Московский Физико-Технический Институт
(Государственный университет)

Кафедра банковских информационных технологий

Token Ring

Выполнила студентка 285 гр. Марина Белялова

Москва, 2017

Содержание

1 Описание задачи

В данной работе описана реализация модели сетевого протокола Token Ring на языке Java. Целью данной работы являлось исследование зависимости характеристик latency и throughput от числа нод и загруженности нод, а так же поиск варианта оптимизации работы для недогруженного и перегруженного режимов передачи пакетов.

- Система состоит из N пронумерованных от 0 до $N-1$ нод. Ноды упорядочены по порядковому номеру. После ноды $N-1$ следует нода 0, т.е. ноды формируют кольцо.
- Соседние в кольце ноды могут обмениваться пакетами. Обмен возможен только по часовой стрелке.
- Каждая нода, получив пакет от предыдущего, отдает его следующему.
- Пакеты не могут обгонять друг друга.

2 Основные сущности и соответствующие классы

2.1 Сообщение

Сообщения, которые передаются в системе, реализованы в классе `Message`. Сообщение имеет размер и при отправке фиксирует в себе время. Содержит в себе логическую величину `hasBeenDelivered`.

2.2 Пакет

Класс `Frame` реализует сущность пакета, который может пребывать в двух состояниях: `isToken()= true`, т.е. фрейм пустой и не содержит в себе сообщение, либо фрейм содержит сообщение, которое может быть доставлено или не доставлено. Когда фрейм представляет собой токен, любая нода, желающая отправить сообщение, может использовать этот фрейм.

2.3 Узел

Класс `Node`, наследник класса `Thread`, реализует узел сети. Каждая нода имеет очередь `Queue<Message> pendingMessages = new ConcurrentLinkedQueue<>()` сообщений, ожидающих отправки, и очередь входящих фреймов `Queue<Frame> enqueuedFrames = new ConcurrentLinkedQueue<>()`. `Node` содержит в себе логику обработки входящих фреймов:

```
public void handleTheFirstFrameInTheQueue() {
    Frame currentFrame = enqueuedFrames.remove();

    if (Settings.debugModeIsOn) printReport(currentFrame);

    if (currentFrame.isToken()) {
        if (IHaveAPendingMessage()) {
            currentFrame.sendMessage(pendingMessages.remove());
        }
    } else {
        if (IAmTheReceiver(currentFrame)) {
            handleIncomingMessage(currentFrame);
        } else if (IAmTheSender(currentFrame)) {
            if (currentFrame.messageHasBeenDelivered()) {
                handleDeliveredMessage(currentFrame);
            } else if (currentFrame.messageNotYetDelivered()) {
                handleUndeliveredMessage(currentFrame);
            }
        }
    }
    forwardFrame(currentFrame);
}
```

В методе `void handleIncomingMessage(Frame currentFrame)` успешность доставки сообщения имеет распределение Бернулли с вероятностью успеха p. Конкретные значения в экспериментах указаны в соответствующих разделах. В случае успеха нода-получатель ставит метку `hasBeenDelivered` у сообщения и записывает в массив `List <Double> deliveryTimes` время, затраченное на доставку сообщения. В методе `void handleDelivered Message(Frame currentFrame)` нода-отправитель, получая обратно сообщение, прошедшее полный круг и полученное адресатом, высвобождает токен. В методе `void handleUndelivered Message(Frame currentFrame)` нода-отправитель реализует логику обработки сообщений, не полученных адресатом, в соответствии с параметром возможного удержания токена token holding time (THT): если время, прошедшее с момента отправки сообщения, превысило THT, то нода удаляет сообщение и выпускает токен. Конкретные значения THT в экспериментах указаны в соответствующих разделах.

2.4 Генератор сообщений

Экземпляр класса `MessageGenerator`, наследник класса `Thread`, помещает сообщение на случайную ноду. События появления сообщений описываются экспоненциальным распределением с характерным временем $\tau = \frac{1}{\lambda}$. Последовательные интервалы времени между появлением сообщений получаются функцией: $F^{-1}(p) = \frac{-ln(1-p)}{\lambda}$, где p –

случайная величина, равномерно распределённая в интервале $[0, 1]$. Обоснование выбора того или иного характерного времени приведено в разделе "Основные параметры и метрики".

2.5 Вспомогательный класс запуска

Класс `Launcher` осуществляет инициализацию нод, фреймов, начальное распределение фреймов по нодам, запуск тредов нод и треда генератора сообщений, остановку тредов по достижении необходимого числа доставленных сообщений, логгирование в файл.

3 Основные параметры и метрики

Для измерения времени использовалась системная функция `System.nanoTime()`.

Введём следующие параметры запуска:

- N – число нод.
- F – число фреймов.
- M – целевое число доставленных сообщений. Во всех расчётах было выбрано число $10 * N$.
- τ – характерное время появления сообщений в генераторе.
- THT (token holding time) – время возможного удержания токена.

Введём следующие величины, которые рассчитываются по результатам запуска.

- T – время выполнения.
- l – среднее время доставки сообщений, т.е. latency. Среднее значений массива `List <Double> deliveryTimes`.
- t – среднее число передаваемых сообщений в единицу времени, т.е. throughput. $t = \frac{M}{T}$.
- P – среднее число сообщений, передаваемых в системе в единицу времени.
- LR (load rate) – загрузка системы сообщениями. $LR = \frac{P}{N}$.

3.1 Подбор характерного времени появления сообщений в генераторе τ

τ подбиралось таким образом, чтобы обеспечить необходимую загрузку системы сообщениями P. В абстрактной системе, появление новых элементов в которой происходит со средним временем τ, а удаление элементов – со средним временем l, среднее число элементов будет составлять $\frac{l}{\tau}$. Однако, в нашей системе есть ограничение числа передаваемых сообщений сверху: число фреймов F. Таким образом,

$$P = min\{\frac{l}{\tau}, F\} \tag{1}$$

Чтобы обеспечить $P = F$, будем задавать $\tau < \frac{l}{F}$. Таким образом, далее будем считать, что $P = F$, и исследовать зависимость интересующих нас величин не от P, а от $LR = \frac{P}{N} = \frac{F}{N}$.

3.2 Характеристики процессора

Вычисления проводились на процессоре Intel Core i5 3337U.

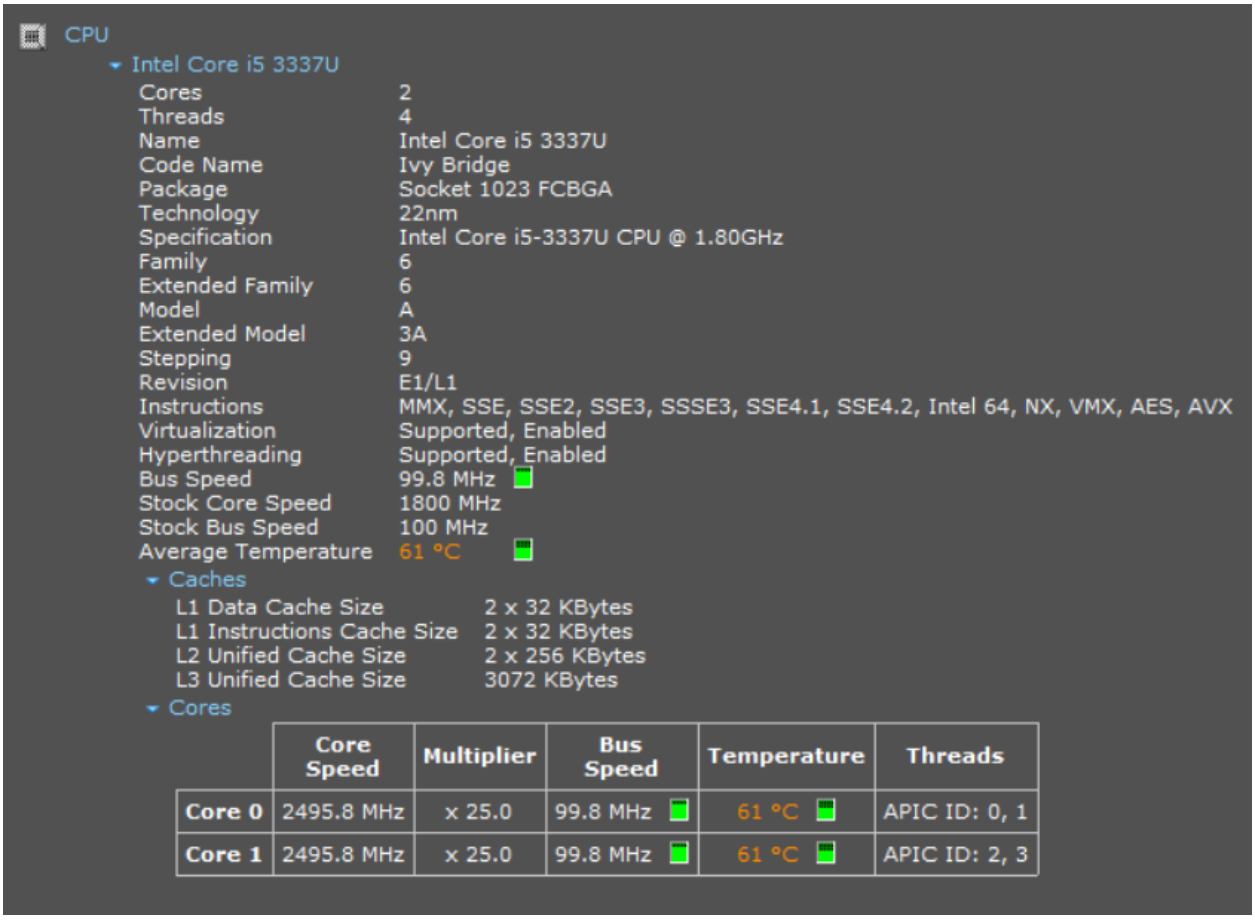

Рис. 1: Характеристики процессора

4 Исследование зависимости latency и throughput от числа нод и загруженности нод

4.1 Параметры

Вероятность успеха доставки сообщения $p = 0.85$.

THT (token holding time) = `Double.MAX_VALUE` (без ограничения сверху на время доставки сообщения).

N принимало значения `{5, 10, 15, 25, 50, 65, 85, 100}`.

LR принимало значения `{0.2, 0.4, 0.6, 0.8, 1.0}`.

4.2 Результаты

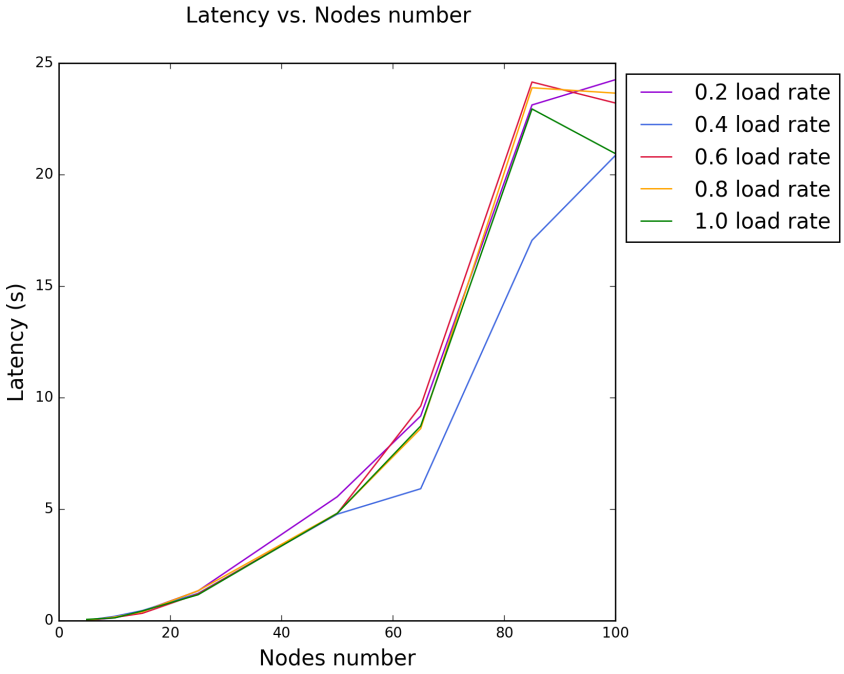

Рис. 2: Зависимость latency от числа нод N

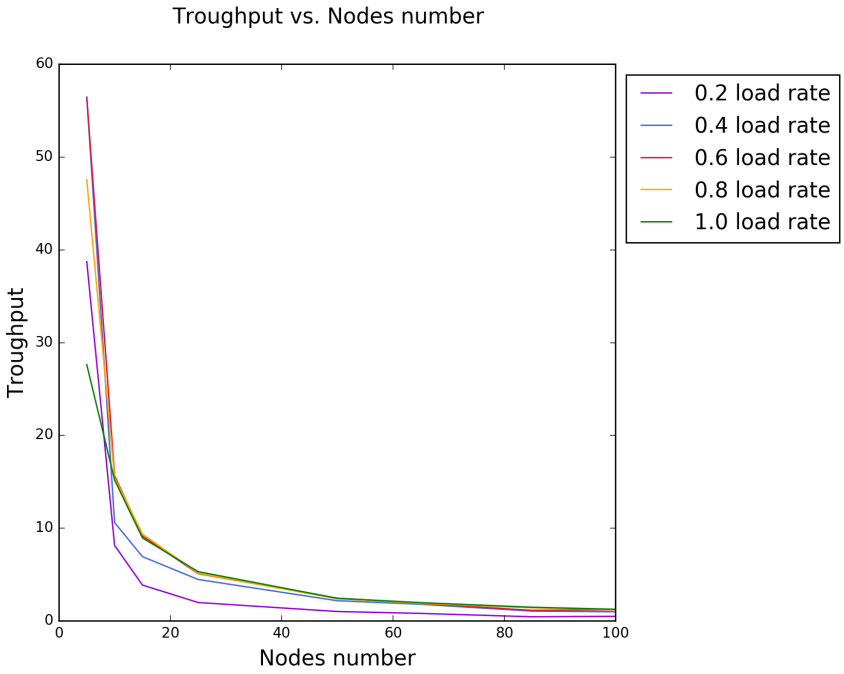

Рис. 3: Зависимость throughput от числа нод N

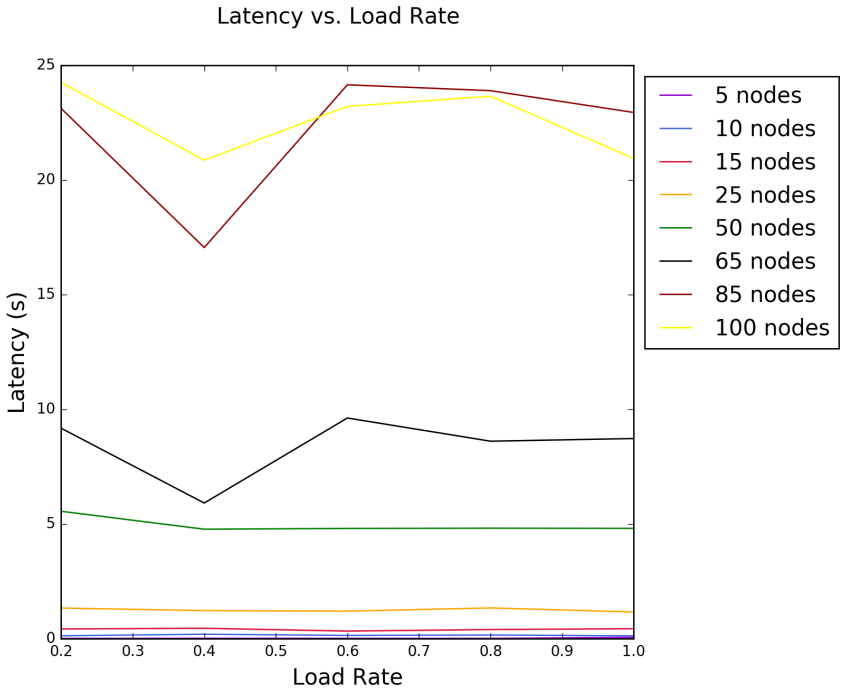

Рис. 4: Зависимость latency от загрузки LR

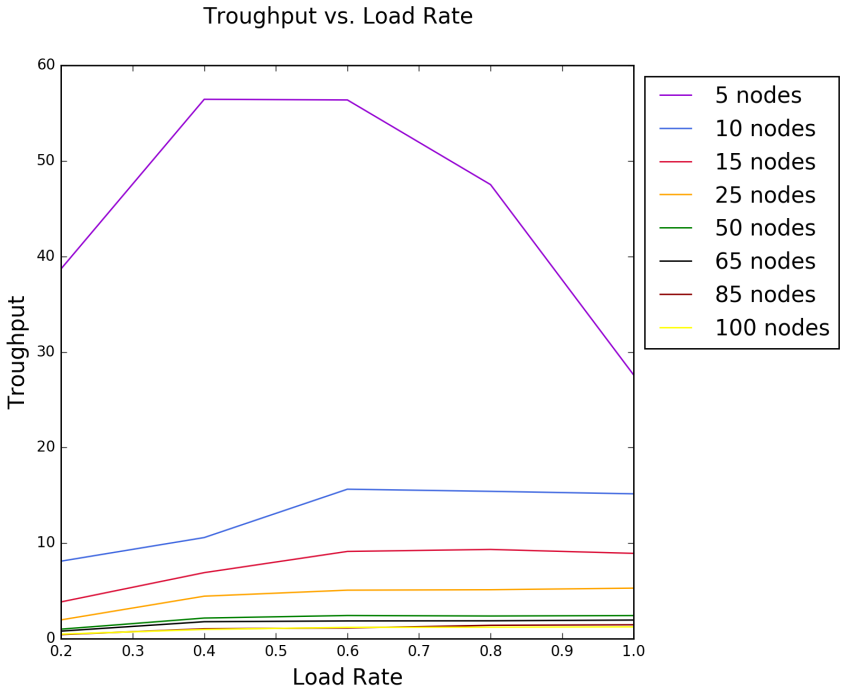

Рис. 5: Зависимость throughput от загрузки LR

4.3 Выводы

В соответствии с результатами, на процессоре с двумя ядрами выгоднее использовать меньшее число узлов и с точки зрения latency, и с точки зрения throughput.

Для большого числа нод (65, 85, 100) с точки зрения latency выгоднее одновременно пересылать $0.4 * N$ сообщений.

Для $N = 5$ число пересылаемых сообщений $0.4 * N$ также оказывается выгодным с точки зрения throughput.

5 Исследование зависимости latency и throughput от token holding time при различных значениях загруженности ноды

5.1 Параметры

Вероятность успеха доставки сообщения $p = 0.75$.

Число нод $N = 15$.

Число фреймов принимало значения от 1 до 4 (недогруженный режим) и от 12 до 15 (перегруженный режим), т.е. LR принимало значения `{0.067, 0.133, 0.2, 0.267}` и `{0.8, 0.867, 0.933, 1}`.

THT (token holding time) определялось следующим образом. Сначала для определения l – среднего времени доставки сообщений – при выбранных N и LR производился калибровочный запуск с THT = `Double.MAX_VALUE` (без ограничения сверху на время доставки сообщения). Затем THT определялся следующим образом: l умножался на множитель из перечня: `{0.7, 1.0, 1.5, 5.0, 10, 15, 20}`, который указан на графиках.

5.2 Результаты

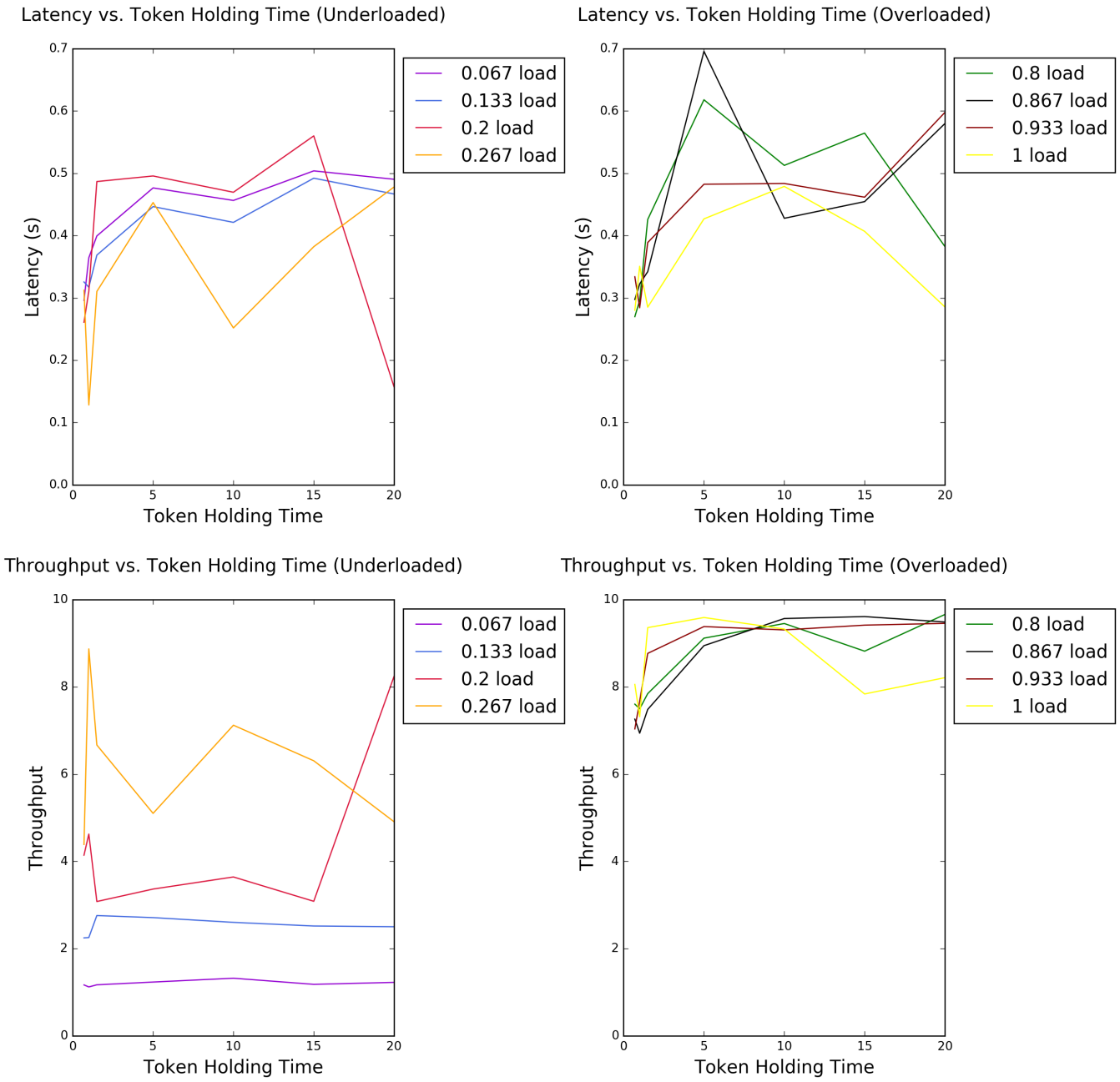

Рис. 6: Latency and Throughput vs. THT (Underloaded and Overloaded case)

5.3 Выводы

Для недогруженного режима, скорее всего, будет выгодно взять множитель при l, описанный в разделе "Основные параметры и метрики равным 0 (совершенно запретить повторные отправки).

Для перегруженного режима увеличение множителя даёт улучшение throughput, но ухудшение latency.

Ссылки

[1] http://web.chandra.ac.th/rawin/03-TokenRing.pdf

[2] https://en.wikipedia.org/wiki/Token_ring

[3] http://searchnetworking.techtarget.com/definition/Token-Ring

[4] http://protocols.netlab.uky.edu/~calvert/classes/571/lectureslides/TokenRing.pdf