Моделирование TokenRing.

Выполнил: Колпаков Георгий

Реализация

Реализация модели протокола TokenRing выполнена на языке Java и представляет из себя maven проект и включает в себя 5 классов.
1 класс Constants включает в себя используемые константы.
Классы MacAddress и Frame описывают доменную модель.

Класс Node моделирует поведение узла tokenring.

Класс TokenRingModel является точкой входа и содержит в себе main метод, который запускает модель для разных наборов данных и в папке проекта генерирует csv файл „calc.csv” с результатами моделирования. Main метод не требует аргументов.

Класс MacAddress содержит в себе массив из 6 байт мак адресса.
Класс Frame содержит в себе:

1. поле id: уникальный автоматически генерирующийся идентификатор фрейма
2. поле destinationAddress: mac-адресс узла-получателя фрейма
3. поле sourceAddress: mac-адресс узла-отправителя фрейма
4. поле data: передаваемые данные
5. поле fcs: контрольная сумма фрейма
6. поле creationNanoTime: время создания фрейма, которое берется из метода java System.nanoTime
7. поле distanceBetweenAddresses: расстояние между узлом-получателем и узлом-отправителем фрейма.

Класс Frame является интерпретацией Information Frame, формата передаваемых данных в протоколе TokenRing. Класс frame в отличие от Information Frame не содержит полей начала и конца кадра(в объектно-ориентированных языках програмирования это ненужно), не содержит байта Access Control(этот байт не используется, поскольку в рамках этой модели не делается различий между токеном и information frame).

Также класс Frame содержит поле creationNanoTime, которое используется для проведения вычислений.

Класс Node содержит в себе поля

1. address: адрес узла
2. frameConsumer: функция, работающая с фреймом, при его получении, в нашей модели эта функция вычислияет время доставки фрейма
3. nextNode: узел, следующий в кольце за данным
4. isFinished: флаг, обозначающий завершение работы tokenRing
5. frames: очередь обрабатываемых узлов

Класс Node содержит в себе следующие публичные методы(не являющиеся геттерами или переопределенными методами класса Object):

1. getQueueTask: возвращает объект анонимного класса, реализующий интерфейс Runnable. Этот объект осуществляет работу с очередью данного узла, а именно валидация фреймов по чек-сумме, обработка фреймов, посланных данному узлу, пересылка фреймов, посланных другим узлам сети
2. finish: устанавливает для данного узла флаг завершения работы сети
3. enqueueFrame: добавляет фрейм в очередь обработки фреймов данного узла-отправителя

Класс TokenRingModel содержит в себе следующие поля:

1. nodes: связный список узлов этого tokenring
2. executorService: ExecutorService с фиксированным количеством потоков, равным количеству узлов в сети, который исполняет таски узлов по обработке своих очередей фреймов
3. messageTimeout: таймаут в мс, выдерживаемый между посылкой сообщений в сеть
4. numberOfMessagesToSend: количество сообщений, которое будет отослано в запуске данной модели
5. sendTimes: словарь, ключами которого являются пересланные фреймы, а значениями — время их доставки

Класс TokenRingModel содержит в себе следующие публичные методы:

1. run: метод, запускающий модель и возвращающий после её завершения поле значение поля sendTimes. Метод отсылает сообщения, передавая их между случайными узлами. Начало отправки заключается во включении фрейма сообщения в очередь узла-отправителя. После передачи всех сообщений в очереди узлов, запускается метод stop который ожидает передачи всех узлов и осуществляет shutdown executorService модели.
2. Main: точка входа в модель. Для каждого набора количества узлов(10, 20, 50, 100, 200, 500, 1000, 2000) и количества сообщений(20%, 40%, 60%, 80%, 90% от количества узлов) 10 раз запускается модель. Для 10 запусков вычисляется средняя пропускная способность, определяемая как количество переданных сообщений, разделенное на время работы запуска модели, и среднее время отклика, определемое как среднее время передачи сообщения в сети. После завершения моделирования, в файл calc.csv в папку проекта сохраняется csv файл из 4 полей: количество узлов, количество сообщений, средняя пропускная способность(кол-во сообщений в наносекунду), сренднее время передачи сообщений в сети(в наносекундах).

Запуск

Запуск проводился на ноутбуке со следующей конфигурацией:

Процессор Intel Core i7-6500U, 2 физических ядра, 4 виртуальных(режим HT), частота 2.6 ггц

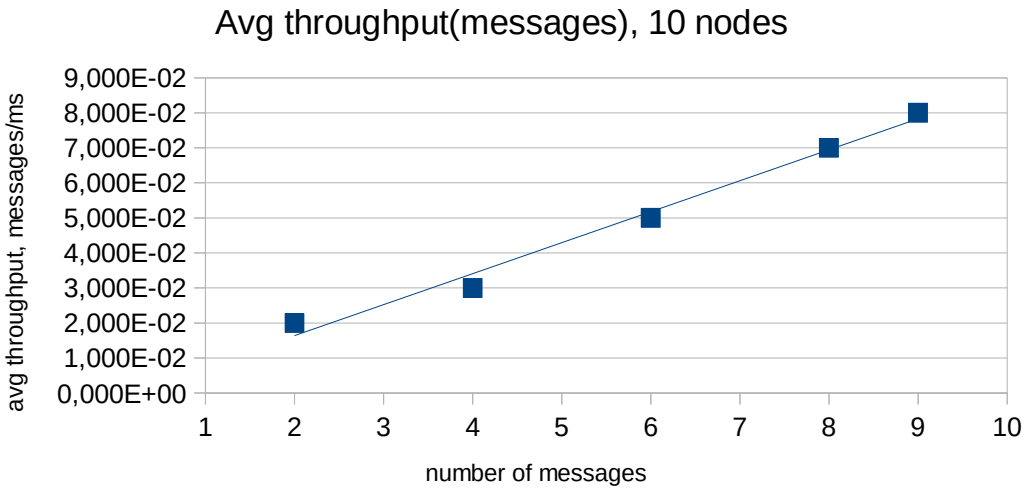
ОЗУ: 8гб

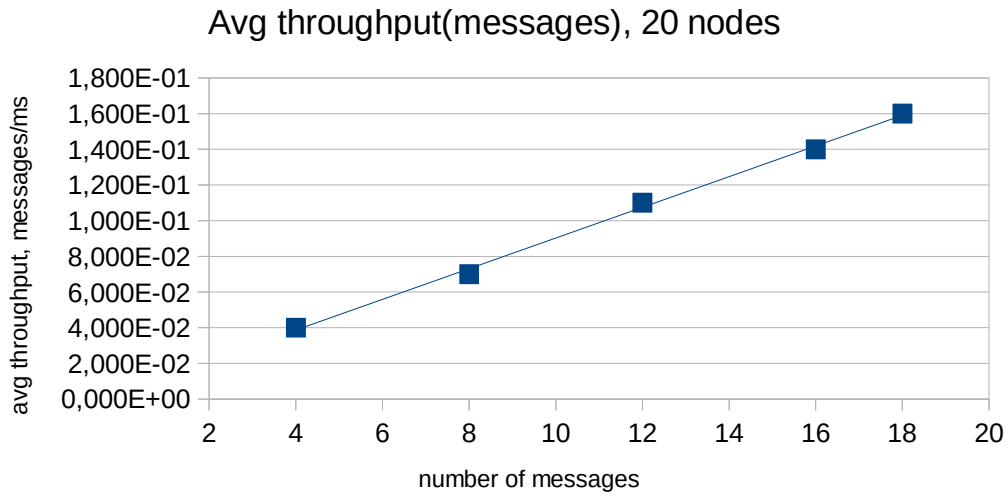
ПЗУ: 256 Гб SSD

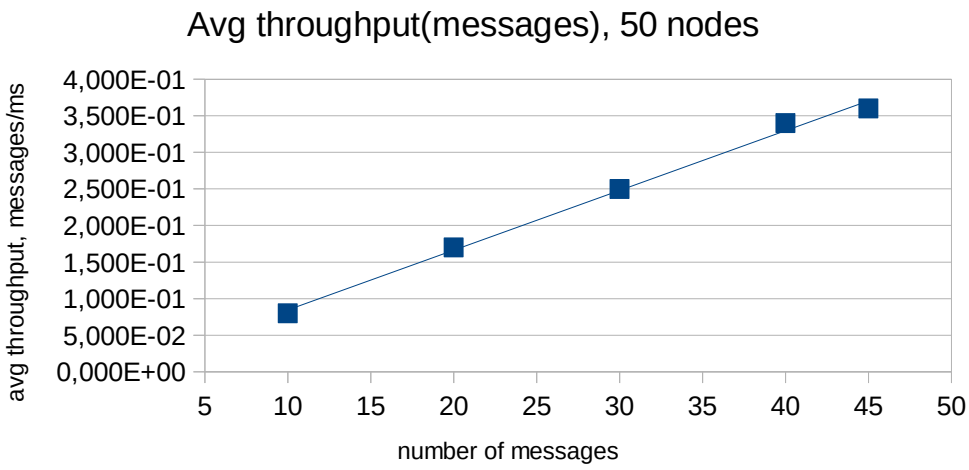
Операционная система windows 10

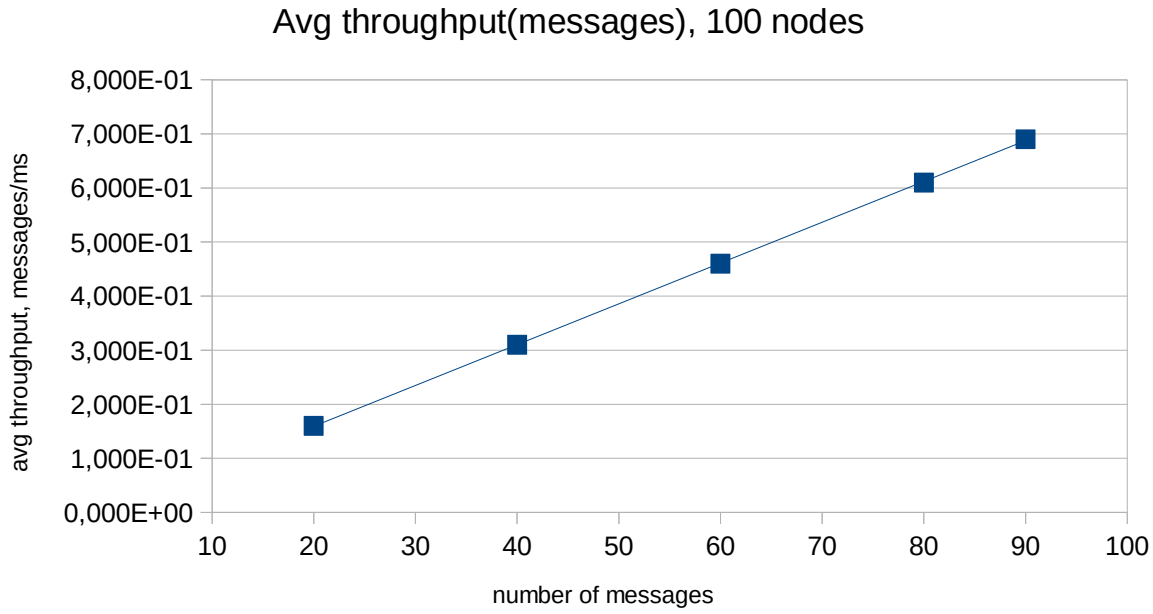
JDK: 1.8.0_92

Измерение средней пропускной способности.

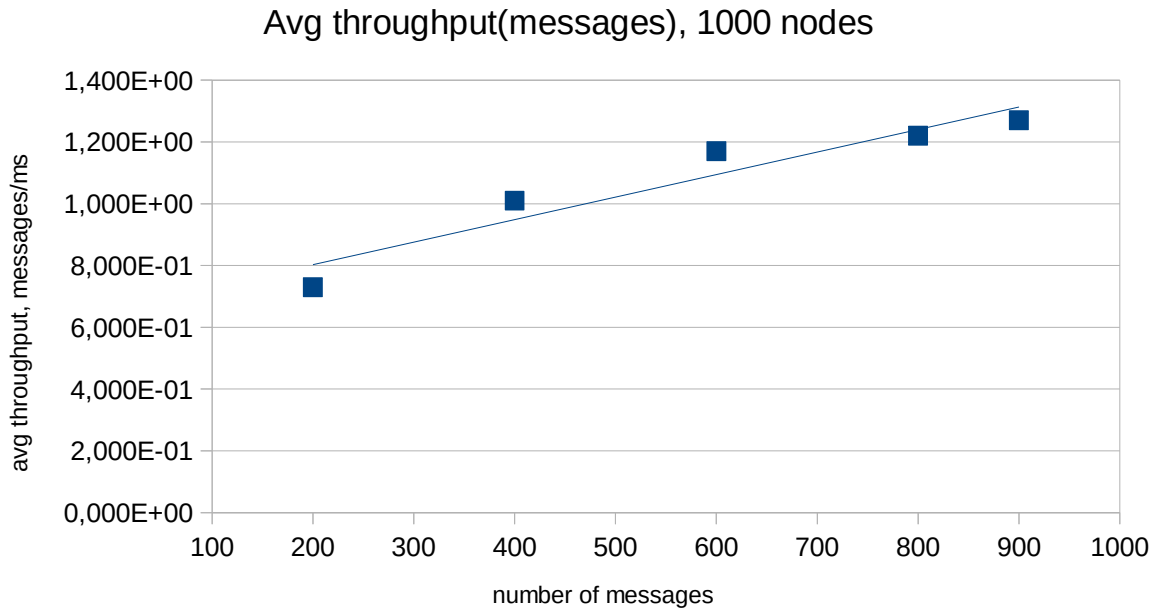


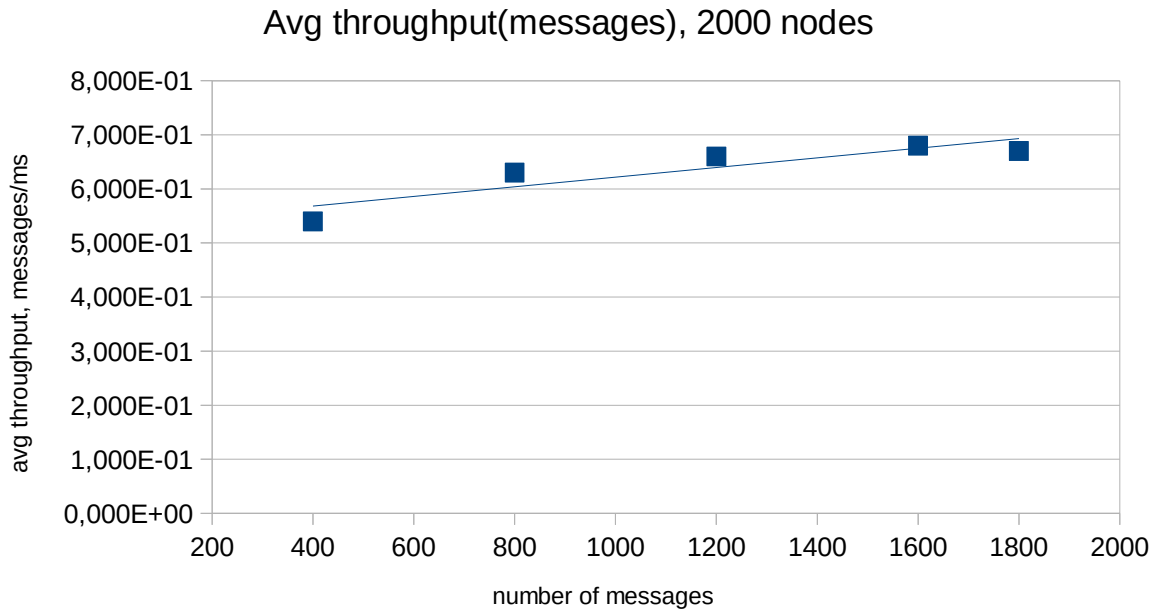






Выше представлены графики зависимости средней пропускной способности от количества переданных, за сессию, сообщений для 10, 20, 50 и 100 узлов.
На всех графиках наблюдаем линейную зависимость средней пропускной способности от количества переданных сообщений. Результат соответствует ожиданиям: когда сеть недогружена(количество сообщений в разы меньше количества узлов), пропускная способность невысока, поскольку не используются все ресурсы сети, когда сеть загружена(количество сообщений становится близко к количеству узлов в сети), сеть показывает свою наибольшую производительность, поскольку загружены все ресурсы в сети.

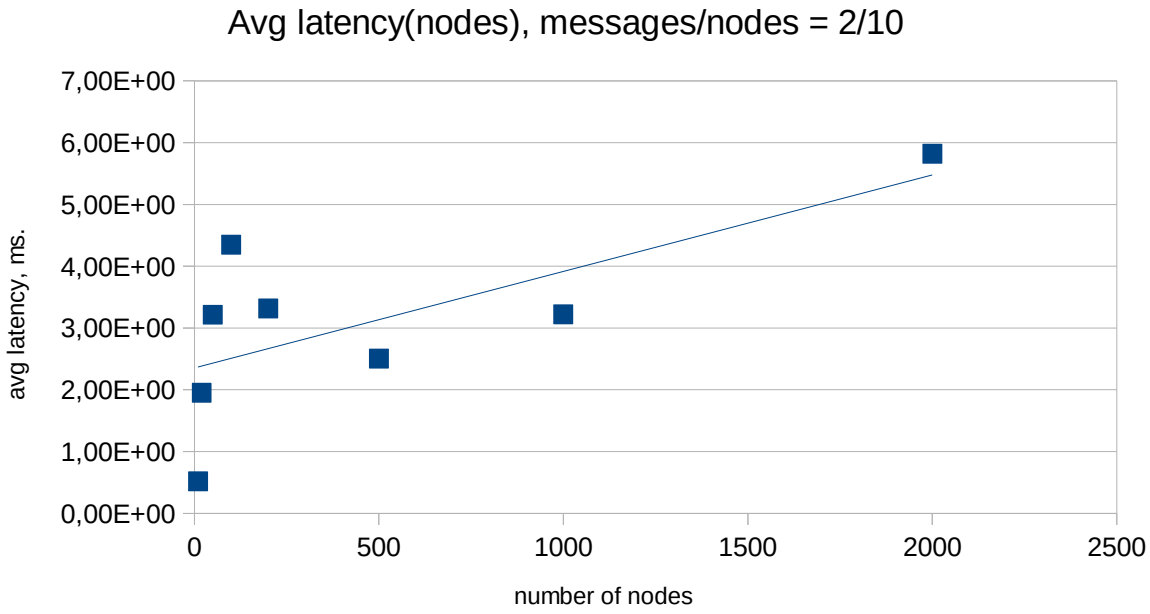


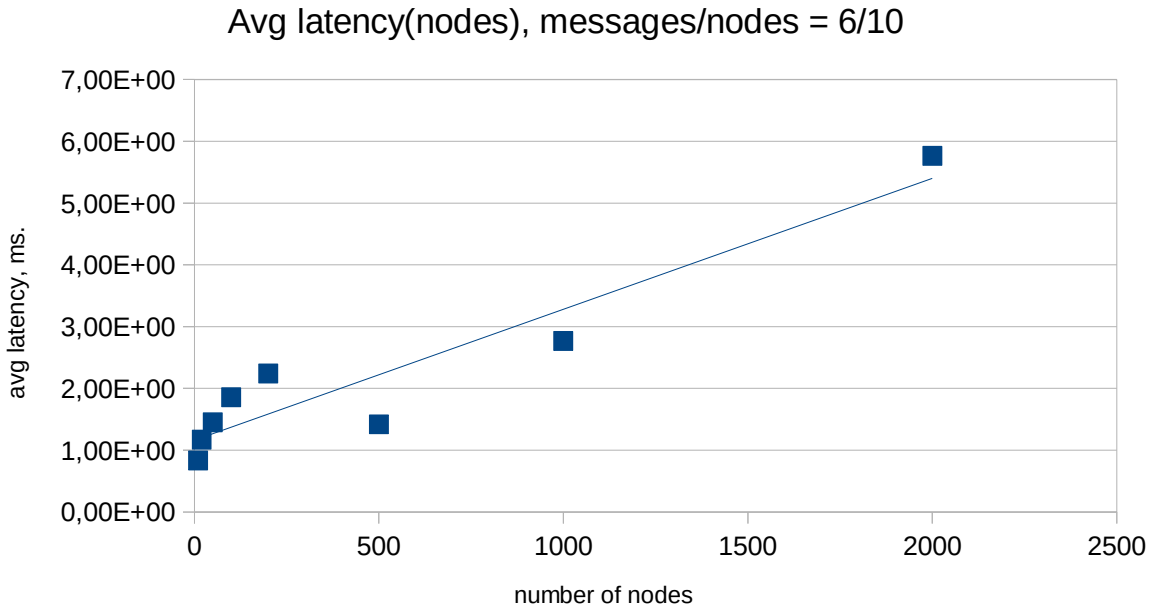


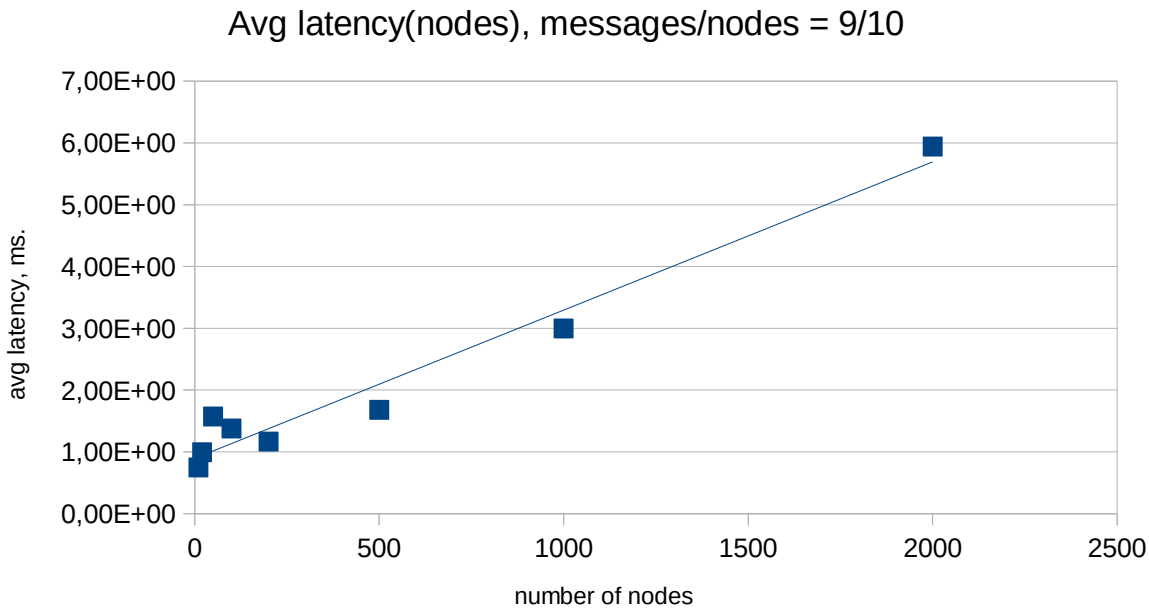
Выше представлены графики зависимости средней пропускной способности от количества переданных, за сессию, сообщений для 1000 и 2000 узлов.

Можно увидеть, что линейная зависимость между пропускной способностью и количеством переданных сообщений практически исчезла по сравнению с запусками для меньшего количества узлов. Связываю это с несовершенством модели, поскольку для каждого узла создается свой поток, то для большого количества моделируемых узлов(больше 1000) переключение контекста между потоками занимает все большое и большое время, уменьшая долю процессорного времени, затрачиваемого на непосредственно работу.

Измерение времени доставки сообщения







Выше представлены графики зависимости времени доставки сообщения от количества узлов в сети для запусков, в которых количество отправленных сообщений было равно 20%, 60% и 90% от всех узлов в сети.

Везде видим сложную, похожую на кубическую, зависимость времени доставки сообщений от количества узлов. Теоретически ожидается линейная зависимость, поскольку с ростом размера сети линейно растет и число узлов, через которое пройдет сообщение во время передачи. Вероятнее всего, причину такой сложной зависимости нужно искать в оптимизациях применяемых jvm для большого количества потков, которые могут снижать время переключение контекста, а вместе с ним и ускорять передачу сообщения по сети.