

Тестирование задержек при передаче сообщений в различных системах по шаблону издатель-подписчик.

В работе исследовались задержки в нескольких сервисах передачи сообщений. Задержки исследовались в зависимости от длины и количества сообщений с использованием следующих сервисов:

- 1) DDS(data distribution service) с помощью ROS2: OpenSplice, FastRTPS, RTI Connext, Cyclonedds;
- 2) Сервис ZeroMQ.

При передаче сообщений без перерывов возникает очередь на принятие и обработку этих сообщений соответствующим узлом — подписчиком. Исследовалось только время на передачу сообщения, так как подписчик никак не обрабатывал полученный пакет, а только принимал и записывал время, поэтому время на обработку сведено к минимуму. Процессы узлов(подписчика и издателя) привязываются к определенным процессорам с эксклюзивным использованием, что означает, что эти процессы монопольно используют ресурсы соответствующих процессоров и другие процессы их не прерывают.

Данный документ описывает последовательность действий для воспроизведения тестовых сцен и приводит результаты этих тестов в виде графиков, которые иллюстрируют увеличение времени передачи пакетов.

В полученных графиках показано, как с увеличением количества сообщений при маленьких(50 символов) и больших (500 000 и 60 000 символов для opensplice и 60 000 для остальных) сообщениях увеличивается время на передачу пакетов узлу и какое время занимает передача основной части этих сообщений.

1. Описание тестовых сценариев.

Были написаны 2 тестовых сценария с 2 узлами — издатель(publisher), подписчик(subscriber):

1. Издатель отправляет без задержки 5 000 (символьных) сообщений длиной 50 символов.
2. Издатель отправляет без задержки 5 000 (символьных) сообщений длиной 500 000 символов(для opensplice) или 60 000 символов(для всех).

Издатель записывает в файл publisher.txt время отправки в наносекундах при каждой отправке сообщения.

Подписчик принимает сообщение и записывает время его получения в массив. Когда он получает 5 000 заданных сообщений он завершает свою работу и переносит все данные из массива в файл subscriber.txt.

В тестах используются стандартные конфигурации DDS и выставлены следующие политики QoS:

- 1) History: Keep All
- 2) Reliability: Reliable
- 3) Durability: Transient local

В конфигурации ZeroMQ для передачи сообщений использовалось tcp соединение, максимальный размер очереди 6000.

2. Подготовка Linux для тестирования.

Ссылка на репозиторий с проектом:
<https://github.com/OSLL/ROS2-DDS-Testing/tree/master>

Для успешного запуска тестов необходима Ubuntu 18.04.

Для сборки и конфигурации ядра необходимо запустить скрипт *linux-rt-install.bash*, который находится в корневой папке проекта.

Скрипт скачает версию linux-4.4.12 и соответствующий rt патч для него, затем разархивирует и пропатчит ядро. В процессе будет вызвана утилита для конфигурации ядра, настроить ядро необходимо вручную. При конфигурации необходимо выбрать параметр “Fully Preemptible Kernel” (CONFIG_PREEMPT_RT_FULL). Данную опцию можно найти в General Setup -> Preemption Model. Затем будет начата сборка и установка ядра, которая может занять продолжительное время.

3. Установка ROS2 и необходимых DDS.

3.1. Установка ROS2.

Для установки ROS2 необходимо выполнить скрипт *ros2_install.bash*, который находится в корневой папке проекта. Для тестов необходим дистрибутив ROS2 dashing, который поддерживается только версией Ubuntu 18.04, именно его устанавливает данный скрипт.

3.2. Установка DDS.

Чтобы установить используемые в тестах DDS, необходимо выполнить следующие команды:

```
sudo apt update
```

```
sudo apt install ros-dashing-rmw-opensplice-cpp
sudo apt install ros-dashing-rmw-connnext-cpp
sudo apt install ros-dashing-rmw-cyclonedds-cpp
```

FastRTPS установлен по умолчанию при установке ROS2.

3.3. Установка ZeroMQ.

Для установки используйте команду:

```
sudo apt install libzmq3-dev
```

4. Подготовка среды для тестирования.

Чтобы создать cpuset'ы для процессов, используется скрипт `create_cpuset.sh`, который необходимо запускать от пользователя `root`, лежащий в корне проекта. Он создает два cpuset'а и резервирует под каждый один процессор, устанавливая для них эксклюзивное использование. Ноды(узлы) при запуске записывают свои `pid` в cpuset'ы, таким образом привязываясь к процессору и выставляют себе повышенный приоритет.

Проект также необходимо собрать и настроить среду.

Все описанные в этом пункте действия записаны в скрипт `env_prep.bash`.

В итоге необходимо выполнить 2 команды:

- 1) Перейдем в режим суперпользователя: `sudo su`
- 2) Запустим скрипт: `./env_prep.bash`

5. Тестирование.

Оба узла при запуске добавляют свой pid в соответствующую cgroup и выставляют себе повышенный приоритет, поэтому запуск тестов необходимо осуществлять от имени root пользователя(команда sudo su)!

Запуск тестов осуществляется из директории test_delays для тестирования DDS и в директории ZMQTesting/build для тестирования ZeroMQ. Файлы с результатами будут созданы в этих же директориях.

5.1. Запуск тестов в ROS2:

После получения последнего сообщения подписчик выведет соответствующее сообщение, после этого можно завершать работу издателя с помощью комбинации клавиш: Ctrl+C.

Замена используемой DDS выполняется заменой параметра:

```
RMW_IMPLEMENTATION=rmw_fastrtps_cpp
RMW_IMPLEMENTATION=rmw_opensplice_cpp
RMW_IMPLEMENTATION=rmw_connext_cpp
RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
```

Запуск 1 сценария теста:

```
RMW_IMPLEMENTATION=rmw_fastrtps_cpp ros2 launch
test_sub_and_pub run_test1_3.launch.py message_number:=5000
message_length:=50
```

Запуск 2 сценария теста:

```
RMW_IMPLEMENTATION=rmw_fastrtps_cpp ros2 launch
test_sub_and_pub run_test1_3.launch.py message_number:=5000
message_length:=60000
```

Дополнительный тест для opensplice:

```
RMW_IMPLEMENTATION=rmw_opensplice_cpp ros2 launch  
test_sub_and_pub run_test1_3.launch.py message_number:=5000  
message_length:=500000
```

Чтобы сохранить результаты тестирования DDS надо сохранить файлы после 1 теста, так как издатель перезаписывает файл publisher.txt, а подписчик дописывает их в конец файла subscriber.txt.

5.2. Запуск тестов для ZeroMQ:

В случае корректного завершения программы ничего не выводят, иначе будет выведена ошибка. Все действия необходимо делать из директории ZMQTesting/build в режиме суперпользователя (команда sudo su).

Запуск 1 сценария теста:

```
./sub 50
```

```
./pub 50
```

Запуск 1 сценария теста:

```
./sub 60000
```

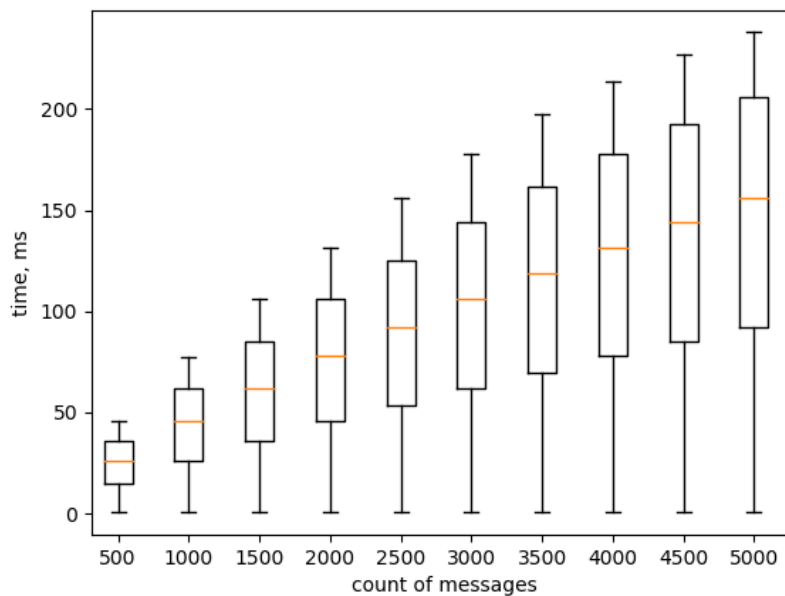
```
./pub 60000
```

6. Результаты.

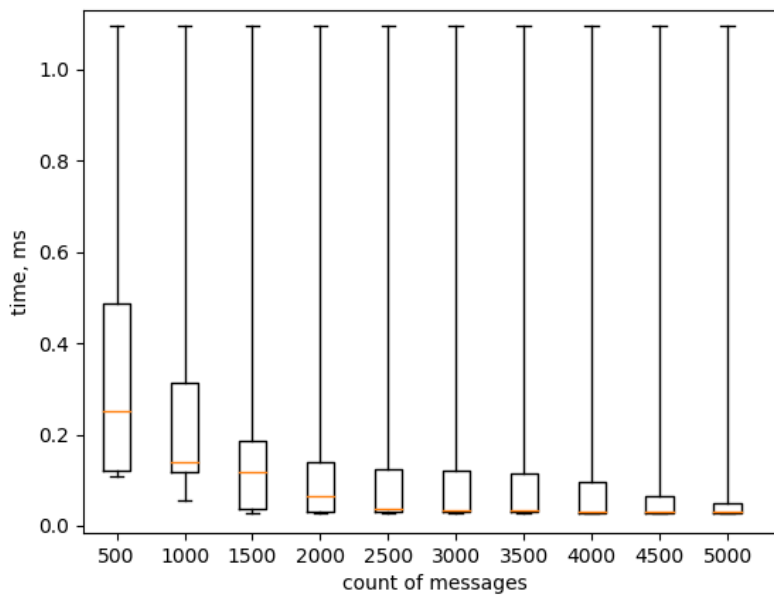
Разность соответствующих значений в файлах является задержкой, по этой задержке мы и строим следующие графики:

1 сценарий:

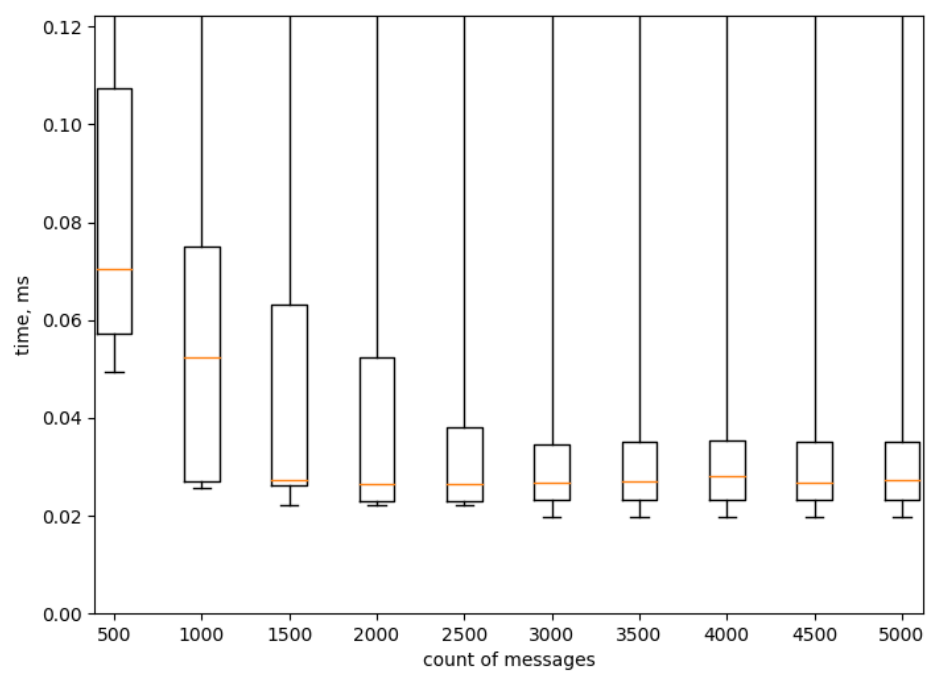
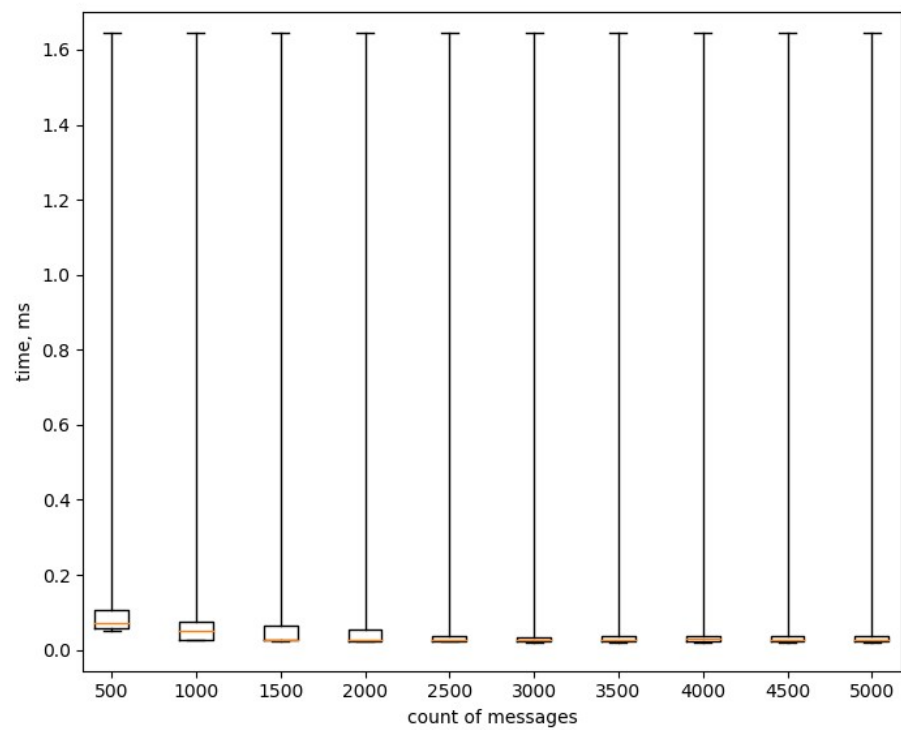
OpenSplice:



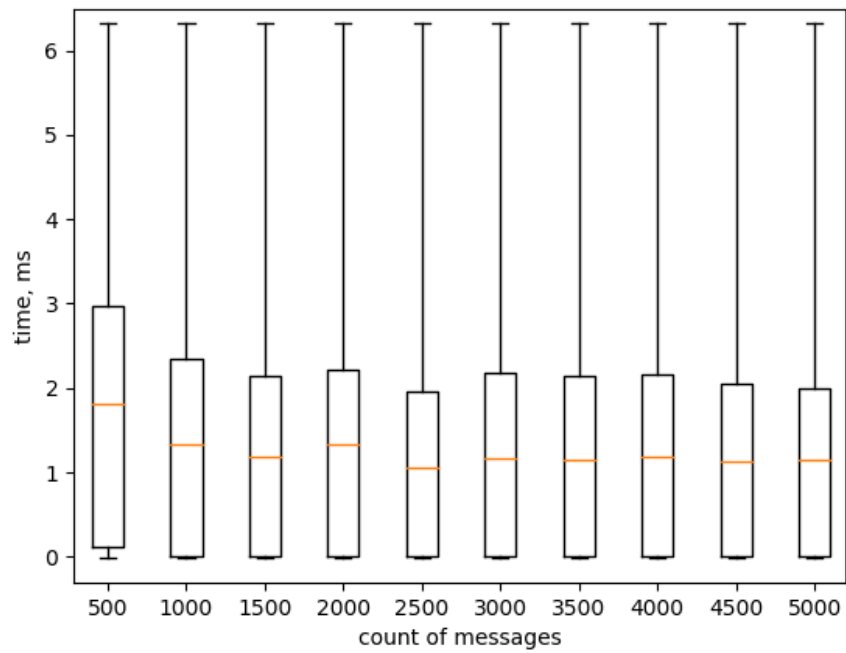
FastRTPS:



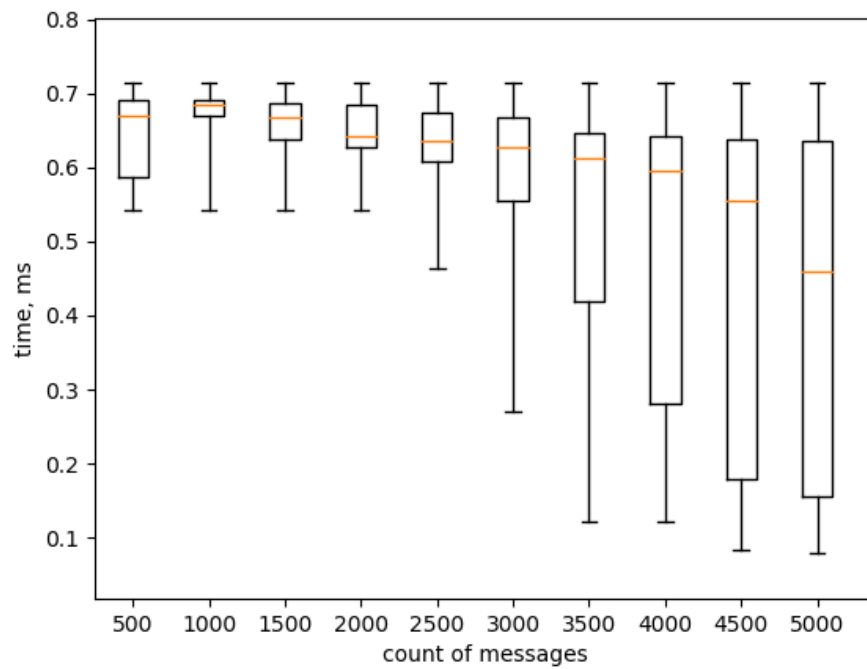
RTI Connex:



Cyclonedds:

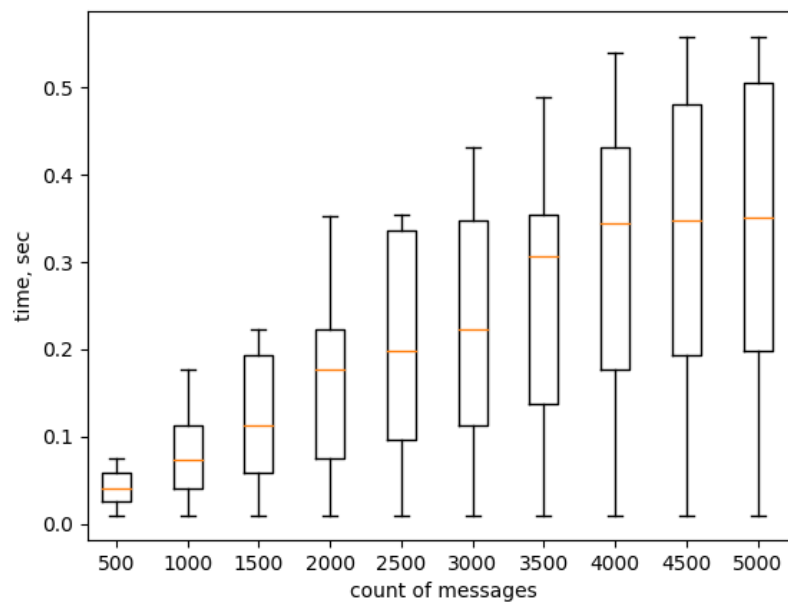


ZeroMQ:

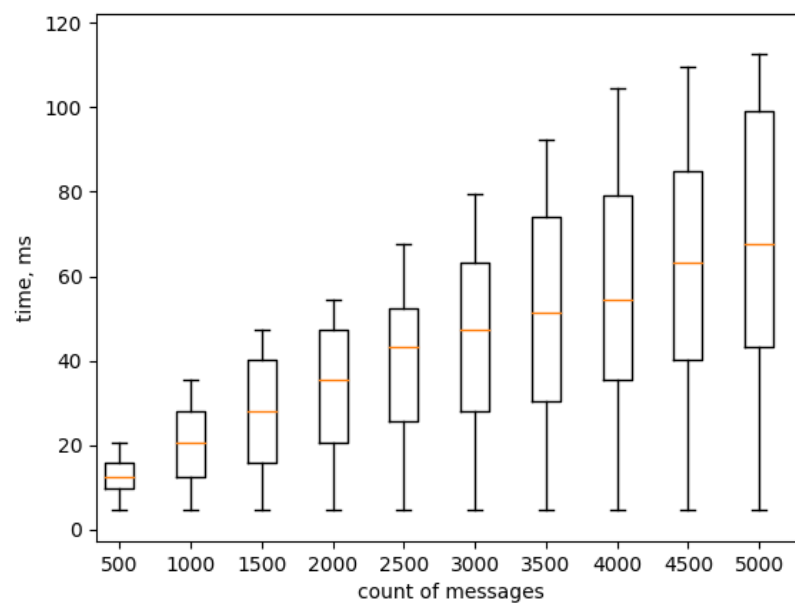


2 сценарий:

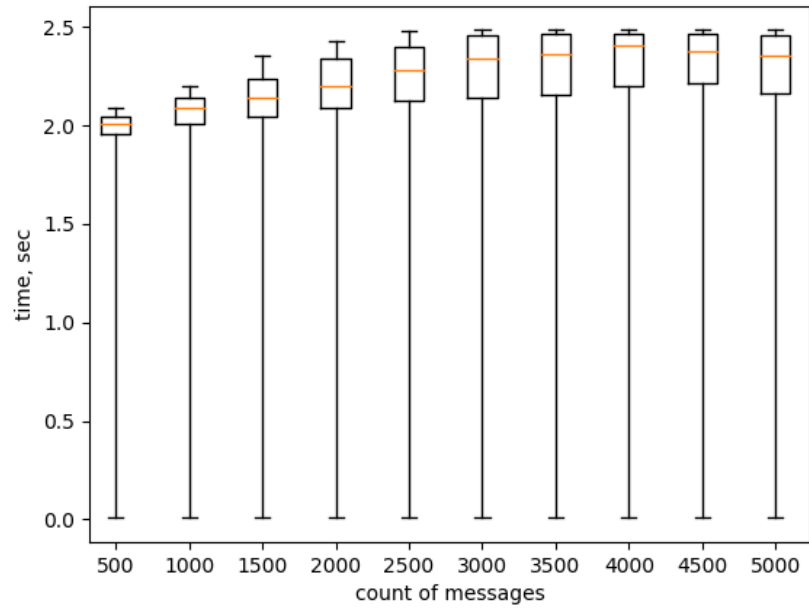
OpenSplice(500 000):



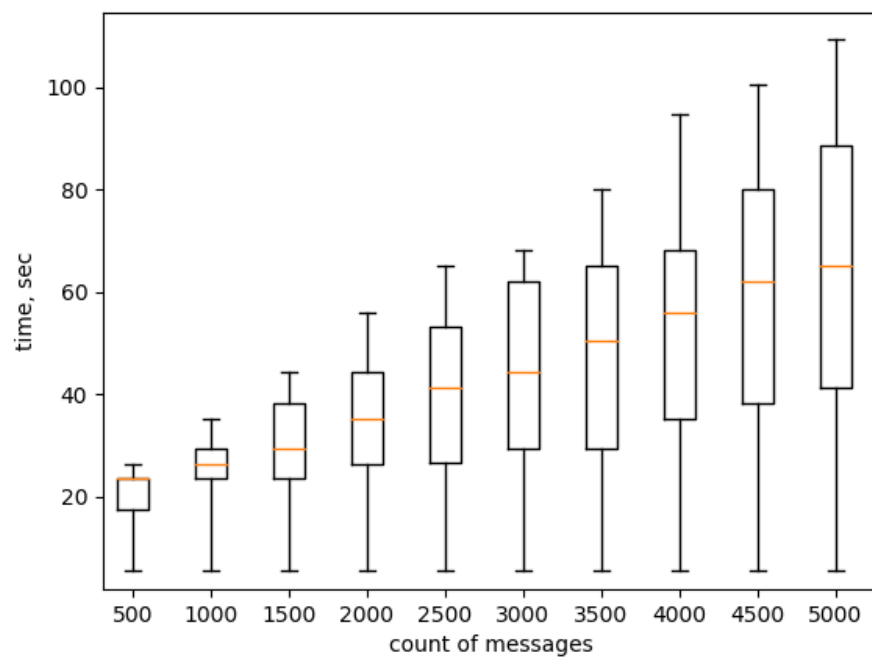
OpenSplice(60 000):



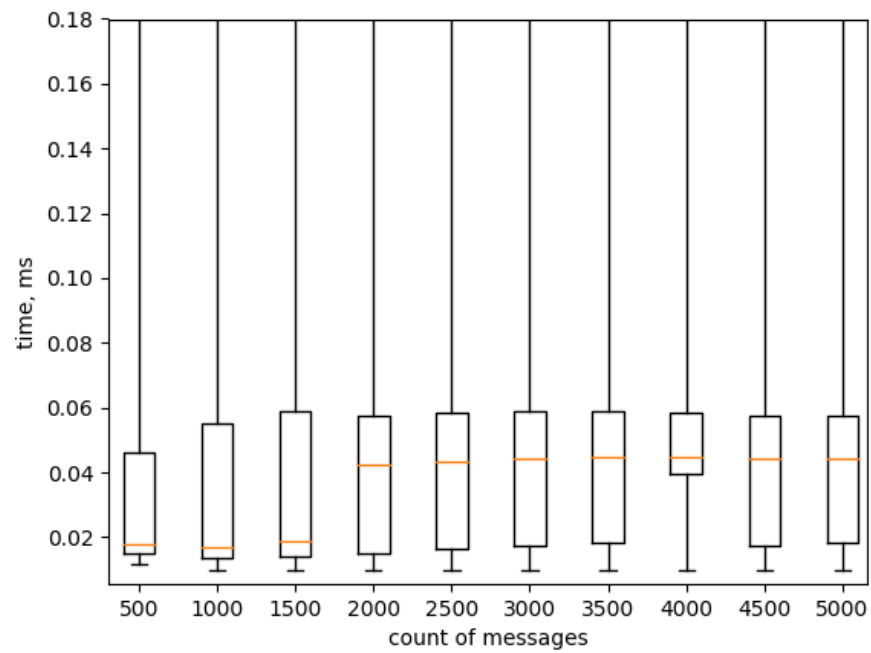
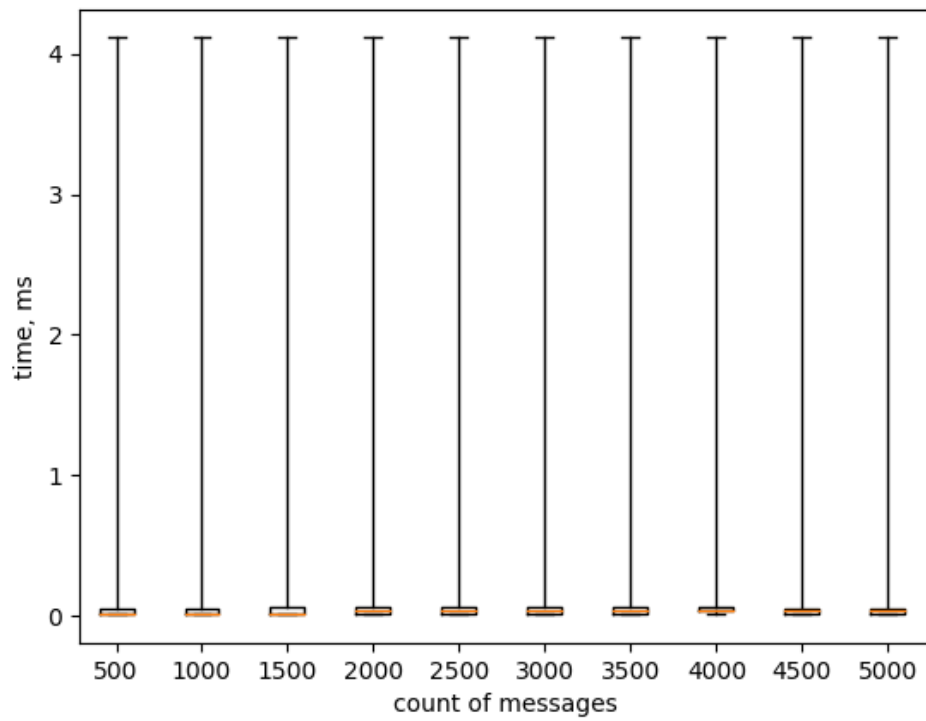
FastRTPS:



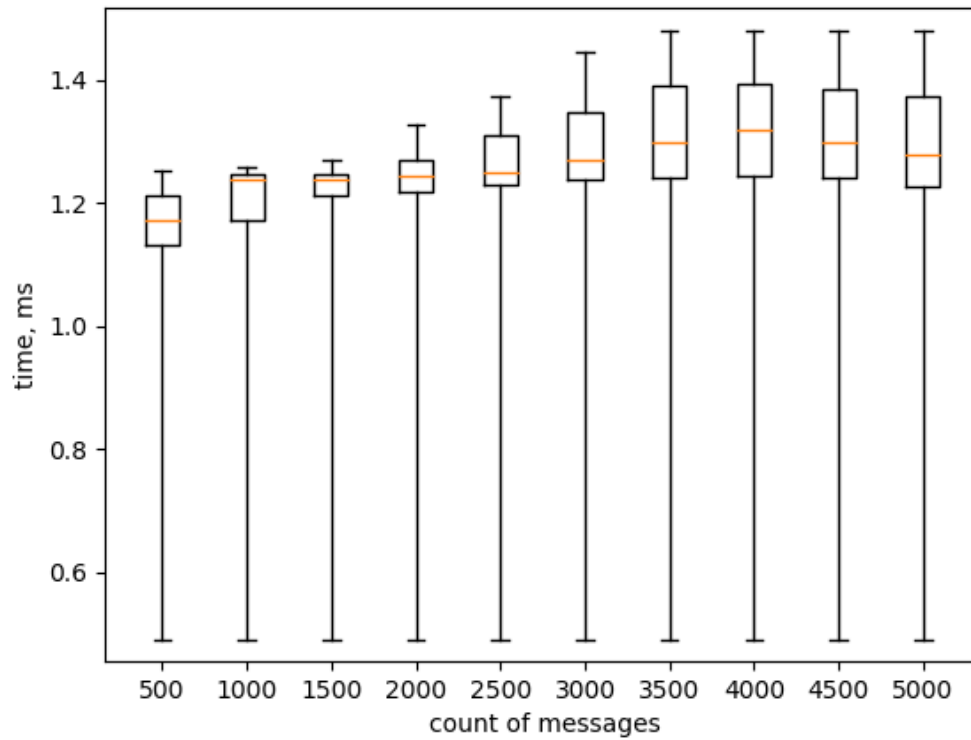
RTI Connex:



Cyclonedds:



ZeroMQ:



Тесты были проведены на следующей системе:

ROS2 Dashing

ОС: Ubuntu 18.04.3 LTS

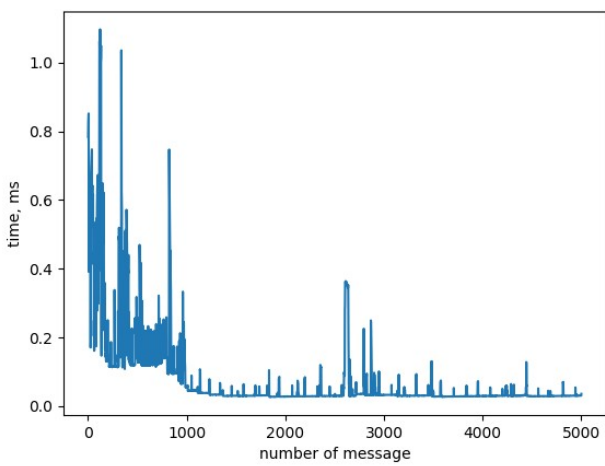
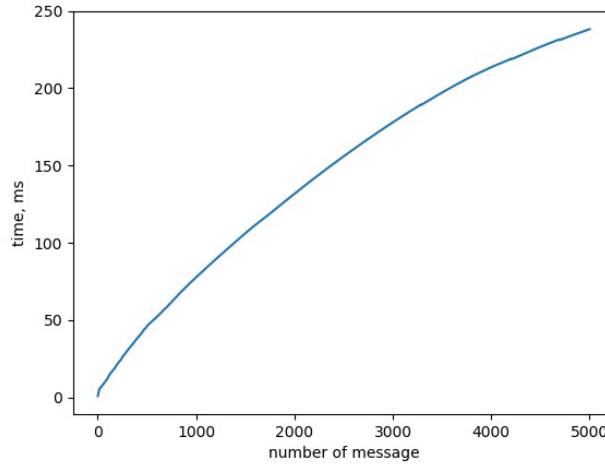
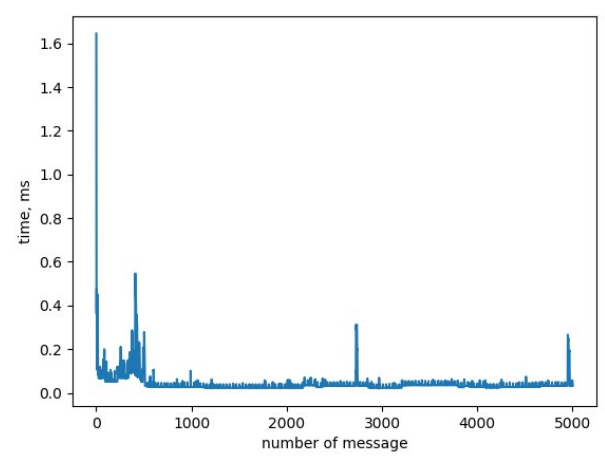
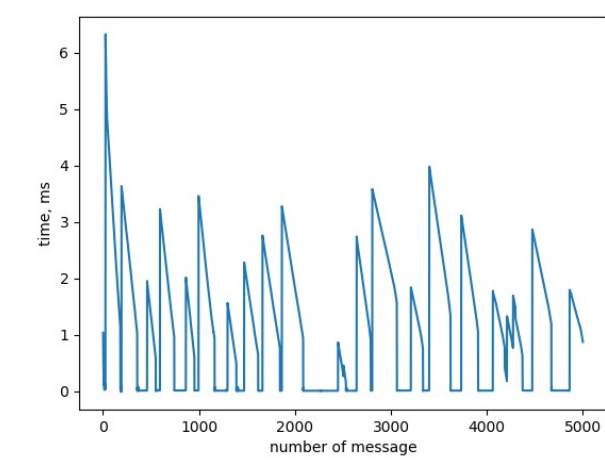
Ядро: linux-5.0.21-rt16

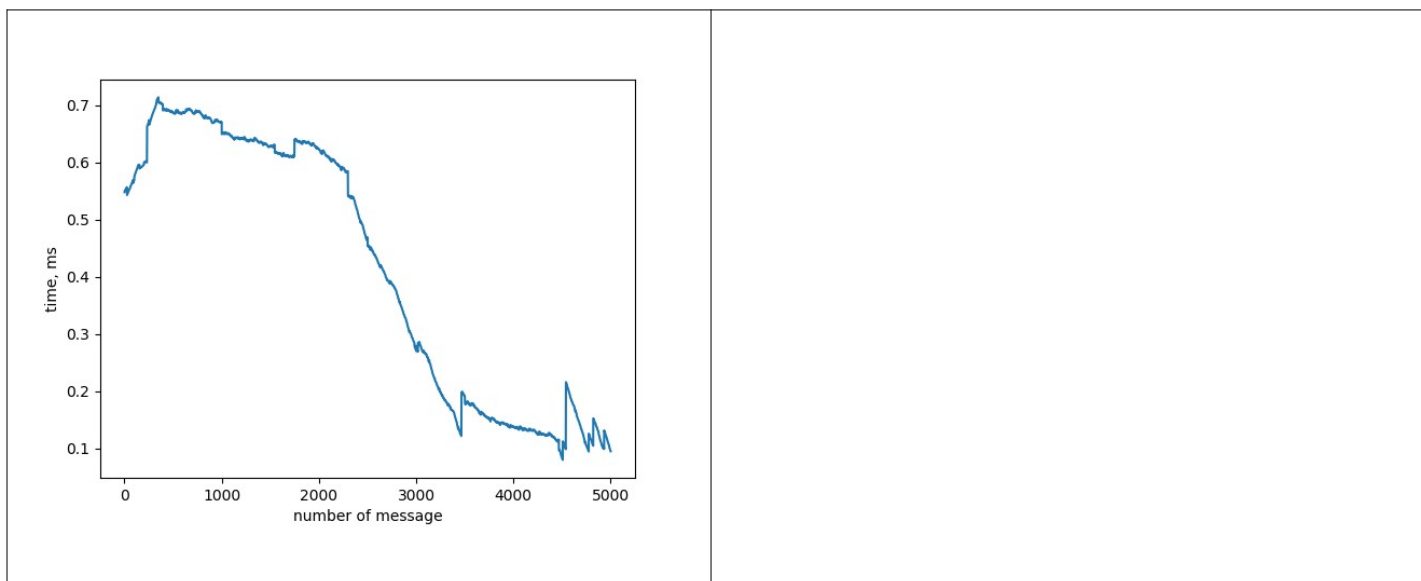
Процессор: Intel Core i5-8250U CPU 1.60GHz × 8

Оперативная память: 8Gb

7. Сравнение используемых сервисов.

7.1. Графики задержки для каждого сообщения этих сервисов при длине сообщения в 50 символов:

FastRTPS	OpenSplice
 A line graph showing latency in milliseconds for FastRTPS. The x-axis is 'number of message' from 0 to 5000, and the y-axis is 'time, ms' from 0.0 to 1.0. The graph shows high initial latency (up to 1.1 ms) for the first 1000 messages, which then drops significantly and remains low (mostly below 0.2 ms) for the rest of the 5000 messages.	 A line graph showing latency in milliseconds for OpenSplice. The x-axis is 'number of message' from 0 to 5000, and the y-axis is 'time, ms' from 0 to 250. The graph shows a smooth, continuous upward curve, starting near 0 ms and reaching approximately 240 ms at 5000 messages.
RTI Connex	Cyclonedds
 A line graph showing latency in milliseconds for RTI Connex. The x-axis is 'number of message' from 0 to 5000, and the y-axis is 'time, ms' from 0.0 to 1.6. The graph shows a sharp initial peak of about 1.6 ms, followed by a drop to near 0 ms, with a few small spikes throughout the 5000 messages.	 A line graph showing latency in milliseconds for Cyclonedds. The x-axis is 'number of message' from 0 to 5000, and the y-axis is 'time, ms' from 0 to 6. The graph shows a highly oscillatory pattern with frequent peaks and troughs, with peaks reaching up to 6 ms and troughs dropping to 0 ms.
ZeroMQ	



Как видно по результатам и на следующих графиках при маленьких сообщениях FastRTPS и RTI Connexth передает сообщения значительно быстрее и успевает обрабатывать очередь, а OpenSplice не успевает обрабатывать очередь и очередь увеличивается вместе с задержкой. Cyclonedds показывает не худший результат, но значительно медленнее чем FastRTPS и RTI Connexth, на графике видно, что сначала очередь накапливается, а затем полностью обрабатывается, поэтому на графике наблюдаются скачки задержки. В начале работы FastRTPS и RTI Connexth наблюдается большая задержка, а затем она уменьшается и держится достаточно стабильно. В итоге FastRTPS и RTI Connexth показывают схожие результаты, которые являются наилучшими, но RTI Connexth показал себя немного стабильнее FastRTPS.

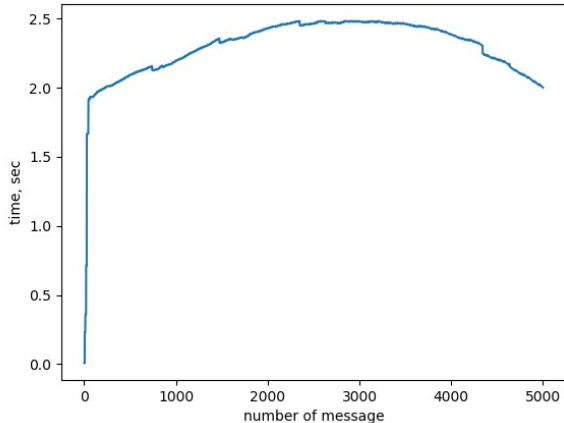
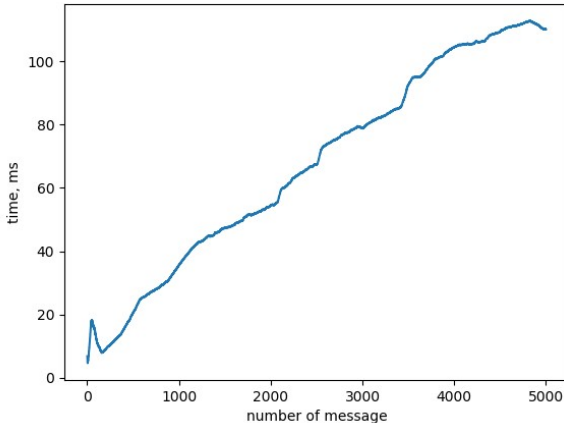
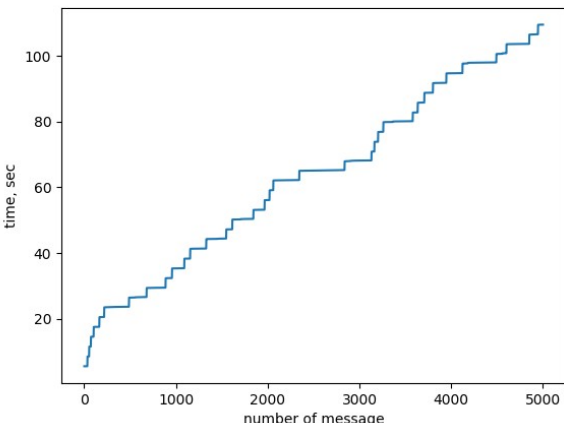
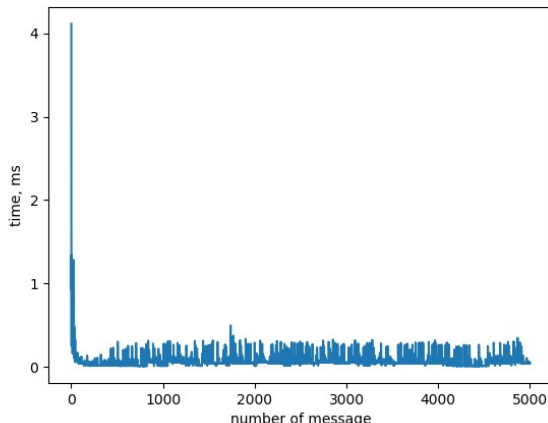
У сервис ZeroMQ вначале немного увеличивается время передачи сообщений, но затем постепенно уменьшается. Задержки сравнимы с результатами FastRTPS, но большую часть времени значительно уступает ему.

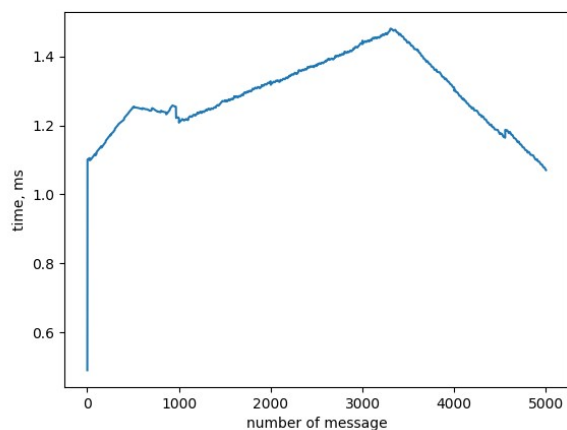
По итогу по скорости можно выстроить следующий рейтинг:

1. RTI Connexth
2. FastRTPS

3. ZeroMQ
4. Cyclonedds
5. OpenSplice

7.2. Графики задержки для каждого сообщения этих DDS при длине сообщения в 60000 символов:

FastRTPS	OpenSplice
	
RTI Connex	Cyclonedds
	
ZeroMQ	



В случае с сообщениями длиной 60 000 символов OpenSplice выдает результаты даже лучше чем при длине сообщения 50, но время доставки сообщения все время растет. У FastRTPS время доставки сообщения в начале резко увеличивается, затем постепенно увеличивается, пока очередь растет, а потом постепенно уменьшается, когда перестали поступать новые сообщения. RTI Connnext выдает худший результат со ступенчатым графиком, задержки постоянно растут. В случае с Cyclonedds время доставки крайне мало и он показывает наилучший результат с небольшими колебаниями на всей временной прямой.

Сервис ZeroMQ показывает хороший результат с аналогичным FastRTPS поведением, но обрабатывает менее постепенно, чем упомянутый FastRTPS.

По итогу по скорости можно выстроить следующий рейтинг:

1. Cyclonedds
2. ZeroMQ
3. OpenSplice
4. FastRTPS
5. RTI Connnext