**Построение и тестирование token-ring.**

В ходе работы я реализовал протокол токен ринг, собрал статистику с узлов в ходе работы системы, построил графики для latency и throughput при различных параметрах token-ring.

**Описание протокола Token Ring**

1. Система состоит из *N* пронумерованных от *0* до *N-1* узлов (потоков). Узлы упорядочены по порядковому номеру. После состояния *N-1* следует узел *0*, т.е. узлы формируют кольцо.
2. Соседние в кольце потоки могут обмениваться пакетами. Обмен возможен только по часовой стрелке.
3. Каждый поток, получив пакет от предыдущего, отдает его следующему.
4. Пакеты не могут обгонять друг друга.

**Общее описание реализации.**

Система состоит из класса процессора, отвечающего за инициализацию узлов, запуск/остановку процесса пересылки пакетов и сбор статистики.

Класс узла использует блокирующую очередь в которую поступают сообщения от предыдущего узла. Получив сообщение, узел обновляет статистические данные (количество обработанных сообщений, время задержки) и отправляет его в очередь следующего узла.

Main класс запускает и останавливает спустя некоторое время систему, передавая на вход разные параметры числа узлов и числа пакетов, изначально находящихся на каждом узле. После запуска процессора, Main создает ExecutorService для сбора статистики: раз в 10 секунд статистика системы собирается и записывается в csv файл. Процессор получает команду остановить работу, спустя 3 минуты.

**Характеристики.**

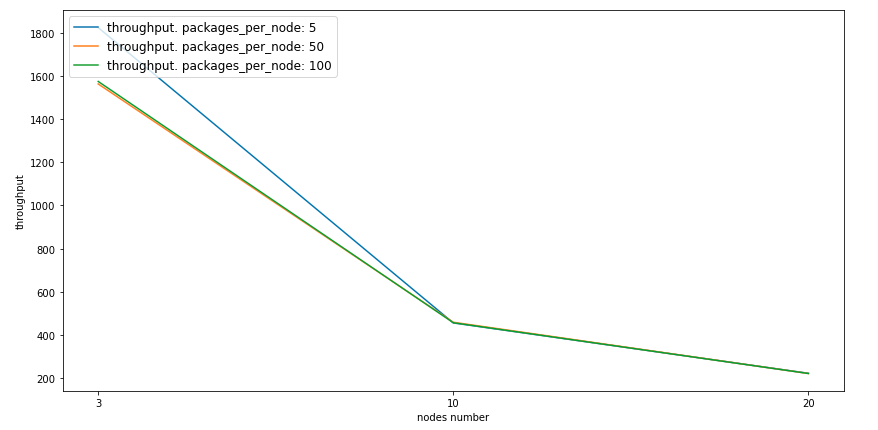
1. Latency рассчитывается так: когда узел достает из очереди очередной пакет, он рассчитывает сколько времени прошло с момента transitionStartTime, сохраненного в пакете и прибавляет разницу к переменной totalLatency. После чего сеттит текцщее время в пакет и отправляет в очередь следующего узла. При этом увеличивается число пакетов, отправленных данным узлом. Средняя задержка считается как сумма totalLatency со всех узлов, поделенная на сумму пакетов, отправленных всеми узлами.
2. Throughput считается как усредненное по узлам количество отправленных пакетов, деленое на время, прошедшее со старта системы.

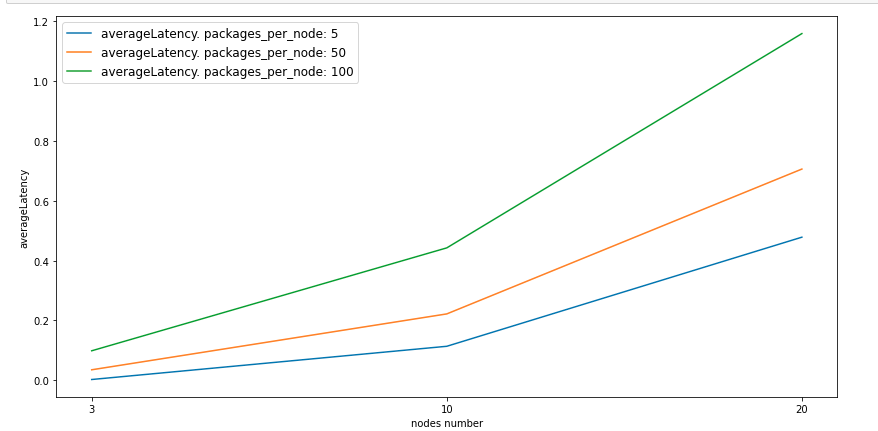
**Варианты реализации очереди**

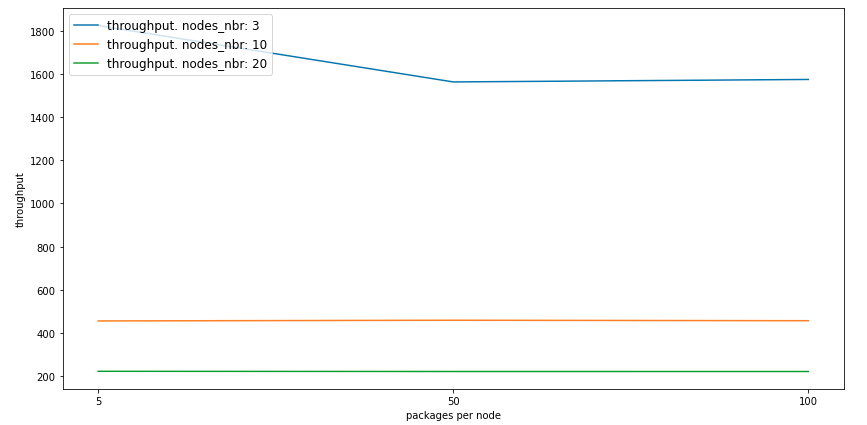
В ходе экспериментов были рассмотрены 2 варианта очереди: ArrayBlockingQueue и LinkedBlockingQueue.

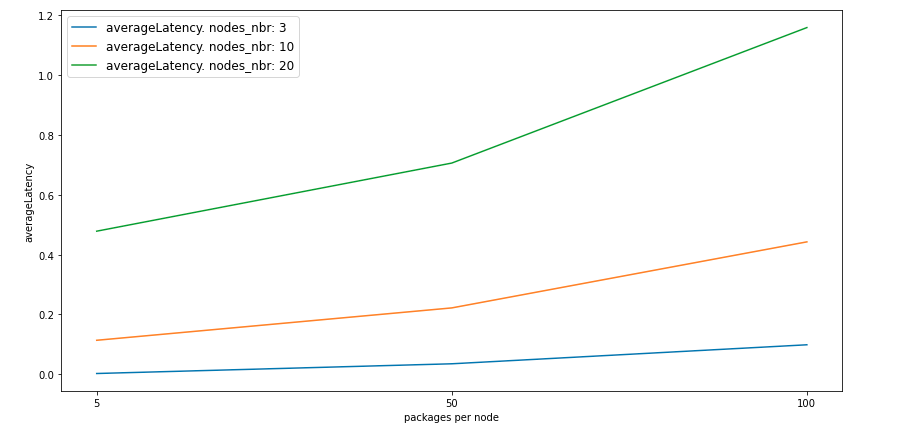
Для ArrayBlockingQueue имеется ограничение на число пакетов, так как размер очереди фиксированный. В реализации если очередь переполнялась, пакеты терялись и увеличивался счетчик потерянных пакетов.

**Графики и статистика (LinkedBlockingQueue)**

****

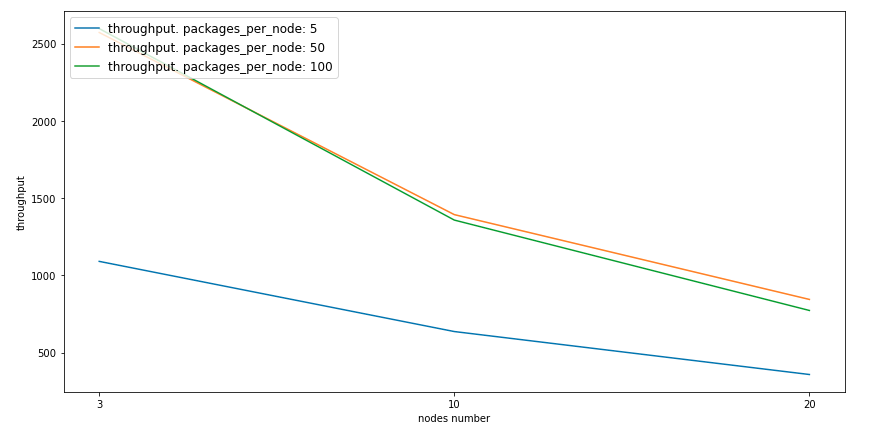
****

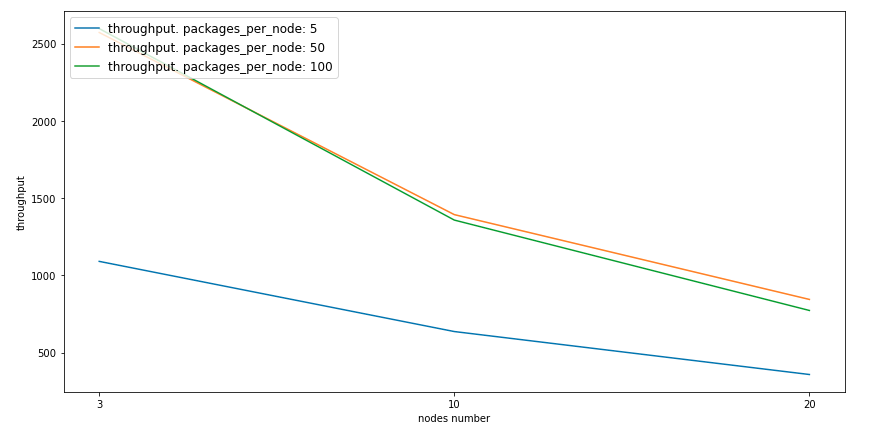
****

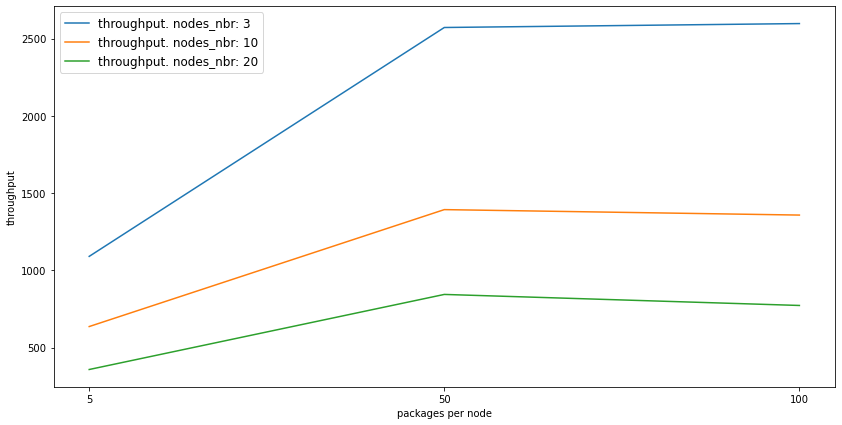
****

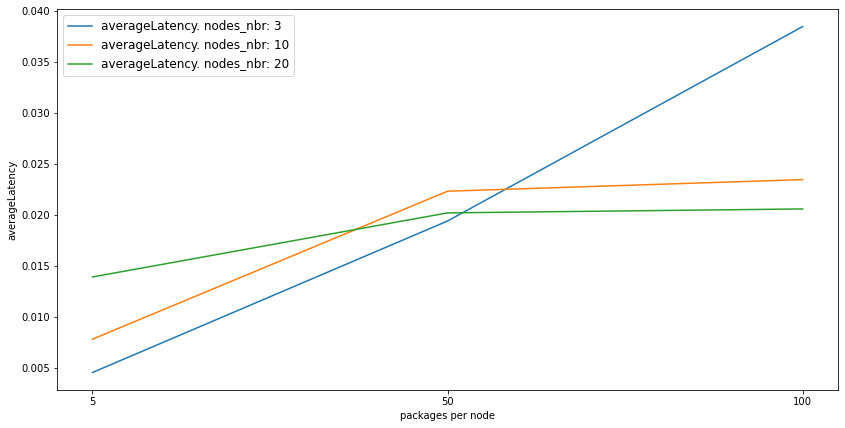
Из графиков видно, что при росте числа узлов пропускная способность падает, а задержка растет. При фиксированном числе узлов, при росте числа пакетов задержка увеличивается, а пропускная способность незначительно снижается.

**Графики и статистика (ArrayBlockingQueue)**









При росте числа узлов начинают теряться пакеты, поэтому полностью доверять можно только графику для 3 узлов. Видно, что при росте числа узлов пропускная способность падает, а задержка растет. При росте числа пакетов как показывают графики пропускная способность растет даже для 3 узлов, однако задержка как нестранно тоже растет.

Сравнивая статистику для 2х реализаций я пришел к выводу что Linked очередь работет лучше для большего числа узлов. Однако при малом равном количестве узлов и большей нагрузке лучше справляется Array

