**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 — издетель(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-connext-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.

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

**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

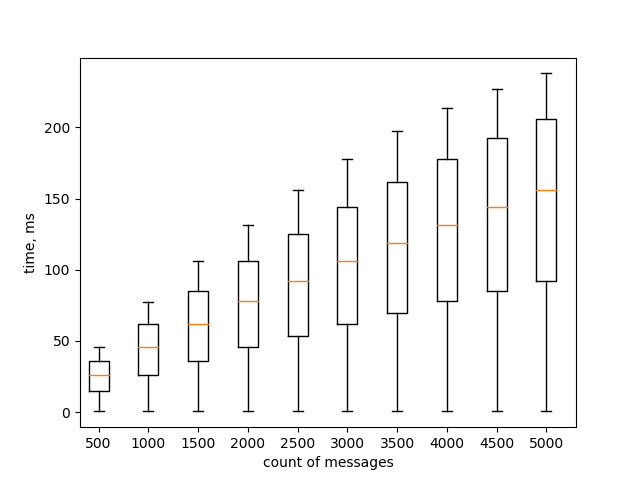
./pub 60000

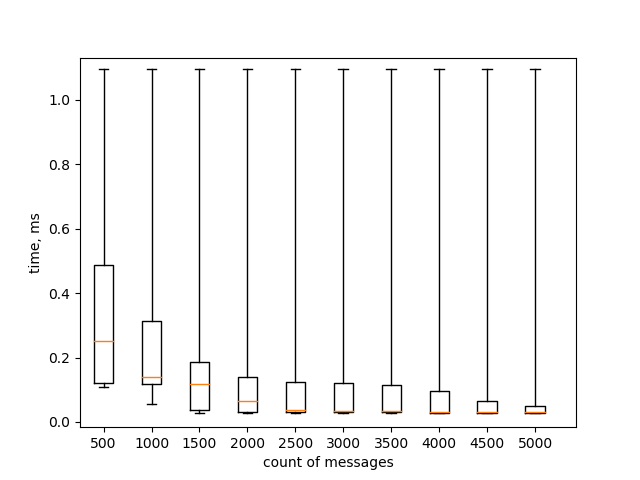
**6. Результаты.**

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

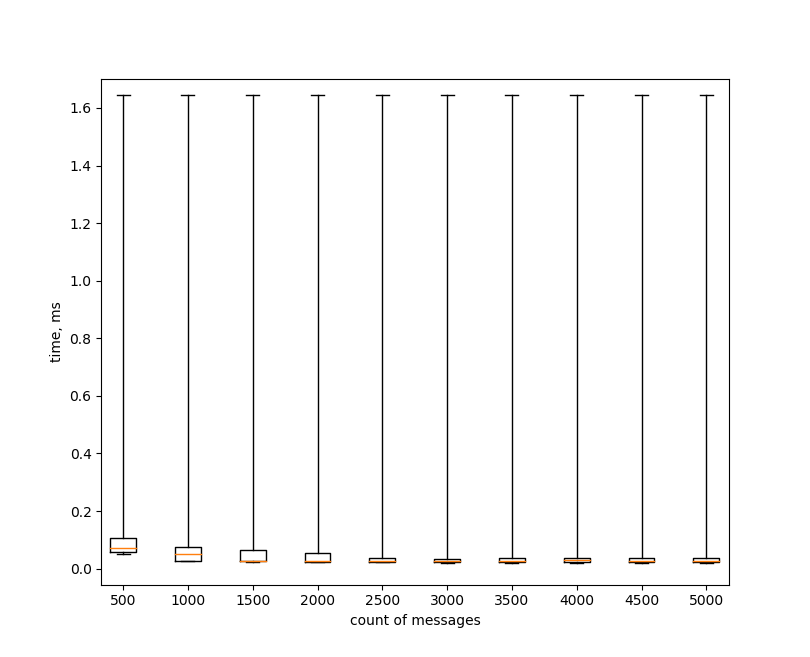
1 сценарий:

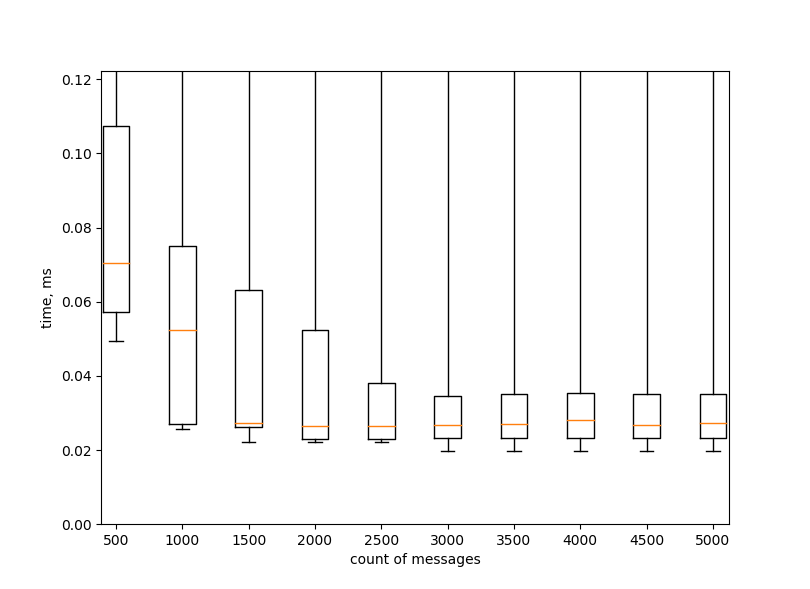
OpenSplice:



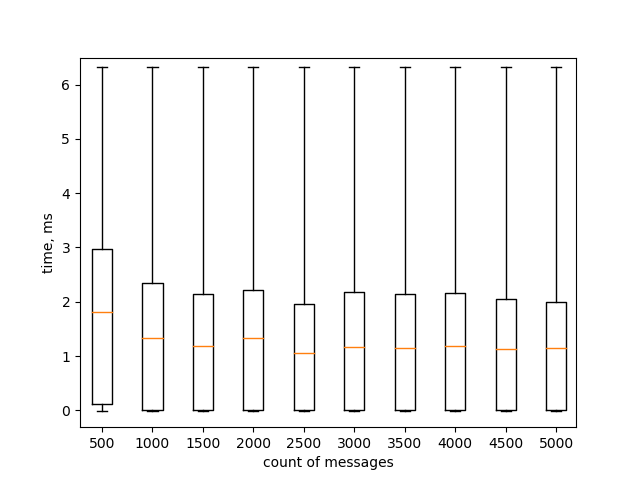
FastRTPS:

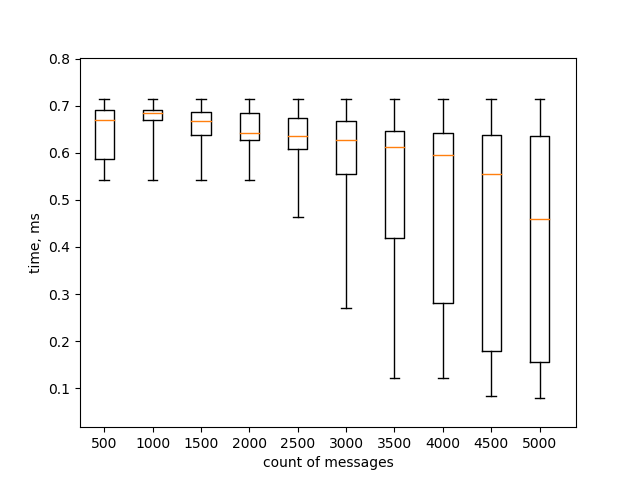
RTI Connext:





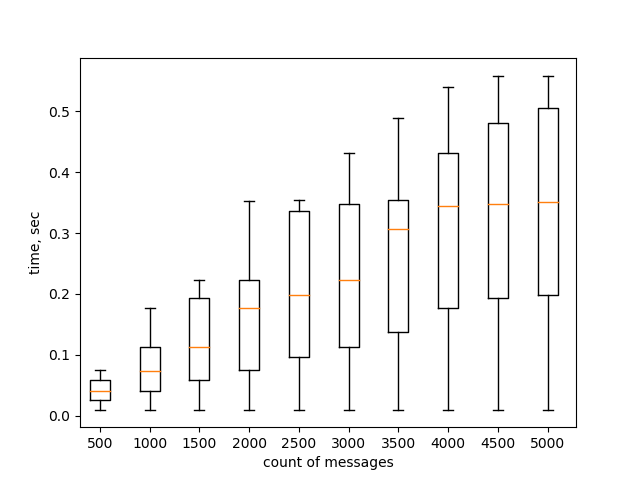
Cyclonedds:

ZeroMQ:

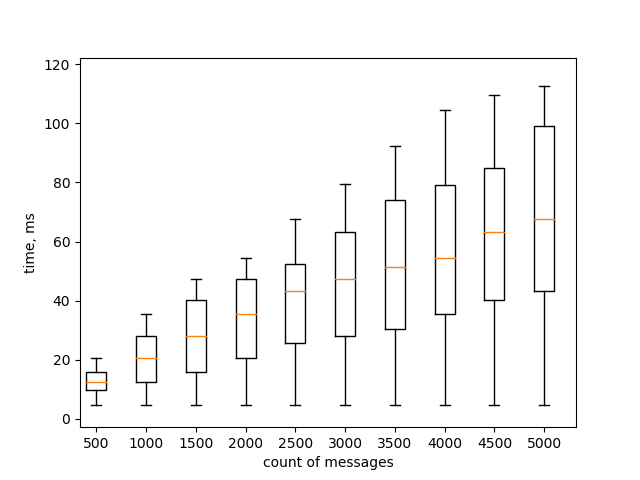


2 сценарий:

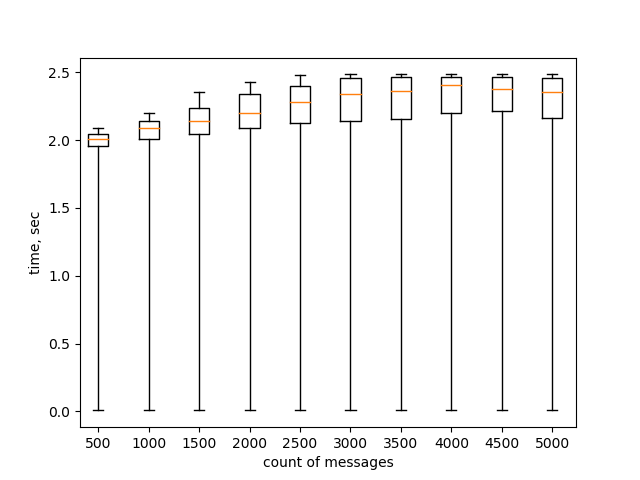
OpenSplice(500 000):

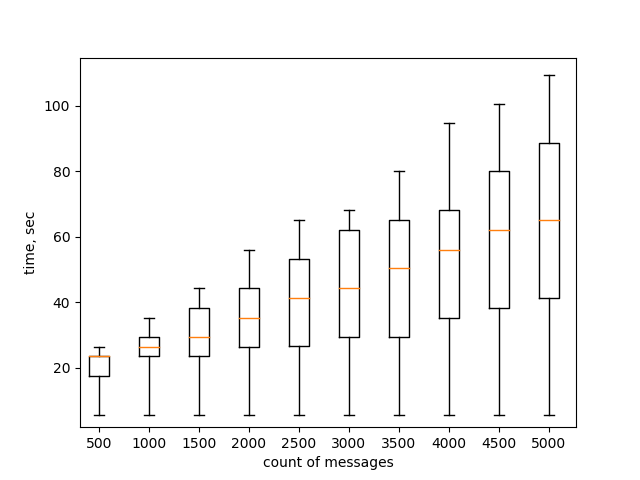


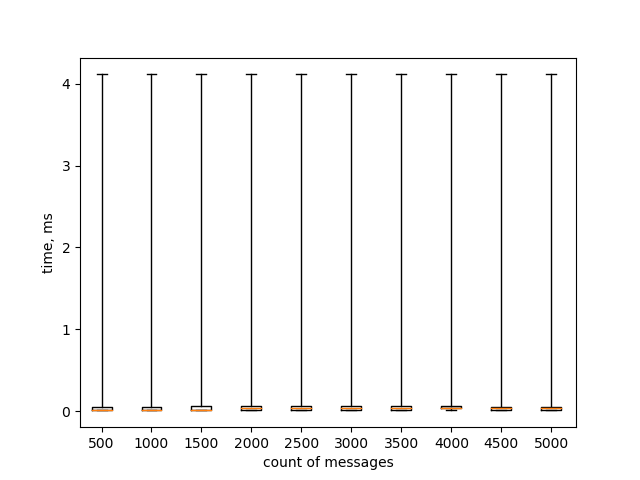
OpenSplice(60 000):

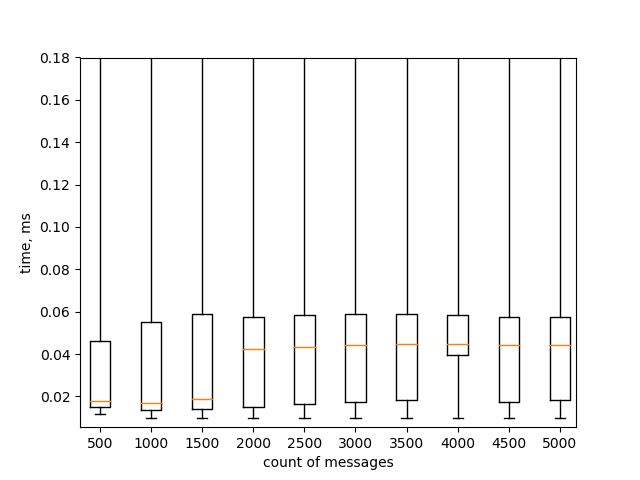


FastRTPS:

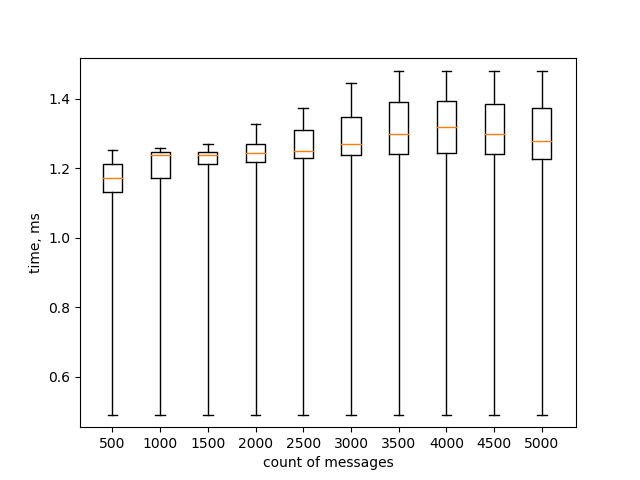
RTI Connext:



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 севрвисов при длине сообщения в 50 символов:

|  |  |
| --- | --- |
| FastRTPS | OpenSplice |
|  |  |
| RTI Connext | Cyclonedds |
|  |  |
| ZeroMQ |  |
|  |  |

Как видно по результатам и на следующих графиках при маленьких сообщениях FastRTPS и RTI Connext 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 Connext, на графике видно, что сначала очередь накапливается, а затем полностью обрабатывается, поэтому на графике наблюдаются скачки задержки. В начале работы FastRTPS и RTI Connext наблюдается большая задержка, а затем она уменьшается и держится достаточно стабильно. В итоге FastRTPS и RTI Connext показывают схожие результаты, которые являются наилучшими, но RTI Connext показал себя немного стабильнее FastRTPS.

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

По итогу по скорости можно выстроить следующий рейтинг:

1. RTI Connext

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 Connext | 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 Connext выдает худший результат со ступенчатым графиком, задержки постоянно растут. В случае с Cyclonedds вермя 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 Connext