Quantitative Evaluation of Embedded Systems (IN4390)
Practical assignments 2020/2021

1 Assignment 1

Learning Objectives. At the end of this assignment you will be able to **(i)** quantitatively compare different ROS 2.0 configurations by applying extensive measurements, **(ii)** derive the confidence level of a given experiment for a certain confidence interval, and **(iii)** tell which factors have a significant impact on ROS 2.0's communication delays.

Deadline. The due date for mandatory questions is **Monday 07.12.2020 at 23:59** and the due date for optional questions is **Monday 14.12.2020 at 23:59**. These deadlines are firm. Submissions that are uploaded after the deadline will not be graded.

Team Registration. The assignments should be done in teams of two students. Register your team on Brightspace (go to "Groups").

Deliverables. Answer the following questions in a single report (including generated plots) and upload it on BrightSpace. Please do not forget to add your names, student numbers, team number and the date of submission to the file you upload. *Make sure you run all your tests for the same question on the same computer since the results may significantly vary between machines. Different questions can be answered using different machines, as long as you specify this.* Please upload any additional scripts that you created to get the plots or calculate the results in a separate zip file. Also, in the report, include the names of the tools that helped you with analysing the measurements.

Assessment Instructions. You must answer all questions. The grade is on a scale from 0-10. You will *pass* this mandatory assignment only if you gain at least 6 points. If answers are incorrect, you will receive feedback from TA's and you will have one more chance to submit the assignment.

1.1 Preparation

Step 1: Read the paper. The assignment is mainly based on the following paper, published in the EmSoft conference in 2016: "Exploring the Performance of ROS2" (https://ieeexplore.ieee.org/document/7743223). Read the paper carefully.

Step 2: Learn about ROS. If you are interested in learning more about ROS 2.0 before starting this lab, here is a link containing basic ROS 2.0 tutorials and examples: https://index.ros.org/doc/ros2/Tutorials/

A particularly useful tutorial can be found in the following link. It describes in detail how to build packages with colcon and ends with a nice example of how to launch a talker and listener node. https://index.ros.org/doc/ros2/Tutorials/Colcon-Tutorial/

If you already know ROS 1.0 and wondering why we decided to go for ROS 2.0, here is a nice slide/video that explains all differences: Slides at https://www.osrfoundation.org/wordpress2/wp-content/uploads/2015/04/ROSCON-2014-Why-you-want-to-use-ROS-2.pdf and video at https://vimeo.com/107531013

Step 3: Application model. The assignment focuses on evaluating the end-to-end latency of publish-subscribe communication in ROS using a simple application model with a single talker and listener node that communicate through a topic *chatter*. The application's code is provided for you in "Step 4". Figure 1 illustrates the communication between the nodes. The "talker" node iterates through a set of message sizes (256b-4Mb) and publishes each of them multiple times (120 times by default) on the chatter topic while timestamping each publication. The "listener" node receives the data and records a timestamp. At the end of the evaluation, these timestamps are stored in the `evaluation` folder in `publish_times\publish_time_XKbyte.txt` and `subscribe_times\subscribe_time_XKbyte.txt` respectively, where X corresponds to the data size. The difference between the publish and subscribe times are referred to as transport time, are calculated through the `calculate` executable and are stored in `transport_times\transport_time_XKbyte.txt`

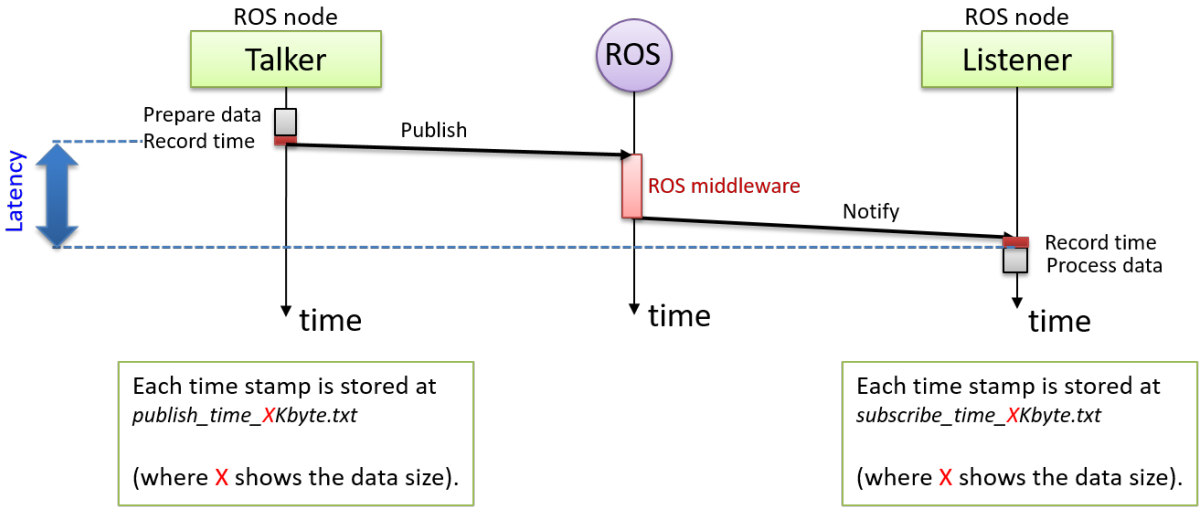

Figure 1: Application model and measurements.

Step 4: Install Ubuntu In order to run this assignment's code, you will need a computer equipped with Ubuntu (preferably 18.04). If Windows is currently installed on your computer, the preferred option is to set up your computer to dual-boot Ubuntu and Windows. If you are unable to dual-boot, please skip to section 1.1.2 and follow the instructions there, as all of the required packages have already been installed for you. Dual-booting is also preferred for Mac users, although this is more tricky.

1.1.1 Dual-Boot setup

1. Start by making a full backup of your hard drive, in case things go wrong during installation. If you are unable to make a backup, please do not continue and skip to the Live USB setup in section 1.1.2.
2. Create an Ubuntu 18.04 Live USB using these instructions:
   Windows: `https://ubuntu.com/tutorials/create-a-usb-stick-on-windows`
   Mac: `https://ubuntu.com/tutorials/create-a-usb-stick-on-macos`
3. Follow these instructions to install Ubuntu:
   Window: `https://help.ubuntu.com/community/WindowsDualBoot`
   Mac: `https://help.ubuntu.com/community/MactelSupportTeam/AppleIntelInstallation#Dual-Boot_Mac_OSX_and_Ubuntu`

   You may need to disable secure boot in BIOS to be able to boot from the USB.

1.1.2 Live USB setup

IMPORTANT NOTE - Rufus is unable to create an Ubuntu 18.04 persistent USB. This means that any new data **will be removed from the USB every time you power off the machine**. To create a persistent Live USB, please approach the TAs

during a lab, or ask someone who already has Ubuntu installed to download *mkusb* and create a persistent USB using the same .iso which is mentioned below

IMPORTANT NOTE 2- These instructions are for Windows users only

1. Obtain an empty USB 3.0 flash drive, at least 8GB. All items on the flash drive will be wiped, so make sure there is nothing important on there.
2. Download the live-ros.iso using the link here `https://filesender.surf.nl/?s=download&token=9d92718e-5c9b-4b87-88f5-eec21fdec061`
3. For Windows: download Rufus from here `https://rufus.ie/`
4. Create a Live USB using the following settings (you can also try MBR if GPT does not work):

```
Boot selection: live-ros.iso
Partition Scheme: GPT
Target System: UEFI (non CM)
```

5. Boot into the USB. You will probably need to enter you BIOS settings and modify the boot order so the USB is selected first. You may also need to disable secure boot.
6. Select the "Use Ubuntu without installing" option for Live USB (without persistency). If you are using a persistent USB, you can select the "Run Ubuntu - persistent live". If you have enough RAM (16GB) you can select the "... to RAM" option as this will improve the performance once the Ubuntu session has been copied onto RAM.

Step 5: Download the assignment's contents. In order to redo some of the experiments done in the paper, we have prepared a package for you, which you can download from the course website. If you are using the ROS2 Live USB, the assignment files have already been downloaded for you to the `/assignments` folder. An overview of the workspace structure is presented in Figure 2. Within the workspace package, the `src` directory contains the source code (written in C++) which is used to build the ROS2 nodes:

1. *interprocess_eval*: Package containing two nodes: `listener` and `talker`. The talker node will by default send (publish) a set of messages of varying size to the listener node. Those nodes also save the timestamps of when the message was sent and received. This is the main communication method which will be used in Questions 1-4.
2. *artificial_load*: Package containing one simple node that puts artificial stress on the CPU by repeatedly performing a computationally intensive task. This node will be useful for Question 3.
3. *interprocess_remote_eval*: Similarly to `interprocess_eval`, this package also contains a `listener` and `talker` node. The main difference is that in this scenario the talker and listener communicate between two different machines over the network. Furthermore, the listener establishes a socket connection with the talker in order to confirm the message was received. This is used as part of the optional questions and requires some setup.
4. *intraprocess_eval*: Package containing a `chat_intraprocess` executable that internally initialises two nodes, a listener and a talker and enables an intra-process communication between them. This package is not used in any of the compulsory or optional questions. Nevertheless, feel free to experiment with it and see what kind of results you can obtain.

The `evaluation` directory contains the different sized messages to be sent and collects the evaluation results in the `publish_time` and `subscribe_time` folders after each experiment. Finally, in the main directory, you will find components that are required to answer the questions. With the help of this package, you will be able to test a variety of configurations:

1. *inter-process or intra-process communication*: in the first case, nodes act as separate processes in the system while in the second case, all nodes are a part of one process. This is controlled by running nodes from the `interprocess_eval` and `intraprocess_eval` package respectively.
2. *Message size*: the setup allows you to run the experiment for a number of data sizes (varying from 256B to 4MB). By default, the `talker` node performs 120 tests for each message size.

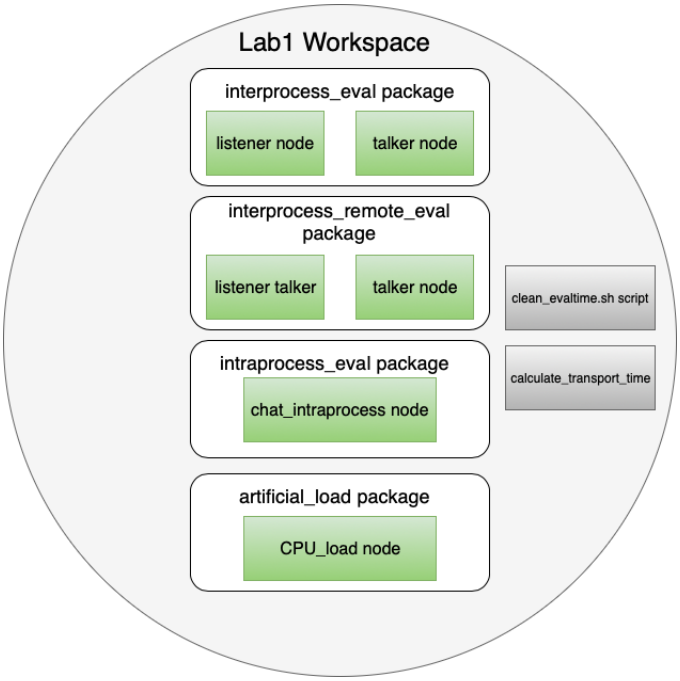

Figure 2: Workspace structure

3. *DDS choice*: The setup performs compilation using all distribution services (DDS) available on your system. Most likely the three common ones, i.e., FastRTPS, OpenSplice, and Connext. Later, by following the setup of the lab, you will see that a different executable is generated for each for those services. This parameter is, once again, controlled by running the corresponding executable. For example, if you want to use the `fastrtps` while measuring the inter-process communication overhead, use the following commands:

```
$ ros2 run interprocess_eval listener_interprocess__rmw_fastrtps_cpp
$ ros2 run interprocess_eval talker_interprocess__rmw_fastrtps_cpp
```

4. *Quality of service parameters*: you can pass a wide set of parameters to the DDS that best fit your use case, the three default configurations you can test were named *reliable*, *best-effort*, and *history*. You can check the lab source implementation details and ROS 2.0 documentation (`https://design.ros2.org/articles/qos.html`) to learn more about the impact of each setup on the communication. This parameter is defined at the beginning of the `talker.cpp` and `listener.cpp` source code and can be modified through changing the `#define QoS_Policy` and rebuilding the code (you will learn how to rebuild a ROS2 workspace in the lab environment setup section).

5. *Presence of artificial CPU load*: running the CPU_load node from the artificial_load package will cause high CPU load that might affect the communication. The source code of this node is located in `src/artificial_load/CPU_load.cpp` and can be executed like any other ROS node.

6. *Message publish frequency*: This parameter is defined at the beginning of the `talker` and `listener` .cpp source code and can be modified through changing the `#define PUBLISH_Hz` and rebuilding the code (you will learn how to rebuild a ROS2 workspace in the lab environment setup section).

1.2 Setting Up the Lab Environment

If you running Ubuntu on the Live Persistent USB, please skip ahead to the CPU frequency scaling in section 1.2.2.

1.2.1 Installing ROS 2.0 via Debian Packages

The following link provides a set of instructions on how to install ROS2-Dashing through Linux debian packages. The essential elements of this tutorial are listed below. For the purpose of this lab you must use ROS2 Dashing Diademata release on a computer equipped with Ubuntu (prefferably 18.04).
https://index.ros.org//doc/ros2/Installation/Dashing/Linux-Install-Debians/

Setup Locale:

```
$ sudo locale-gen en_US en_US.UTF-8
$ sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
$ export LANG=en_US.UTF-8
```

Setup Sources:

```
$ sudo sudo apt update && sudo apt install curl gnupg2 lsb-release
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc
  | sudo apt-key add -
$ sudo sh -c 'echo "deb [arch=amd64,arm64] http://packages.ros.org/ros2
  /ubuntu `lsb_release -cs` main" > /etc/apt/sources.list.d/ros2-latest.list'
```

Setup Install ROS 2.0 packages:

```
$ export CHOOSE_ROS_DISTRO=dashing
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install ros-$CHOOSE_ROS_DISTRO-desktop
$ sudo apt install ros-$CHOOSE_ROS_DISTRO-ros-base
```

Setup Environment:

```
$ sudo apt install python3-argcomplete
```

Sourcing the setup script:

```
$ source /opt/ros/$CHOOSE_ROS_DISTRO/setup.bash
```

(Optional) Add this line to your .bashrc so you dont have to source every time

```
$ echo "source /opt/ros/$CHOOSE_ROS_DISTRO/setup.bash" >> ~/.bashrc
```

Install additional RMW implementations (Opensplice and RTI connext):

```
$ sudo apt update
$ sudo apt install ros-$CHOOSE_ROS_DISTRO-rmw-opensplice-cpp
$ sudo apt install ros-$CHOOSE_ROS_DISTRO-rmw-connext-cpp
```

1.2.2 Environment Setup

Real Time Priority settings Now that ROS 2.0 is installed and configured, you need to adjust the maximum real-time priority allowed for non-privileged processes (Linux 2.6.12 and higher) and the memory which the application can lock for its own use. This is done by editing the /etc/security/limits.conf file (root access required). This step is not required if you are using the Live USB:

```
$ /etc/security/limits.conf
```

and adding the following two lines:

```
username - rtprio 98
username - memlock -1
```

where `username` should be replaced with your username. After this part is finished, the computer **needs to be restarted**. Alternatively, you can run the assignment as the root user using the `sudo su -` command.

CPU frequency scaling settings To avoid biased results, make sure that your system has CPU frequency scaling disabled. Follow those simple steps to ensure your processor is running at the same speed during the measurements. **TODO**: add support for intel_pstate

Install cpufrequtils:

```
$ sudo apt-get install cpufrequtils
```

Edit the following file:

```
$ sudo nano /etc/default/cpufrequtils
```

by adding the following content (adjust the frequencies to your CPU):

```
GOVERNOR="performance"
MIN_SPEED="2100MHz"
MAX_SPEED="2100MHz"
```

Restart the cpufrequtils daemon:

```
$ sudo /etc/init.d/cpufrequtils restart
```

Check if the settings were correctly applied:

```
$ cpufreq-info
```

After finishing the experiments you should enable the CPU-frequency scaling back to avoid undesired system behavior (for example increased power consumption). Restarting your computer should revert the changes to the original ones you had on your system (most likely an `ondemand` policy). You can also run

```
$  service cpufrequtils stop
```

1.2.3 Building the Project

In order to be able to build ROS2 packages you first need to install colcon. More information can be found in the following address:
https://colcon.readthedocs.io/en/released/user/installation.html

```
$ sudo sh -c 'echo "deb [arch=amd64,arm64] http://repo.ros2.org/ubuntu
  /main `lsb_release -cs` main" > /etc/apt/sources.list.d/ros2-latest.list'
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc
  | sudo apt-key add -
$ sudo apt update
$ sudo apt install python3-colcon-common-extensions
```

Source ROS2 (you need to execute this command every time you open a new Terminal window, unless you add it to the .bashrc):

```
source /opt/ros/dashing/setup.bash
```

Now **download the project package** from the course page.
To build the executables, navigate to the laboratory workspace directory. Next, run the following command:

```
$ colcon build --symlink-install
```

Finally, compile the .cpp script which is used to calculate the transit times at the end of an experiment:

```
$ g++ calculation_transport_time.cpp -o calculate
```

1.2.4 Running an Example Experiment

Now that everything has been set, you should be able to run a basic experiment. Open two different terminals and navigate to the main project directory on both. Next, setup the install.bash of project on both terminals:

```
$ source ./install/setup.bash
```

On the first terminal run the following command:

```
$ ros2 run interprocess_eval listener_interprocess
```

On the second terminal run the following command:

```
$ ros2 run interprocess_eval talker_interprocess
```

This will initiate two ROS nodes that are periodically sending messages of varying size from the talker to the listener (from 256B, up to 4MB). After the evaluation finishes executing you can find the publish and subscribe times in `./evaluation/publish_time` and `./evaluation/subscribe_time` respectively.

Running the `./calculate` executable that you generated earlier should calculate the transport time of the messages (results stored in `./evaluation/transport_time`).
By running the `clean_evaltime.bash` script, you can clean the aforementioned directories (Be careful, this will remove all results from the latest experiment).

2 Mandatory Questions

2.1 Question 1 (2 points)

The goal of this question is to derive a box-plot diagram [1] that shows the latency of ROS communication between the talker and listener nodes (in the provided benchmark) as a function of data size (consider all available message sizes). For this experiment, nodes must communicate with each other using **inter-process** communication and the DDS must be set to **FastRTPS** with its default configurations.

Instructions: For this question, you should start the `listener` and `talker` nodes:

```
$ ros2 run interprocess_eval listener_interprocess__rmw_fastrtps_cpp
$ ros2 run interprocess_eval talker_interprocess__rmw_fastrtps_cpp
```

These commands must be executed in separate terminal windows on the same machine to obtain inter-process communication measurements for the default configuration that utilises FastRTPS as the underlying DDS.

In order to obtain a set of measurements for the aforementioned setup, you can run the `calculate` executable (should have been generated as a part of the lab environment setup). This will calculate the transport times by subtracting the subscribe times from their corresponding publish times and generate its outputs in the `evaluations` folder under the `transport_time` sub-folder.

Notes: The outputs are expected to have a similar trend as the one shown in Figure 14 of the paper "Exploring the Performance of ROS2".

Please note that you need to take a lot of measurements in order to have a smooth trend. By default, the `talker` node repeats the measurement for each data size 120 times. To achieve an even more accurate result you can change the `#define EVAL_NUM` parameter in the `talker` and `listener` source code to a larger value. Remember to always clean your `evaluation` folder before running the next experiment using the `clean_evaltime.bash` script which deletes all data from the publish, subscribe and transport time folders.

Deliverables:

- Explain what you have done, e.g. how many times you have repeated your experiment.
- Draw the box plot diagram and interpret the results, e.g. say why there is a certain trend.

Your answer should not be longer than one page (including plots).

2.2 Question 2 (2 points)

The goal of this experiment is to learn how to calculate the confidence interval of a set of data for a given confidence level. To learn how to calculate those, have a look at https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/confidence-interval/

Instructions: For this experiment, you must use the data provided in the following file: `question2_transport_time.txt`, which represents a set of *transport times*. For these data, calculate the confidence intervals for an 80% and 98% confidence levels.

Deliverables:

- Present the results and calculations in the report, together with a brief explanation of the obtained results.

[1] Check wiki for the definition of a box-plot diagram. Here, we are interested in *minimum, first quartile, median, third quartile, and maximum* values.

2.3 Question 3 (3 points)

The goal of this experiment is to evaluate the impact of background workload on the latency of ROS 2.0's communication between the nodes under two setups: (i) nodes use real-time priorities and (ii) nodes do not have real-time priorities. These two setups show how the node's communication delay is affected by the underlying scheduling algorithms and its parameters. Linux prioritizes the execution of a process or thread that is assigned to a real-time priority (from 1 to 99 with 99 being the highest priority) over other processes, namely, if a non-real-time process is running at the time a real-time process is activated, then the non-real-time process will be preempted and the CPU will be given to the real-time process. To learn more about real-time priority in Linux, have a look here: https://www.kernel.org/doc/html/latest/scheduler/index.html

Instructions: The `artificial_load` package which is a part of the lab1 workspace will enable you to easily cause background workload that are highly CPU demanding. By default this node will run infinitely. You can stop it manually at any time using `Ctrl + Z`. Make sure to stop that node after your experiment.

In order to run the artificial CPU load node, in a separate terminal window, run the following ROS node. It will act as a simulation of a real scenario where your system is executing other computationally heavy tasks except of your ROS2 application.

```
$ ros2 run artificial_load CPU_load
```

For this task, make sure you disabled the CPU frequency scaling by following the environment setup instructions. Follow the instructions from section 2.6 to make sure this option is disabled in your system.

Now repeat the steps from Question 1. First, without running the artificial CPU load node (you can either run those two separate terminals or add `&` at the end of each command). Make sure the `#define QoS_Policy` in the `talker` and `listener` implementation files is set to default `3` (`history`).

```
$ ros2 run interprocess_eval listener_interprocess__rmw_fastrtps_cpp
$ ros2 run interprocess_eval talker_interprocess__rmw_fastrtps_cpp
```

The second time, before running the listener and talker nodes, start the artificial CPU load node (you might want to run it in a separate terminal, it will be easier to kill it later on).

```
$ ros2 run artificial_load CPU_load
```

Compare the results for different message sizes performed with and without the artificial CPU load node running.

By default, the `listener` and `talker` nodes set their priority to a 98 real-time priority. The next step is to repeat the whole experiment with the two nodes running on non-real-time priority. In order to force this, delete `#define RUN_REAL_TIME` from the `talker_interprocess.cpp` and `listener_interprocess.cpp` files and rebuild the workspace.

```
$ colcon build --symlink-install
```

At the end, you should have obtained 4 different data sets **(i)** real-time priority without extra CPU load **(ii)** real-time priority with extra CPU load **(iii)** non-real-time priority without extra CPU load **(iv)** non-real-time priority with extra CPU load.

Deliverables: Draw the **box plots** and **histograms** of all four types of output. You do not need to show plots for all message sizes. Instead, choose 3 message sizes which you think highlight the most relevant features of the sample distribution. Explain the trends.

2.4 Question 4 (3 points)

This question allows you to compare the impact of different DDS configurations on the end-to-end latency. The question has four steps:

1. While keeping CPU frequency disabled and QoS set to 3 (history mode), obtain at least 120 latency samples for each of the following six configurations:
   (a) **FastRTPS-256B:** DDS = FastRTPS, message size = 256Byte
   (b) **OpenSplice-256B:** DDS = OpenSplice, message size = 256Byte
   (c) **Connext-256B:** DDS = Connext, message size = 256Byte

(d) **FastRTPS-128KB:** DDS = FastRTPS, message size = 128KByte

(e) **OpenSplice-128KB:** DDS = OpenSplice, message size = 128KByte

(f) **Connext-128KB:** DDS = Connext, message size = 128KByte

2. Draw the histogram of each of the six data sets.
3. Use ANOVA to explain whether or not the choice of DDS has a significant impact on the latency.
4. Discuss the results (explain what you obtained from ANOVA). If you are going to suggest one of these DDSes to a user, which one do you suggest and why? Briefly explain your answer.

3 Assignment 1: Optional Parts

Deadline. The due date for optional questions is on **Monday 14.12.2020 at 23:59**. The deadline is firm. Submissions that are uploaded after the deadline will not be graded.

Deliverables. Upload a single report for all of the optional questions that you would like to answer on BrightSpace. Please do not forget to add your names, student numbers, and the date of submission to the file you upload.

Assessment Instruction. If your grade for an optional question is *below* 50% of the points of that question, you will not receive any point for that question. For example, if a question has 20 points and you scored 9, then you will not receive any point for that question. Consult a TA if you are not sure if your answer will qualify for grading.

3.1 Question 5 (10 points)

The goal of this question is to show you a possible impact of uncontrolled variables when doing an experiment. As mentioned in Lecture 2, when repeating an experiment, (ideally) you need to have control over all influential variables that may impact your measurements. One of those variables can be the CPU-frequency scaling option that might be active on some laptops (in particular, when they run on battery). Look at your power management setup and see if your setup allows such an option. For the goal of this experiment, you need to activate one of the smart CPU-frequency scaling options on your machine. After completing this question, don't forget to switch the CPU-frequency scaling off if you want to complete the remaining questions.

Instructions: Turn on the CPU-frequency scaling of your machine. Redo all measurements of Question 3. Search for any significant difference between the diagrams. In particular, compare the results you get for "small and large message sizes" under "heavy artificial load" and with "real-time priorities".

Deliverables: Plot the diagrams and explain why you see a difference between the outputs in the presence of CPU-frequency scaling even when node run with real-time priorities.

3.2 Question 6 (30 points)

The `interprocess_remote_eval` package should enable you to test the communication overhead between two different machines within the same network. The `talker` node from this package should be executed on one computer while the `listener` on the other one. Find the correct configuration to establish connection between those two ROS2 nodes (IP values, etc.). *The eduroam network won't allow you to perform this test.* Once you are able to get the measurements, repeat the test while simulating some extra network activity (for example by downloading a file on a third machine that is connected to the same network).

Deliverables:

- Explain what you did and draw the box plot of the latency in all scenarios.
- Draw a curve that shows the average latency of each of two scenarios (no extra load on the network, additional extra load on the network) together with the 80% confidence interval of each point on the curve.
- Interpret your results.

3.3 Question 7 (40 points)

The goal of this question is to observe how different interactions of multiple nodes affect the communication delay and overhead (on a single machine). So far only one-to-one communication between nodes was addressed, but in a real case scenario you would have several nodes communicating with one another through common topics (think of a sensor distributing data to multiple controllers). This added complexity can potentially affect the communication overhead. Note that for this assignment you will have to expand the given lab code.

Instructions: Refactor the given code so that you have a scenario where several "listeners" are subscribed to the same topic where a "talker" node is publishing like the one shown in Figure 3. Repeat the measurements from Question 1 while varying the number of listeners (from 1 to 10 listeners) subscribed to the topic. Compare your results with those from Question 1 as well as amongst the "listeners" themselves.

Deliverables: Plot the diagrams and explain if you see a difference between the outputs in the presence of multiple listeners. Explain how the communication delay and overhead are affected.

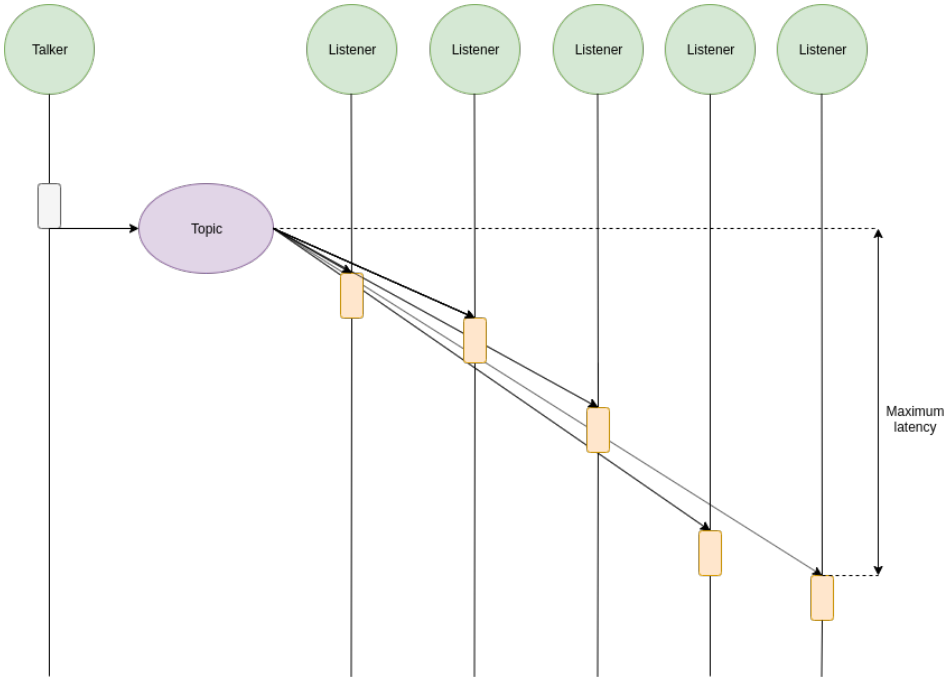

Figure 3: Application model for multiple listeners

3.4 Question 8 (40 points)

The goal of this question is to observe how different interactions of nodes affect the communication delay and overhead. So far only one-way communication was addressed (ping), but in several cases there might need to be two-way communication established between the nodes (ping-pong). One example of this would be a consumer node returning an acknowledgement to guarantee data integrity. Furthermore, information may propagate through several stages in the case of a processing chain (i.e Image processing pipeline). Both of those scenarios significantly contribute to the delay of communication between nodes. Note that for this assignment you will have to expand upon the given lab code.

Instructions: Refactor the given code so that you create ping-pong communication between a set of two nodes. Repeat the measurements from Question 1 but considering the round-trip times. Compare the results with the measurements taken from Question 1 for the "ping" part and the round-trip time. Additionally, create a chain of five nodes that propagate a message $(A \rightarrow B \rightarrow C \rightarrow D \rightarrow E)$ and measure the end-to-end latency between the producer node (A) and the final consumer node (E) as well as the latency between individual steps.

Deliverables: Plot the diagrams and explain your observations. Elaborate on how the end-to-end latency is affected in each case.