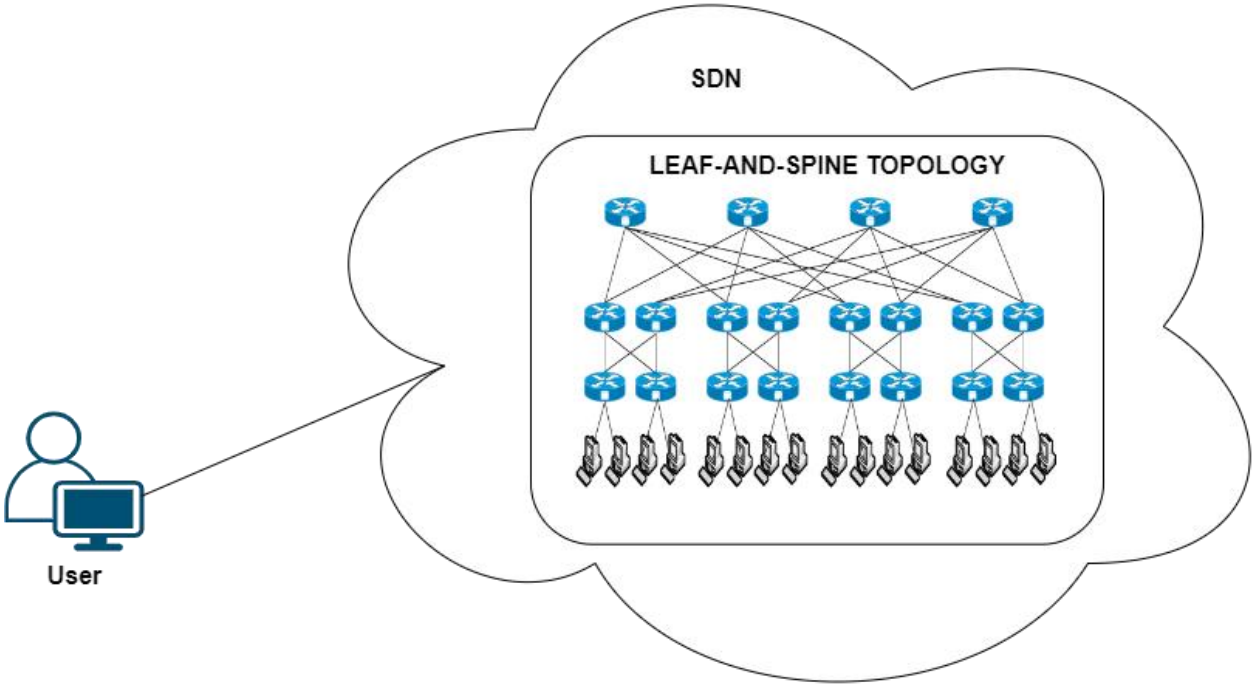


Figure 1 - Overview of the Flow reservation in Data Centers system

The system has to:

1. Expose a RESTful interface allowing hosts to subscribe for a new host-to-host flow, specifying the expected flow load (in GiB)
2. Guarantee that each physical link is reserved for at most one host-to-host flow

In this way, a remote User will be able to subscribe a new host-to-host flow for reserving a path from a source host to a destination host.
When a path is correctly reserved, the hosts connected at the edge of the network can communicate by transmitting large files one to the other.
If no path is available (because all the links are reserved for others host-to-host flows), the flow reservation request is denied.

A reserved path must be maintained until the amount of transmitted data reaches the data load set by the user, and then it can be automatically deallocated (the sooner, the better).

2. Design

2.1 REST interface

A RESTful interface will be exposed to the user, providing a set of end-points that can be used to retrieve network information and to subscribe a new host-to-host flow.

2.1.1 GET “reserved/links”

This end-point allows the User to get the current state of the network links, in particular it shows all the links reserved for host-to-host flows. It is reachable at the following URL:

- http://<CONTROLLER_IP>:8080/dc/network/get/reserved/links/json

The list of reserved links is given in JSON format as below:

```
“src_node: <nodeId> port: <portNumber>”:
    “dest_node: <nodeId> port: <portNumber>”
```

2.1.2 GET “reserved/paths”

This end-point allows the User to get the current state of the network paths, more specifically it shows only the paths currently reserved for a host-to-host flow. It is reachable at the following URL:

- http://<CONTROLLER_IP>:8080/dc/network/get/reserved/paths/json

The list of reserved paths is given in the JSON format as below:

```
“src_ip: <IPv4Address>” : “<List of links>”
```

2.1.3 GET “h2h/flow”

This end-point allows the User to get the current reserved host-to-host flows. It is reachable at the following URL:

- http://<CONTROLLER_IP>:8080/dc/network/get/h2h/flow/json

The list is given in the JSON format as below:

```
“src: <IPv4Address>” : “dest: <IPv4Address>”
```

2.1.4 POST “reserve/flow”

This end-point allows the User to request for a reservation of a new host-to-host flow. It is reachable at the following URL:

- http://CONTROLLER_IP:8080/dc/network/reserve/flow/json

The host-to-host flow is submitted in JSON format specifying the following parameters:

- “srcIp”: IPv4 address of the source host
- “srcMac” : MAC address of the source host
- “destIp” : IPv4 address of the destination host
- “destMac” : MAC address of the destination host

- "dataload" : expected data load to send, in GiB

The module checks if there is an available path (i.e., all the links of a potential path are not already reserved for other host-to-host flow). If no paths are available, the request is denied.

2.2 Host-to-Host flow subscription

To subscribe a new host-to-host flow, the User has to specify the IP address and the MAC address of the source host, the IP address and the MAC address of the destination host, and the expected data load (in GiB).

When the `FlowReservation` controller module receives the subscription request of a new host-to-host flow, some actions have to be performed.

- Check if the given addresses correspond to real hosts in the network. If don't, the error message "`Wrong src/dest IP/MAC: there is no host with the given IP/MAC addresses`" is returned.
- Check if the specified source and/or destination hosts are already part of others subscribed flow paths. If do, the error message "`There is already a path with the given source/destination host`" is returned.
- Retrieve the edge switches corresponding to the source and the destination hosts.
- Compute a configurable number of potential paths between source and destination hosts. This number was set to 20 possible path, in this way the probability of obtaining disjoint paths increases.
- Iterate over the list of detected paths until a path with no reserved link is found. If no path is available, the error message "`There isn't an available path`" is returned and the flow reservation request is denied.
- If there is an available path, this is reserved: the data structures maintaining the network state are updated and flow rules are proactively installed, in flow mode, on the switches belonging to that found path.

For the installation of flow rules on the switches, the MAC addresses of source and destination hosts are used for the `Match` fields, the corresponding action is to send the matching packet on the port connecting the source switch with the next switch of the path.

If the actual switch is the last switch of the path (an edge switch), the packet has to be sent through the output port connected to the destination host.

The hard timeout and the idle timeout are set to zero, in order to have permanent flow rules. In this way, it's up to the controller to proactively remove the flow entries after the amount of transmitted data reaches the data load set by the user. At the same time, also the network state maintained in the controller module is updated.

2.3 Traffic monitoring

Once the flow rules of the path were installed on the switches, the controller starts monitoring the traffic on the last switch of the reserved path, in order to detect when the reserved path can be removed.

To this goal, the bandwidth consumption of the last link of the path is obtained and used to estimate the amount of time needed to transmit the entire data load to the destination. Sometimes, it is not possible to obtain the statistics of the network, because of other Floodlight modules disable the statistic collection, thus a fixed bandwidth was set, to be used when no statistics are available.

When that estimated time is elapsed, the statistics of the corresponding flow entry of the last switch of the path are retrieved, in order to get the number of bytes actually delivered to the destination host. If this number is greater than the initial data load plus a tolerance threshold, the flow is deallocated. Otherwise, if the amount of bytes collected is less than that threshold, the next check will be after a fixed period, set to 15 seconds, since in Floodlight the statistics are collected every 11 seconds.

The tolerance threshold was valued during tests and was necessary because during the transmission of a file, also other bytes must be considered (the packet headers, the packets exchanged for establishing the TCP connection, etc.).

During the experiments different results were obtained: the percentage difference ranges from a minimum of about 0.2% to a maximum of 0.5% of the entire data load, thus the tolerance threshold was set to 0.25% in order to deallocate most of the paths, because the choice was between the probability of never deallocating a path and the probability of deallocating some of them before all data was transmitted.

Furthermore, to manage also the probability of never deallocating a path, a counter was introduced. More specifically, if the number of bytes of the flow remains the same for at least five times, the flow rules are deallocated, because the data load can be considered entirely delivered.

2.4 Host-to-Host flow deallocation

As mentioned before, when the entire data load reached the destination host, the host-to-host flow has to be deallocated.

The flow tables of the switches that are part of the reserved flow are updated in a proactive manner, through `FlowDelete` message in flow mode, to remove the flow rules no longer needed. Moreover, the path to deallocate is removed also from the data structure keeping the network state inside the controller itself.

3. Implementation

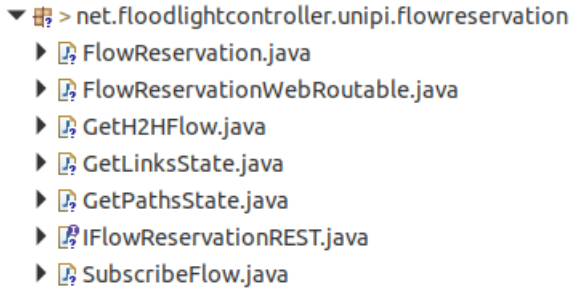


Figure 2 - Implemented package

The implemented package "**`net.floodlightcontroller.unipi.flowreservation`**" contains:

- The main module implementing the flow reservation system: *`FlowReservation.java`*
- Other modules dealing with the REST API management

For simplicity, only the main module of the Flow Reservation system was described, leaving out those for the management of the RESTful interface, because short and easy to understand directly from the code.

3.1 FlowReservation

The `FlowReservation` module is the controller module responsible for the management of the entire flow reservation process.

3.2 Main data structures

The main data structures, used to maintain the abstraction of the network state inside this controller module, are the following:

```
Map<IPv4Address, List<Link>> reservedPaths = new HashMap<>();
```

Data structure storing all the paths reserved for host-to-host flows currently active.

```
List<Link> reservedLinks = new ArrayList<>();
```

Data structure containing all the links which are part of a reserved path.

```
Map<IPv4Address, IPv4Address> requestedH2HFlow = new HashMap<>();
```

Data structure containing all the subscribed host-to-host flows.

3.3 Main methods

The main methods defined are the following:

```
Map<String, String> getLinksState()
```

Returns the list of reserved links.

```
Map<String, String> getPathsState()
```

Returns the list of reserved paths.

```
Map<String, String> getH2HFlow()
```

Returns the list of subscribed host-to-host flows.

```
Int subscribeFlow(Ipv4Address src_ip, MacAddress src_mac, Ipv4Address dest_ip,
MacAddress dest_mac, float dataLoad)
```

Looks for an available path from source host to destination host.

```
Void installFlowRules(Path path, Ipv4Address src_ip, MacAddress src_mac, MacAddress
dest_mac, float dataload, SwitchPort destEdgeSwitchPort)
```

Installs flow rules on all the switches of the path previously found.
At the end, this method starts a thread responsible to monitor the traffic on the last link of the path.

```
void deallocateH2HFlow(IPv4Address src_ip, List<NodePortTuple> switches, Match
match)
```

Deallocates an host-to-host flow.

```
void removePath(IPv4Address src_ip)
```

Updates the data structure in order to remove the reserved path that start from the source host.

```
void removeFlowRules(List<NodePortTuple> switches, Match match)
```

Removes the flow rules on the switches of the path previously installed.

4. Testing

The Flow Reservation system was tested using the Floodlight SDN-controller (an open source OpenFlow controller), with the addition of the implemented modules. Instead, the Data Centers network was simulated using the Mininet application, through which a leaf-and-spine topology, connected to the SDN controller, was created.

The network configuration script was generated using the Mininet Python library, considering the scenario described in the project requirements. Following the leaf-and-spine topology, the network involves several hosts at the edge of the network and a number of switches (edge, aggregate or core), providing connectivity.

To analyze the flow of messages sent in the SDN, the packets directed to a destination host of a reserved path was captured and displayed using the Wireshark software, as a way to demonstrate the test results.

4.1 Flow reservation testing

To observe and test the basic functionalities of the Flow Reservation system, as already mentioned, a leaf-and-spine topology in Mininet emulator was deployed, providing 16 hosts connected using three level of switches: 8 edge switches, 8 aggregate switches and 4 core switches, as illustrated in Fig. 3.

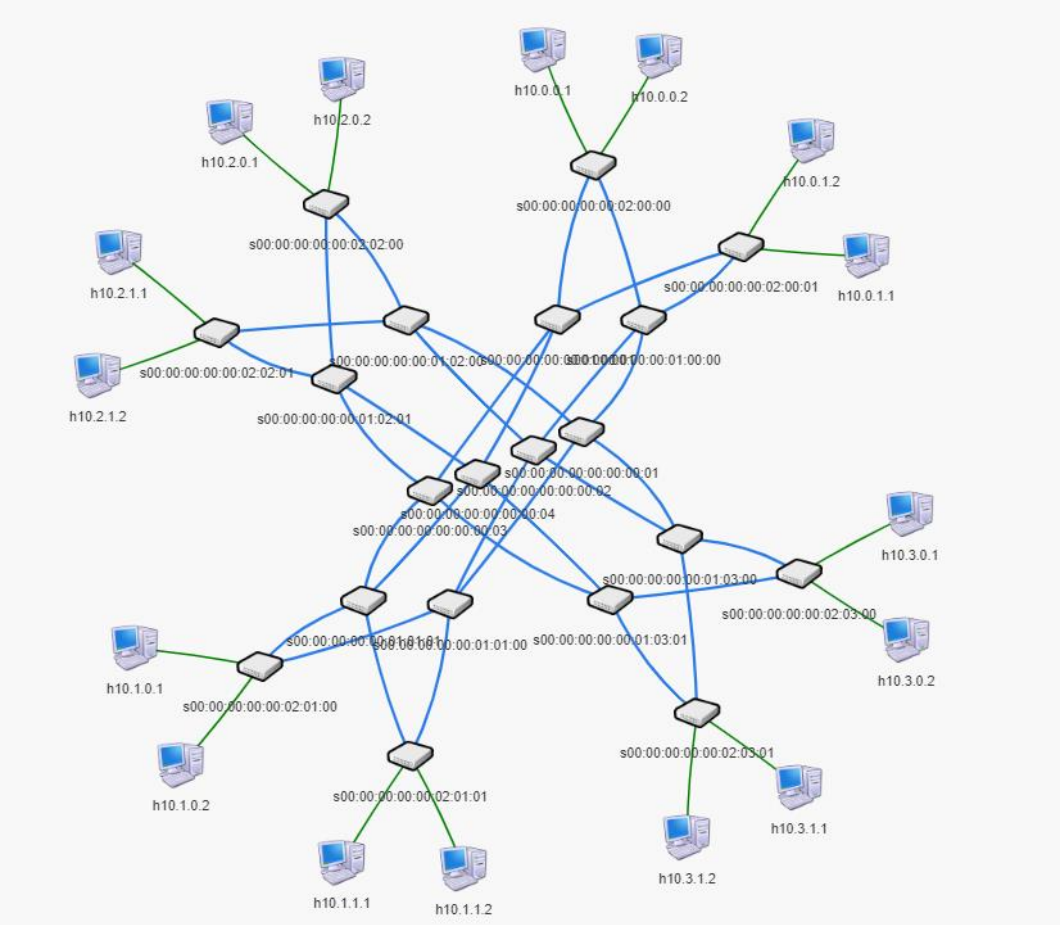


Figure 3 - Mininet topology

The first step to do to test the functionalities of the Flow Reservation system is a POST request using the `FlowReservation RESTful` interface. To this aim, a python script was implemented, in which the following parameters have to be specified, in order to use them in the POST request:

- `srcIP: 10.0.0.1`
- `srcMac: 02:00:00:00:00:00`
- `destIP: 10.1.0.1`
- `destMac: 02:00:00:01:00:00`
- `dataload: 4GiB`

```
student@osboxes:~$ python post_request.py 10.0.0.1 02:00:00:00:00:00 10.1.0.1 02:00:00:01:00:00 4
Namespace(data_load=4.0, dest_ip='10.1.0.1', dest_mac='02:00:00:01:00:00', src_ip='10.0.0.1', src_mac='02:00:00:00:00:00')
OK: path reserved
student@osboxes:~$ python post_request.py 10.0.0.2 02:00:00:00:00:01 10.1.0.2 02:00:00:01:00:01 4
Namespace(data_load=4.0, dest_ip='10.1.0.2', dest_mac='02:00:00:01:00:01', src_ip='10.0.0.2', src_mac='02:00:00:00:00:01')
OK: path reserved
student@osboxes:~$ python post_request.py 10.0.1.1 02:00:00:00:01:00 10.3.0.2 02:00:00:03:00:01 4
Namespace(data_load=4.0, dest_ip='10.3.0.2', dest_mac='02:00:00:03:00:01', src_ip='10.0.1.1', src_mac='02:00:00:00:01:00')
OK: path reserved
student@osboxes:~$ python post_request.py 10.1.1.1 02:00:00:01:01:00 10.3.1.1 02:00:00:03:01:00 4
Namespace(data_load=4.0, dest_ip='10.3.1.1', dest_mac='02:00:00:03:01:00', src_ip='10.1.1.1', src_mac='02:00:00:01:01:00')
OK: path reserved
student@osboxes:~$ python post_request.py 10.0.1.2 02:00:00:00:01:01 10.3.1.2 02:00:00:03:01:01 4
Namespace(data_load=4.0, dest_ip='10.3.1.2', dest_mac='02:00:00:03:01:01', src_ip='10.0.1.2', src_mac='02:00:00:00:01:01')
Reservation requested denied: No path available
```

Figure 4 - POST request

Once a path was correctly reserved, it is possible to get the links reserved for all the host-to-host flows currently active, with a GET request to the `FlowReservation RESTful` interface. In Fig. 5 a possible output is shown.


```
{
  "src_ip: 10.0.0.1": "src_node: 00:00:00:00:00:02:00:00 port: 1 dest_node: 00:00:00:00:00:01:00:00 port: 3; src_node: 00:00:00:00:
00:01:00:00 port: 2 dest_node: 00:00:00:00:00:00:00:02 port: 1; src_node: 00:00:00:00:00:00:00:02 port: 2 dest_node: 00:00:00:00:0
0:01:01:00 port: 2; src_node: 00:00:00:00:00:01:01:00 port: 3 dest_node: 00:00:00:00:00:02:01:00 port: 1; ",
  "src_ip: 10.0.1.1": "src_node: 00:00:00:00:00:02:00:01 port: 1 dest_node: 00:00:00:00:00:01:00:00 port: 4; src_node: 00:00:00:00:
00:01:00:00 port: 1 dest_node: 00:00:00:00:00:00:00:01 port: 1; src_node: 00:00:00:00:00:00:00:01 port: 4 dest_node: 00:00:00:00:0
0:01:03:00 port: 1; src_node: 00:00:00:00:00:01:03:00 port: 3 dest_node: 00:00:00:00:00:02:03:00 port: 1; ",
  "src_ip: 10.0.0.2": "src_node: 00:00:00:00:00:02:00:00 port: 2 dest_node: 00:00:00:00:00:01:00:01 port: 3; src_node: 00:00:00:00:
00:01:00:01 port: 2 dest_node: 00:00:00:00:00:00:00:04 port: 1; src_node: 00:00:00:00:00:00:00:04 port: 2 dest_node: 00:00:00:00:0
0:01:01:01 port: 2; src_node: 00:00:00:00:00:01:01:01 port: 3 dest_node: 00:00:00:00:00:02:01:00 port: 2; ",
  "src_ip: 10.1.1.1": "src_node: 00:00:00:00:00:02:01:01 port: 2 dest_node: 00:00:00:00:00:01:01:01 port: 4; src_node: 00:00:00:00:
00:01:01:01 port: 1 dest_node: 00:00:00:00:00:00:00:03 port: 2; src_node: 00:00:00:00:00:00:00:03 port: 4 dest_node: 00:00:00:00:0
0:01:03:01 port: 1; src_node: 00:00:00:00:00:01:03:01 port: 4 dest_node: 00:00:00:00:00:02:03:01 port: 2; "
}
```


Figure 5 - Reserved paths with links belonging to it

In Figure 6 an example of a host-to-host flow reservation on the implemented topology is illustrated.

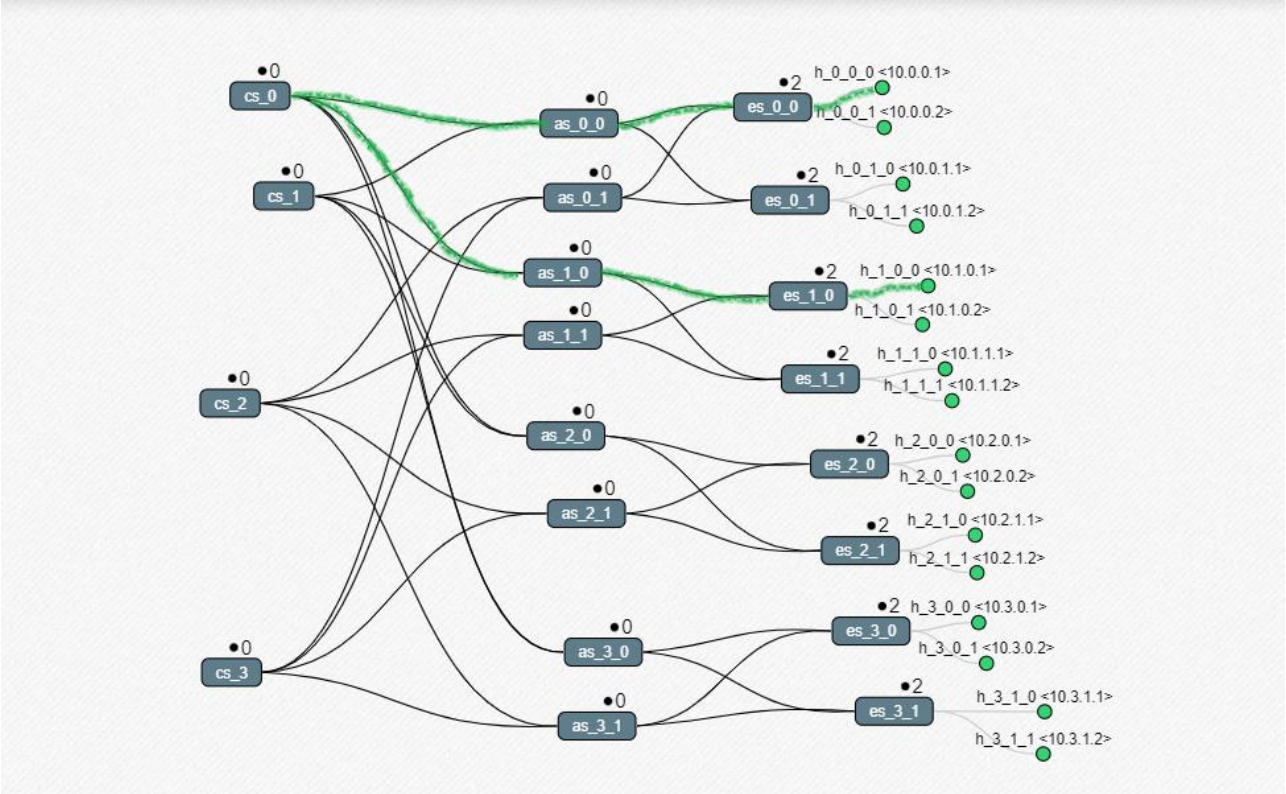


Figure 6 - Reserved path on the implemented topology

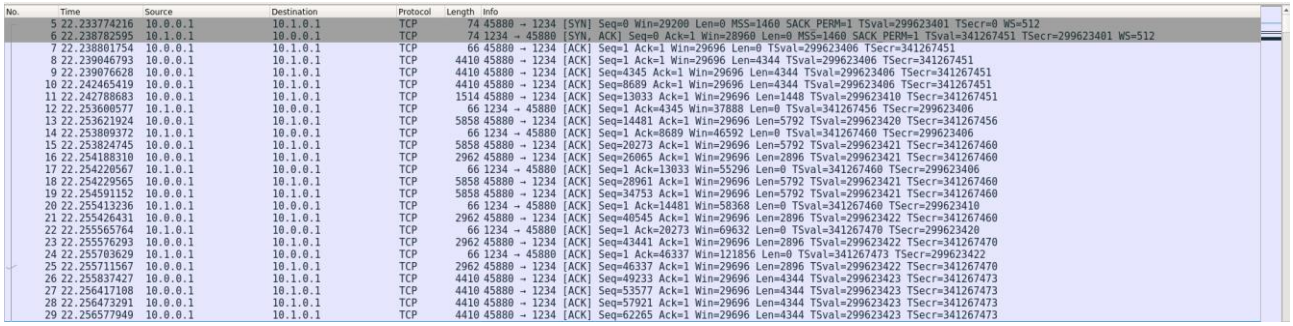
To be sure that in all the switches belonging to the path, the corresponding flow rules was correctly installed, their flow table was checked. In the Figure 7 an example of a switch flow table is shown, and it is possible to observe that there is a flow entry corresponding with the reserved path.

```
student@osboxes:~$ sudo ovs-ofctl dump-flows es_0_0 -O OpenFlow13
 cookie=0x0, duration=753.762s, table=0, n_packets=126870, n_bytes=4306908580, dl_src=02:00:00:00:00:00,dl_dst=02:00:00:01:00:00 actions=output:"es_0_0-eth1"
 cookie=0x0, duration=885.102s, table=0, n_packets=410, n_bytes=29096, priority=0 actions=CONTROLLER:65535
```

Figure 7 - Edge source switch flow table

The second step was to send a file from the source host to the destination host. To this aim, a simple shell script was implemented, in which the `netcat` command was used.

To check if the file packets were correctly delivered to the destination, they were captured with Wireshark.



No.	Time	Source	Destination	Protocol	Length	Info
5	22.233774216	10.0.0.1	10.1.0.1	TCP	74	45880 → 1234 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=299623401 TSecr=0 WS=512
6	22.238782595	10.1.0.1	10.0.0.1	TCP	74	1234 → 45880 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=341267451 TSecr=299623401 WS=512
7	22.238801754	10.0.0.1	10.1.0.1	TCP	66	45880 → 1234 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=299623406 TSecr=341267451
8	22.239046793	10.0.0.1	10.1.0.1	TCP	4410	45880 → 1234 [ACK] Seq=1 Ack=1 Win=29696 Len=4344 TSval=299623406 TSecr=341267451
9	22.239076628	10.0.0.1	10.1.0.1	TCP	4410	45880 → 1234 [ACK] Seq=4345 Ack=1 Win=29696 Len=4344 TSval=299623406 TSecr=341267451
10	22.242465419	10.0.0.1	10.1.0.1	TCP	4410	45880 → 1234 [ACK] Seq=8689 Ack=1 Win=29696 Len=4344 TSval=299623406 TSecr=341267451
11	22.242788683	10.0.0.1	10.1.0.1	TCP	1514	45880 → 1234 [ACK] Seq=13033 Ack=1 Win=29696 Len=1448 TSval=299623410 TSecr=341267451
12	22.253600577	10.1.0.1	10.0.0.1	TCP	66	1234 → 45880 [ACK] Seq=1 Ack=4345 Win=37888 Len=0 TSval=341267456 TSecr=299623406
13	22.253621924	10.0.0.1	10.1.0.1	TCP	5858	45880 → 1234 [ACK] Seq=14481 Ack=1 Win=29696 Len=5792 TSval=299623420 TSecr=341267456
14	22.253809372	10.1.0.1	10.0.0.1	TCP	66	1234 → 45880 [ACK] Seq=1 Ack=8689 Win=46592 Len=0 TSval=341267460 TSecr=299623406
15	22.253824745	10.0.0.1	10.1.0.1	TCP	5858	45880 → 1234 [ACK] Seq=20273 Ack=1 Win=29696 Len=5792 TSval=299623421 TSecr=341267460
16	22.254188310	10.0.0.1	10.1.0.1	TCP	2962	45880 → 1234 [ACK] Seq=26065 Ack=1 Win=29696 Len=2896 TSval=299623421 TSecr=341267460
17	22.254220567	10.1.0.1	10.0.0.1	TCP	66	1234 → 45880 [ACK] Seq=1 Ack=13033 Win=55296 Len=0 TSval=341267460 TSecr=299623406
18	22.254229565	10.0.0.1	10.1.0.1	TCP	5858	45880 → 1234 [ACK] Seq=28961 Ack=1 Win=29696 Len=5792 TSval=299623421 TSecr=341267460
19	22.254591152	10.0.0.1	10.1.0.1	TCP	5858	45880 → 1234 [ACK] Seq=34753 Ack=1 Win=29696 Len=5792 TSval=299623421 TSecr=341267460
20	22.255413236	10.1.0.1	10.0.0.1	TCP	66	1234 → 45880 [ACK] Seq=1 Ack=14481 Win=58368 Len=0 TSval=341267460 TSecr=299623410
21	22.255426431	10.0.0.1	10.1.0.1	TCP	2962	45880 → 1234 [ACK] Seq=40545 Ack=1 Win=29696 Len=2896 TSval=299623422 TSecr=341267460
22	22.255565764	10.1.0.1	10.0.0.1	TCP	66	1234 → 45880 [ACK] Seq=1 Ack=20273 Win=69632 Len=0 TSval=341267470 TSecr=299623420
23	22.255576293	10.0.0.1	10.1.0.1	TCP	2962	45880 → 1234 [ACK] Seq=43441 Ack=1 Win=29696 Len=2896 TSval=299623422 TSecr=341267470
24	22.255703629	10.1.0.1	10.0.0.1	TCP	66	1234 → 45880 [ACK] Seq=1 Ack=46337 Win=121856 Len=0 TSval=341267473 TSecr=299623422
25	22.255711567	10.0.0.1	10.1.0.1	TCP	2962	45880 → 1234 [ACK] Seq=46337 Ack=1 Win=29696 Len=2896 TSval=299623422 TSecr=341267470
26	22.255837427	10.0.0.1	10.1.0.1	TCP	4410	45880 → 1234 [ACK] Seq=49233 Ack=1 Win=29696 Len=4344 TSval=299623423 TSecr=341267473
27	22.256417108	10.0.0.1	10.1.0.1	TCP	4410	45880 → 1234 [ACK] Seq=53577 Ack=1 Win=29696 Len=4344 TSval=299623423 TSecr=341267473
28	22.256473291	10.0.0.1	10.1.0.1	TCP	4410	45880 → 1234 [ACK] Seq=57921 Ack=1 Win=29696 Len=4344 TSval=299623423 TSecr=341267473
29	22.256577949	10.0.0.1	10.1.0.1	TCP	4410	45880 → 1234 [ACK] Seq=62265 Ack=1 Win=29696 Len=4344 TSval=299623423 TSecr=341267473

Figure 8 – Wireshark: captured packet from source host to destination host

After a while, the `FlowReservation` system checks if the data load was entirely delivered at the destination host, and if so deallocates the host-to-host flow (Figure 9), otherwise it performs periodic checks until the data reaches the expected flow load.

```
2023-03-27 12:16:53.204 INFO  [n.f.u.f.FlowReservation] Enable statistics: wait collection of statistics
2023-03-27 12:16:53.403 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2023-03-27 12:16:53.435 INFO  [n.f.t.TopologyManager] Recomputing topology due to: link-discovery-updates
2023-03-27 12:17:04.304 INFO  [n.f.u.f.FlowReservation] bytes delivered: 2497097524
2023-03-27 12:17:04.304 INFO  [n.f.u.f.FlowReservation] packets: 68614
2023-03-27 12:17:04.304 INFO  [n.f.u.f.FlowReservation] File not delivered to destination yet. Wait until next check
2023-03-27 12:17:08.886 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2023-03-27 12:17:08.909 INFO  [n.f.t.TopologyManager] Recomputing topology due to: link-discovery-updates
2023-03-27 12:17:09.616 INFO  [n.f.t.TopologyManager] Recomputing topology due to: link-discovery-updates
2023-03-27 12:17:19.304 INFO  [n.f.u.f.FlowReservation] Enable statistics: wait collection of statistics
2023-03-27 12:17:23.942 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2023-03-27 12:17:24.467 INFO  [n.f.t.TopologyManager] Recomputing topology due to: link-discovery-updates
2023-03-27 12:17:30.406 INFO  [n.f.u.f.FlowReservation] bytes delivered: 3365766472
2023-03-27 12:17:30.406 INFO  [n.f.u.f.FlowReservation] packets: 93308
2023-03-27 12:17:30.406 INFO  [n.f.u.f.FlowReservation] File not delivered to destination yet. Wait until next check
2023-03-27 12:17:31.166 ERROR [n.f.c.w.SwitchStatisticsResource] Invalid or unimplemented stat request type features
2023-03-27 12:17:39.74 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2023-03-27 12:17:45.412 INFO  [n.f.u.f.FlowReservation] Enable statistics: wait collection of statistics
2023-03-27 12:17:54.102 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2023-03-27 12:17:56.512 INFO  [n.f.u.f.FlowReservation] bytes delivered: 4305592110
2023-03-27 12:17:56.512 INFO  [n.f.u.f.FlowReservation] packets: 120263
2023-03-27 12:17:56.512 INFO  [n.f.u.f.FlowReservation] File delivered to destination
2023-03-27 12:17:56.512 INFO  [n.f.u.f.FlowReservation] Deallocate h2h flow
2023-03-27 12:17:56.512 INFO  [n.f.u.f.FlowReservation] Data structure updated correctly
2023-03-27 12:17:56.516 INFO  [n.f.u.f.FlowReservation] Flow rules removed correctly
```

Figure 9 – Check delivered data load and host-to-host flow deallocation

The results from these tests then demonstrate the correct behavior of the `FlowReservation` system, allowing two hosts connected at the edge of the network to communicate by sending large files and the subsequent automatic deallocation of the flow rules from the flow table of the switches belonging to the considered reserved path. The file packets directed to the destination host were then correctly managed and delivered.