Project: Performance Analysis Tool for AFDX Networks

Objective

The objective is to implement **a Python program** that takes as input an AFDX network configuration and generates at the output the following performance metrics of such a network, based on **Network Calculus**:

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

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
- Store and Forward switching technique at the input port of switches
- Transmission capacity of 100Mbps

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

Step1: Design phase

- Read the **Appendix A** to understand the input network configurations as well as the expected metrics at the output;
- Specify the main classes for the considered configurations, and particularly the Station, Switch, Edge, Flow and Target, to implement the required computation program to obtain the expected metrics;
- Specify the main functions to compute the load and delay bounds.

Step2: Interpretation of the Input XML file, verification of the stability condition and generation of the Output 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 computation. You have to complete this file in order to guarantee the right interpretation of any input configuration and to enable the generation of the output metrics respecting the formats of the input and output xml files described in **Appendix A**.
- Compute the load of each network link and verify the stability condition of the network. Note that the links are full duplex and the load may be different on each direction.
- Generate the output xml file respecting the format described in **Appendix A** and containing the load metrics for each link in both directions.

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 any network node based on Theorem 1 when the stability condition is verified.
- Compute the output arrival curve for each flow at the output of each network node based on Theorem 2.

- Enable the delay bound computation based on the iterative procedure using Theorems 1 and 2 along a flow path.
- Test and validate your program using the given "**samples**" on LMS. There are different input configurations (XX.xml) that need to be tested (you can start from the simplest configuration to the most complex one as illustrated in file "Topologies.pdf"). You can then validate your results in comparison with the output metrics given within the samples folder (XX_res.xml).
- Update the output xml file respecting the format described in **Appendix A** and containing the end-to-end delay bounds for each flow to achieve each target.

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 "AFDX.xml" within the **"samples"** folder on LMS

Final Step: **Integration Phase using Open-Timaeus-Net**

- Finalize the integration phase of your implementation using **Open-Timaeus-Net** through comparing your computation engine with the native Open-Timaeus-Net engine "Blocks wo line shaping". The integration procedure is described in the **Appendix B** and Open-Timaeus-Net has to be downloaded on LMS. You can also build other use cases for comparison if needed thanks to the GUI of Open-Timaeus-Net.

Important:

- Your main python file shall be named <first name>_<last name>.py. For instance, "Padme_Amidala.py".
- If you have a single name, then use FNU as your first name. For instance, "Fnu_Yoda.py".