# TP Virtualisation dans l'embarqué

**Mise en œuvre d'un réseau AFDX à l'aide du langage P4**

Master REOC

2020-2021

## 1 Objectifs

The main objective of this practical work is to build an AFDX network architecture. Unfortunately, we can't have AFDX switches or airplane computers (End Systems). So we propose to use the P4 language in order to simulate the behavior of an AFDX switch. Once the P4 code is compiled, the network architecture will be implemented in the Mininet emulation tool.
The behavior of the ES will be to periodically emit data to simulate the emission of data at each BAG.

## 2 Quelques caractéristiques AFDX

Data transmission on an AFDX network is done through Virtual Links (VL). These VLs are single-transmitter and multicast streams.
Figure **??** shows the different fields of the AFDX frame (which is similar to a classical Ethernet frame). The destination address field shows the VL number. The End-Systems transmit the AFDX frames regularly while respecting the BAG constraint (delay between 2 frames of the same LV, with a value between 1 and 128 ms). The role of the switch is to transmit VLs arriving from an input port to one or more output ports. Data filtering in the switch is done at the destination MAC address. The switch must therefore recognize the VL number and redirect the frame to the output port(s). Frames are sent to the output ports according to a FIFO policy (no regulation is made). On the edge equipment, the regularity of the BAG is checked on the input ports.

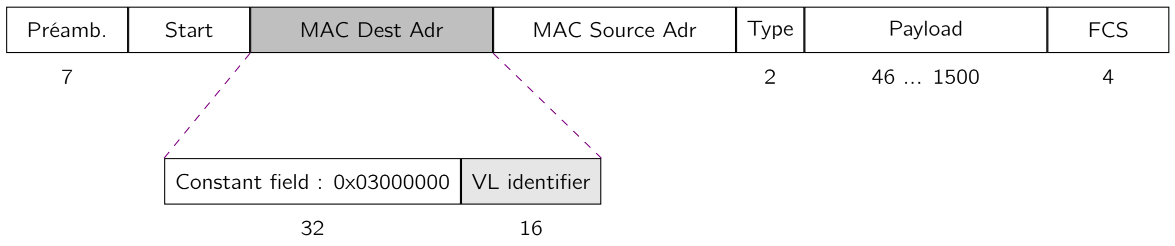


Figure 1: Format of an AFDX frame

Download the application `p4app` available here :

```
wget https://raw.githubusercontent.com/p4lang/p4app/master/p4app
```

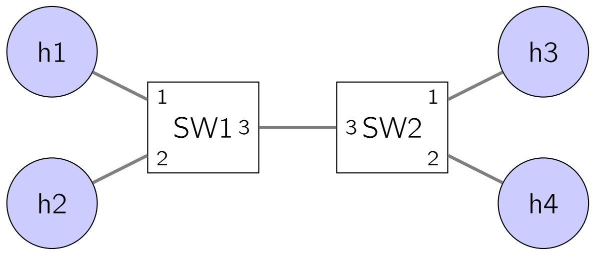
This application works with docker and allows to use the tools associated with P4. Updating the environment is done via :

```
./p4app update
```

Your environment should now be ready.

## 3 Implementation of AFDX Switching

The objective of the TP is to implement and emulate with Mininet the following AFDX architecture:



It is composed of 4 hosts (the End Systems, rated ES), rated h1 to h4, and 2 AFDX switches, SW1 and SW2. The port numbers are shown in the figure (e.g. ES h1 is connected to port 1 of switch SW1). The ES emit VLs periodically. The configuration of the VLs is as follows:

| VL Num. | BAG in ms | source | destinations |
|---|---|---|---|
| 1 | 2 | h1 | h2, h3 |
| 2 | 4 | h1 | h4 |
| 3 | 4 | h2 | h1, h4 |
| 4 | 2 | h2 | h3 |
| 5 | 8 | h3 | h1 |
| 6 | 1 | h3 | h4 |
| 7 | 4 | h4 | h1, h2, h3 |

VLs are multicast streams, so they can have several destinations.

### 3.1 Provided files

Download the file `afdx.p4app`.
This file is an archive for `p4app` containing the following files :

- afdx.p4: source of the P4 program modeling the behavior of the AFDX switch;
- command_1.txt and command_2.txt : configuration of the switch table of SW1 and SW2 ;
- send_h1.py : python script allowing the transmission of an AFDX frame;
- topo.py : python script to create the network architecture in Mininet ;
- p4app.json : configuration file for `p4app`.

**To do.** To unarchive, execute the command :

```
p4app unpack afdx.p4app
```

You should get a directory named `afdx.p4app`.
The network topology is described in a text file named `topo.txt`. In this file, the stations are called hosts and are numbered h1, h2, . . . The switches are named s1, s2, . . . The first 2 lines of the file indicate the number of switches and hosts in the :

```
switches nb_sw
hosts nb_host
```

In the file, the links are described line by line as follows:

```
h1 s1
h2 s1
```

In this example, there is a link between Host 1 and Switch 1 and a link between Host 2 and Switch 1. The order in which the links to a switch are configured also corresponds to the port number of the switch to which the link is connected. For example, h1 is connected to port 1 and h2 is connected to port 2 of s1.

**To do.** Create a file `topo.txt` in which you will indicate the links between the devices.

## 3.2 Modeling AFDX switching in P4

The file `afdx.p4` is only a skeleton for the modeling of the switching in P4. It will thus be necessary to complete it. We consider here that unicast VLs. The steps are as follows:

1. Define the header of an AFDX frame.
2. Define the reading of a frame in the `parse_afdx_frame` parser which rejects frames other than AFDX.
3. Write the actions associated with the switch table.
4. Write the processing of the switch table.
5. Start the switch table processing in the ingress control.

**To do.** Complete the file `afdx.p4`.

## 3.3 The switch table

The files `commands_1.txt` and `commands_2.txt` contain the switch tables for switches 1 and 2, respectively. For each table in the file, a default action must be defined :

```
table_set_default action_table name
```

Then we add the actions to be performed :

```
table_add action_table name conditions => action_parameters
```

**To do.** Build the switch tables for switches 1 and 2 of the considered network architecture.

## 3.4 Test the architecture

Once the files have been built, we can launch the execution of the :

```
p4app run afdx.p4app
```

The python script `send_h1.py` sends an AFDX VL 1 frame through the h1 station. In mininet, to launch it, type `h1 python send_h1.py`. It is possible to visualize the data sent and received by the different hosts by executing on the command line :

```
p4app exec m h1 tcpdump -U
```

**To do.** Simulate the transmission of VLs by the stations and view the transmissions on the destination stations. Analyze the VLs transmission delay using `tcpdump`.

## 3.5 VLs are multicast

.
VLs are multicast streams. To take them into account, you have to modify the P4 code and the configuration of the switch tables of the switches.

**Modifying the P4 code** Each LV is associated to a group via a new table. The action to be performed for this table is the addition of the message in the group via the `intrinsic_metadata`. The switching table will now take 2 parameters: the number of the VL instance to retransmit and the port number on which the sending must be done. Finally, it is necessary to indicate that once the transmission is done, a new analysis must be started by adding the line :

```
apply(name_of_the_commit_table)
```

The type `intrinsic_metadata` is defined by :

```
header_type intrinsic_metadata_t {
    fields {
        ingress_global_timestamp: 48;
        egress_global_timestamp: 48;
        lf_field_list: 8;
        mcast_grp: 16;
        egress_rid: 16;
        resubmit_flag: 8;
        recirculate_flag: 8;
    }
}
metadata intrinsic_metadata_t intrinsic_metadata;
```

**Modification of the switch table for Mininet** This time we will populate the table that assigns the group number to each LV. For example, the following code will add the VL 3 to group 1:

```
table_set_default commut_table _drop
table_set_default set_multicast _drop
table_add set_multicast set_output_ports 3 => 1
table_add commut_table commut 1 3 => 1
table_add commut_table commut 2 3 => 2
table_add commut_table commut 3 3 => 3
mc_node_create 1 1
mc_node_create 2 2
mc_node_create 3 3
mc_mgrp_create 1
mc_node_associate 1 0
mc_node_associate 1 1
mc_node_associate 1 2
```

The VL 3 will then be output on ports 1, 2 and 3.

**To do.** Modify the codes to simulate multicast VLs.