AFDX Project – 2022

WoPANets Extension

• Objective

The objective of this project is to extend the WoPANets tool with an external executable. The purpose of this external executable is to enable the performance evaluation of AFDX networks through computing end-to-end delay bounds of transmitted flows.

• The WoPANets tool

WoPANets purpose

In the context of critical embedded systems, verification of temporal and functional constraints in the worst-case is an essential property, not only to ensure the proper functioning of the system in its environment, but also and specially to guarantee strict certification requirements, particularly for avionics and space.

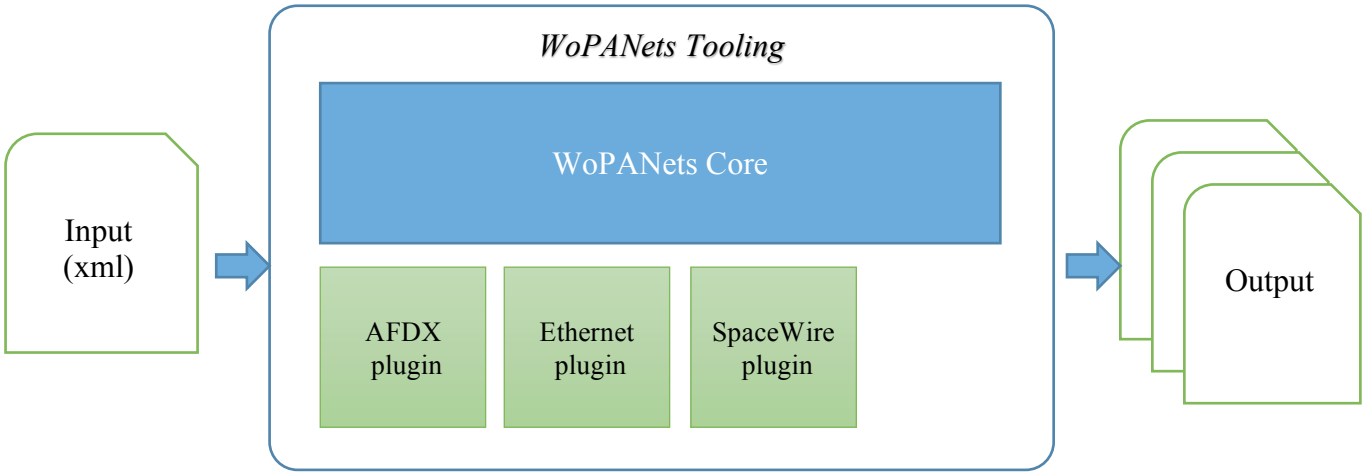
The opportunity to make this worst-case performance analysis since the early design phases will allow designers to make important decisions concerning the system parameters tuning and dimensioning, to avoid wasting time and costs in detailed and erroneous implementation.

The tool WoPANets (Worst-case Performance Analysis of Embedded Networks) provides a first answer to this design challenge. The first applications show that WoPANets reduces development time and costs, while meeting the system requirements since the early design phases.

WoPANets architecture & workflow

WoPANets is a standalone tool developed as a RCP Eclipse application. Such Eclipse RCP application is composed of plugins: some of them are part of the "Core" of WoPANets; some others are not mandatory and are extensions plugins: for instance, the AFDX technology and the related analysis are implemented inside WoPANets based on an additional plugin.

As shown in the following figure, the input of WoPANets is usually a xml file describing the network and the traffic. The output is a set of performance metrics, such as end-to-end delays, backlogs, jitters and link loads.



Example of Input :

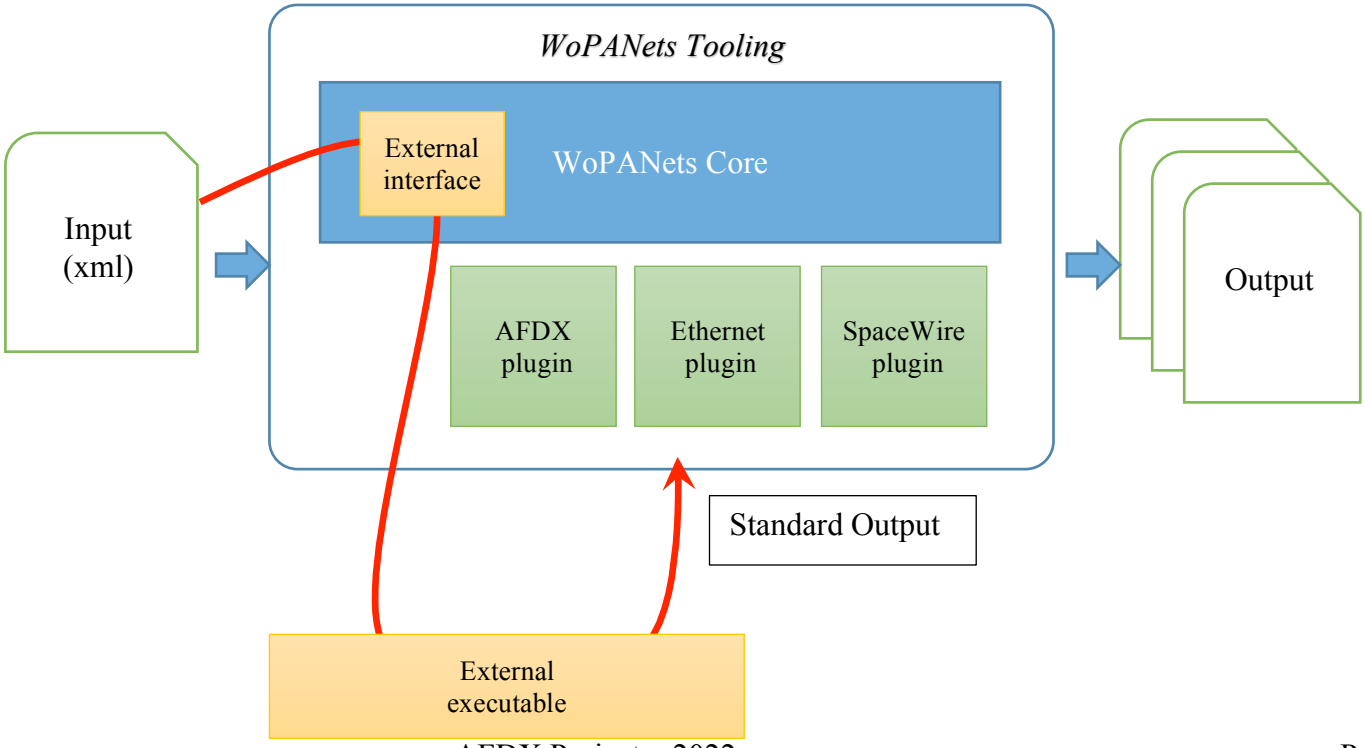
```
<?xml version="1.0" encoding="UTF-8"?>
<elements>
   <network name="Z_ESE" overhead="67" shortest-path-policy="DIJKSTRA"
  technology="AFDX" transmission-capacity="100Mbps" x-type="FULL"/>
   <station name="AFDX Station 5" service-policy="FIRST_IN_FIRST_OUT"
  transmission-capacity="100Mbps" x="96.0" y="65.0"/>
   <station name="AFDX Station 6" service-policy="FIRST_IN_FIRST_OUT"
  transmission-capacity="100Mbps" x="412.0" y="66.0"/>
   <switch name="AFDX Switch 1" service-policy="FIRST_IN_FIRST_OUT"
  tech-latency="60" transmission-capacity="100Mbps" x="252.0" y="66.0"/>
   <link from="AFDX Station 5" fromPort="0" name="AFDX Edge 3" to="AFDX Switch 1"
   toPort="0" transmission-capacity="100Mbps"/>
   <link from="AFDX Switch 1" fromPort="1" name="AFDX Edge 4" to="AFDX Station 6"
  toPort="0" transmission-capacity="100Mbps"/>
   <flow deadline="1" jitter="0" max-payload="1000" min-payload="1000"
  name="AFDX Flow 1" period="1" priority="Low" source="AFDX Station 5">
      <target name="AFDX Station 6">
         <path node="AFDX Switch 1"/>
         <path node="AFDX Station 6"/>
      </target>
   </flow>
</elements>
```

Example of Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
  <delays>
    <flow name="AFDX Flow1 ">
    <target name=" AFDX Station 6" value="45"/>
    </flow>
  </delays>
  ...
</results>
```

WoPANets extension with external executable

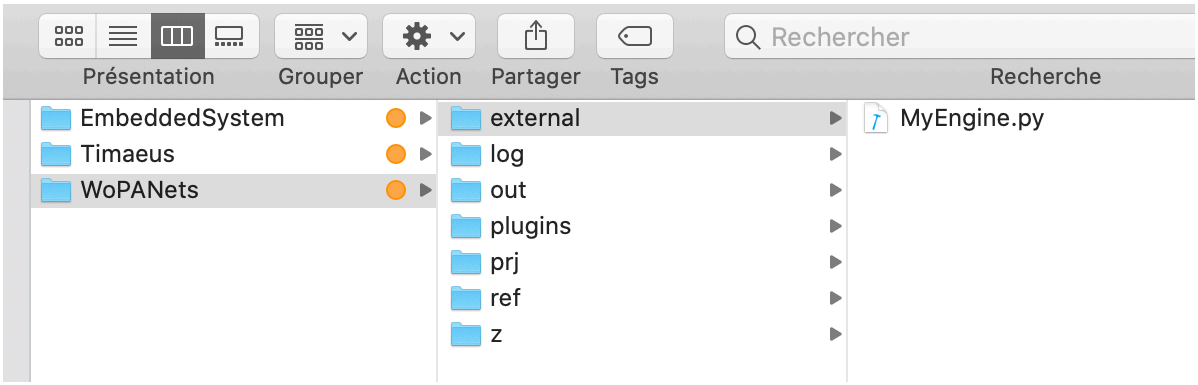
As shown in the following figure, it's also possible to extend WoPANets with external executable. Such an executable has to take as input the WoPANets input (the xml file), and produce output results (end-to-end delays, backlogs, jitters, link usage, etc.) according to the standard WoPANets output.



In our case, **external executables will be Python files**.

Extension procedure

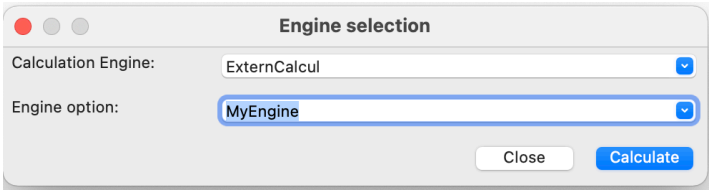
Install your Python file in the WoPANets[1] extension folder:



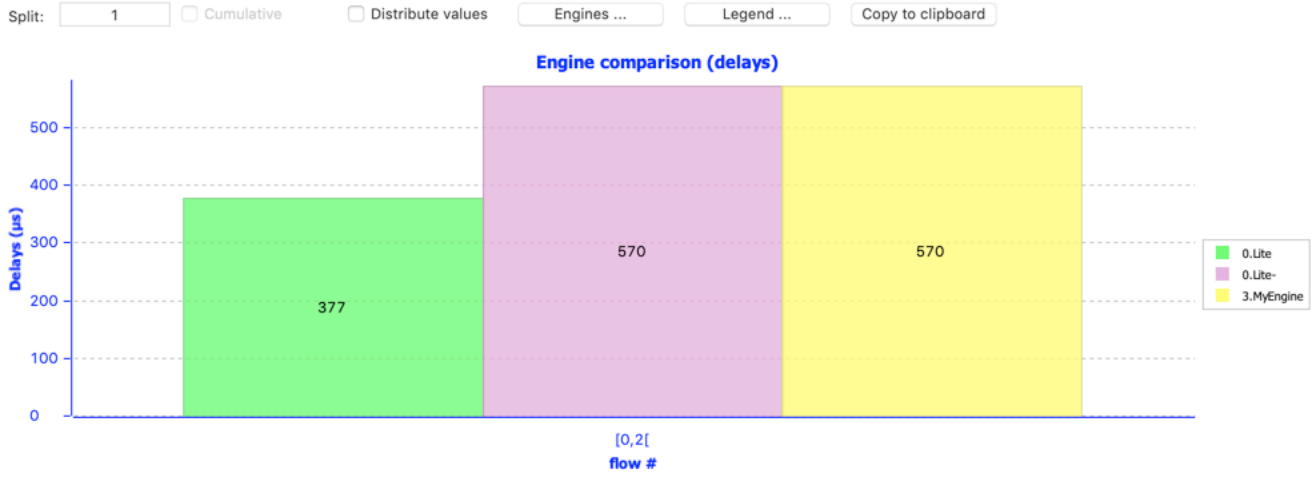
In my case, the extension file is so called “MyEngine.py”.

Inside the “Advanced Preferences”, select the path to the python3 executable.

Then inside WoPANets, click on “Custom Analysis“ ; a window opens ; select “ExternCalcul” as calculation engine, then the name of the executable as option:



The comparison of several engines is possible as follows:



[1] The WoPANets folder is located in your **personal** document folder.

• Required work

The objective is to implement **a Python program** to extend WoPANets and enable the analysis of the following metrics of an AFDX network, based on Network Calculus:

- The end-to-end delay bound for each flow to achieve each destination (multicast communication) to verify the temporal constraint (period).
- The maximum load of each link to verify the stability constraint.

We consider the following assumptions for the analysis:

- FIFO policy within the End-Systems and AFDX switches
- Null technological latency within End-Systems and Switches
- Cut-Through switching technique at the input port of switches
- Transmission capacity of 100Mbps

To create this python program, you have on LMS the following files:

- a base file (**base.py**) to help you to parse the WoPANets input file, so that you focus on the Network calculus issues.
- the **appendix "InputOutputWPN.pdf"** that details the input and output WoPANets formats to be respected by your python program
- Different **AFDX use cases** to test and validate the implemented executable.
- WoPANets download link to build a benchmark of output files of the different AFDX use cases. You can also build other use cases if needed thanks to the GUI of WoPANets.

To achieve this purpose, you need to proceed the following steps:

Step1: Design phase

- Specify the UML Classes Diagram to implement such an extension of WoPANets
- Specify the main functions to compute the load and delay bounds for multicast flows

Step2: Interpretation of the Input XML file

- Parse the input xml file to define the characteristics of the different network nodes (End-system, switches and links) and the transmitted messages. A base file **base.py** is provided on LMS to fasten this step, so that you focus on Network Calculus topics. You have to complete this file in order to fit all the projects requirements. Particularly, you need to complete the Station, Switch, Edge, Flow and Target classes.
- Compute the load of each network link and verify the stability condition of the network

Step3: Compute the end-to-end delay bounds for simple cases

- Define the corresponding affine arrival curve of each transmitted message based on leaky bucket model
- Define the corresponding service curve of each network node based on rate-latency model
- Compute the delay bound for each flow within one based on Theorem 1
- Compute the output arrival curve based on Theorem 2
- Enable the delay bound computation based on the iterative procedure using Theorems 1 and 2

Step4: **Compute the end-to-end delay bound for a representative AFDX of A380**

- Verify your implementation through a representative AFDX of A380 use case on LMS within the **AFDX use case** folder on LMS

Final Step: **WoPANets integration**

- Finalize the extension procedure of WoPANets (described above) and compare your engine with the native WoPANets engine "Native > Cute w/o shaping"