Мини-статья

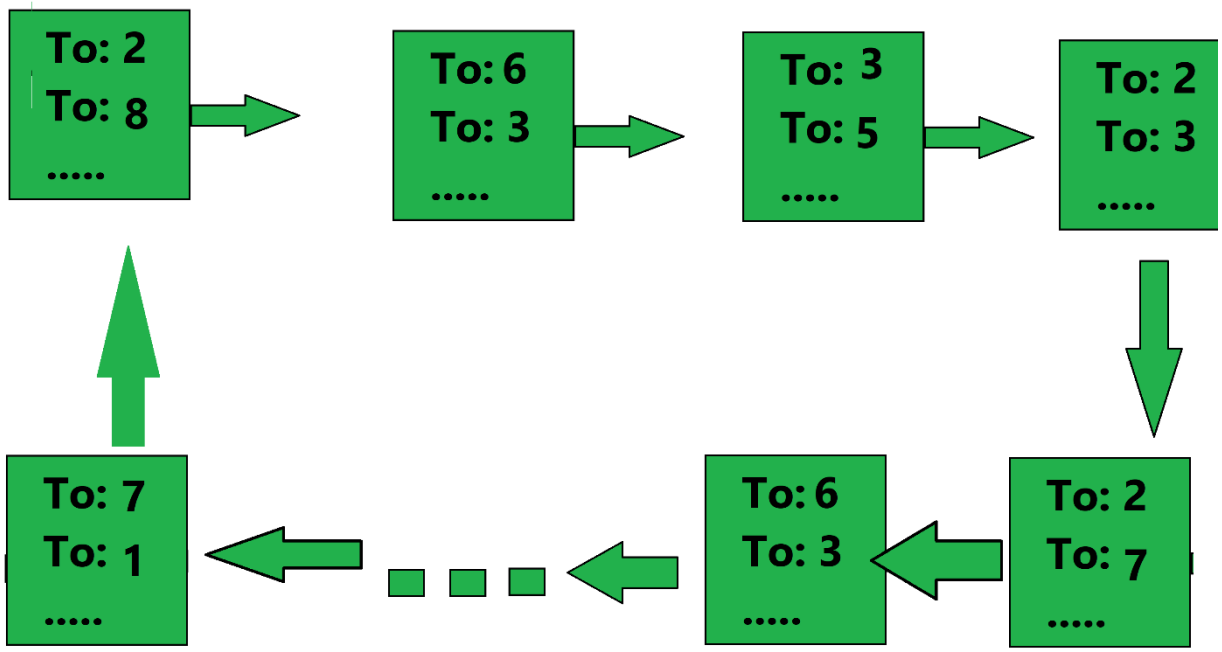
Задача

Исследовать пропускную способность сети (throughput) и характерное время задержки (latency) в зависимости от количества узлов N и количества пакетов P (1...N), находящихся в транзите одновременно.
Попытаться оптимизировать (улучшить) throughput или latency как в целом так и для отдельно взятых конкретных режимов (недогруженная сеть, перегруженная сеть) и исследовать влияние оптимизаций для одного режима на весь спектр режимов.

Throughput сети – количество пакетов, которое прошло за секунду.
Latency сети – время прохождения одного пакета

Исследование

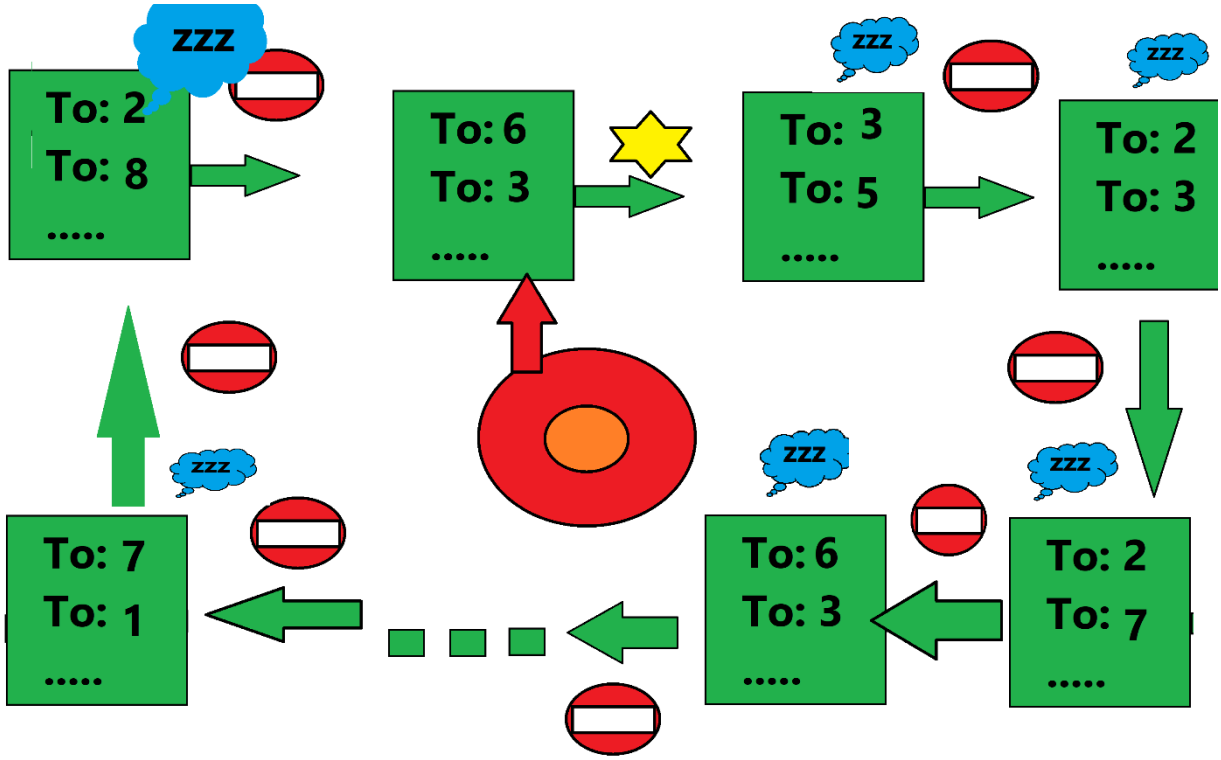
Windows10,Intel Core i5, 8 логических ядер.
Для исследования throughput и latency для пакетов находящихся одновременно в транзите, был использован такой механизм. Каждый узел 1..N нагружаем M пакетами. После чего запускаем работу все узлы. И того у нас в транзите: N x M пакетов в транзите одновременно.
Замеры проводились с помощью библиотеки JMH.



Базовая схема

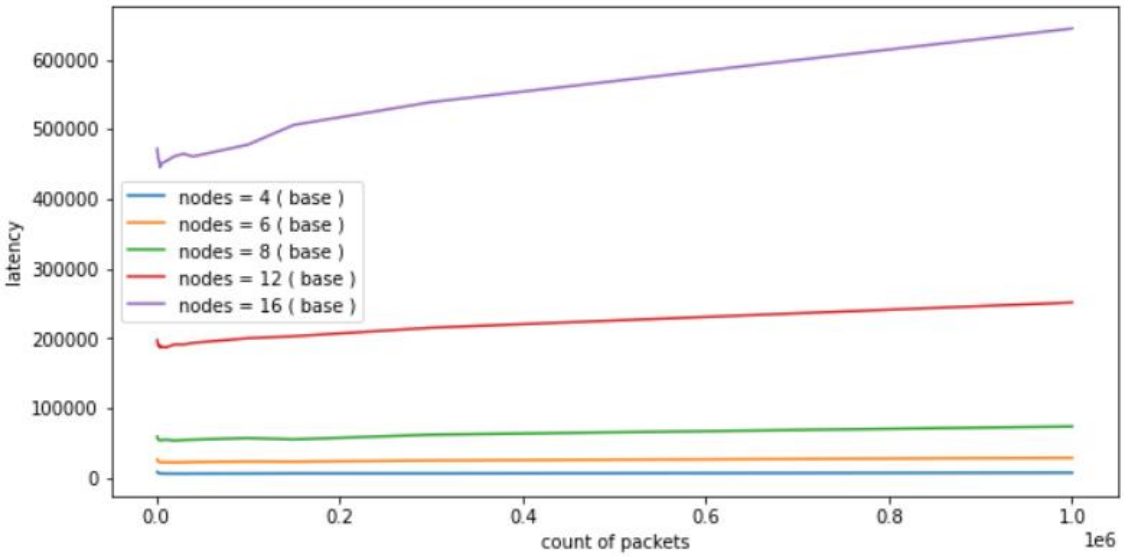
Обычная реализация предполагала, что у нас есть общий флаг, на котором мы синхронизируемся. Только один узел может перемещать информацию по сети в заданный промежуток времени. Не будем утилизировать цпу, добавим wait.

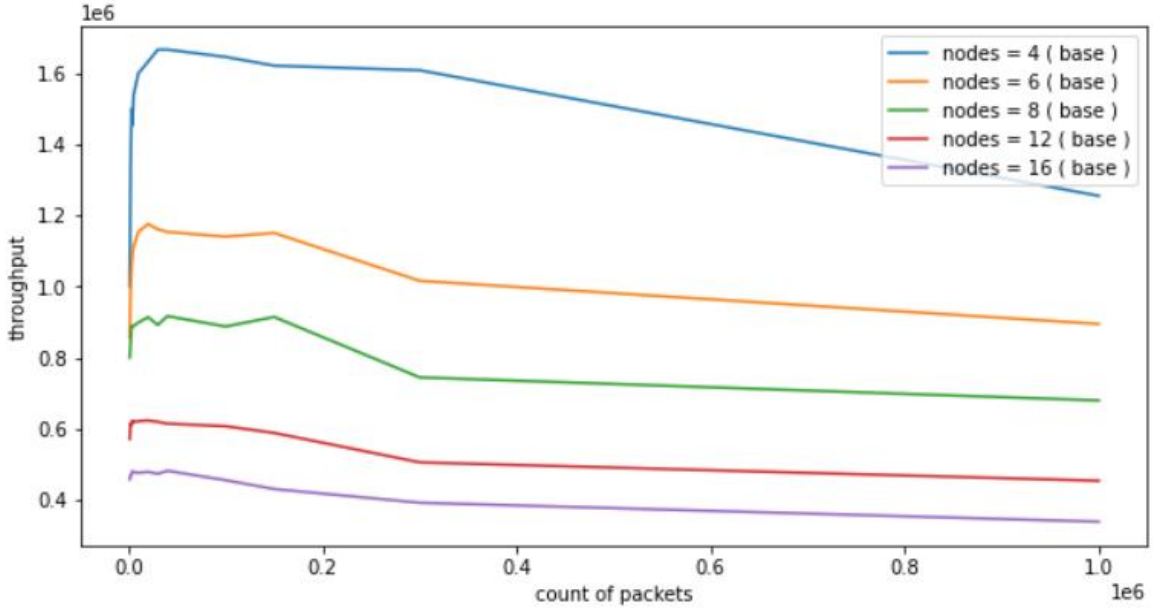
Схема:

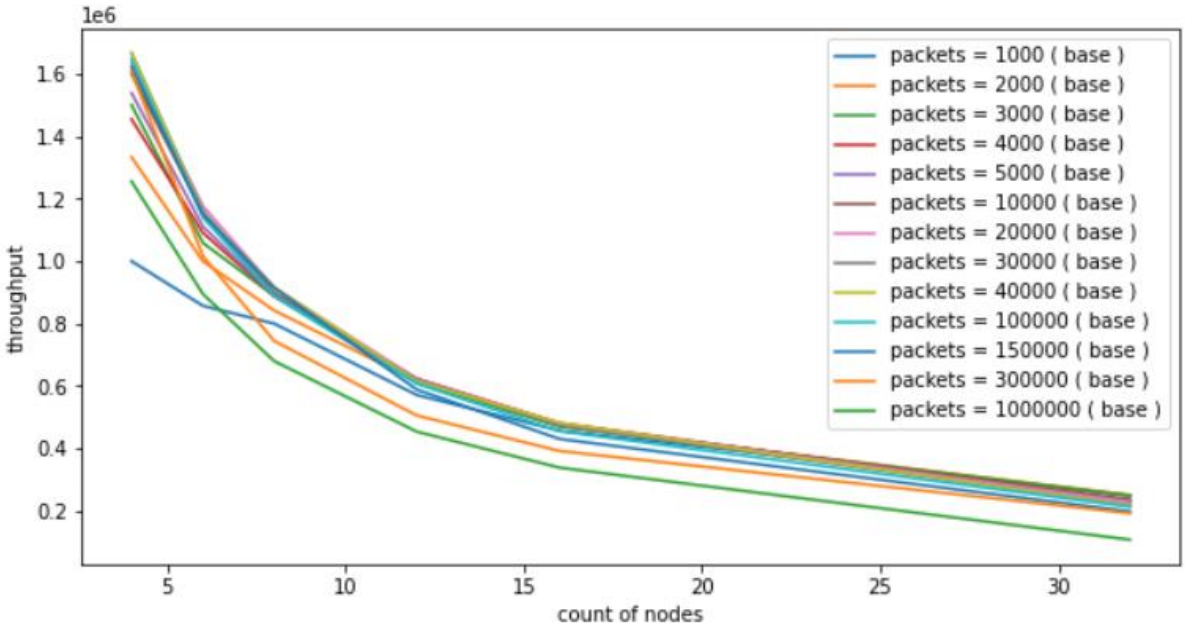


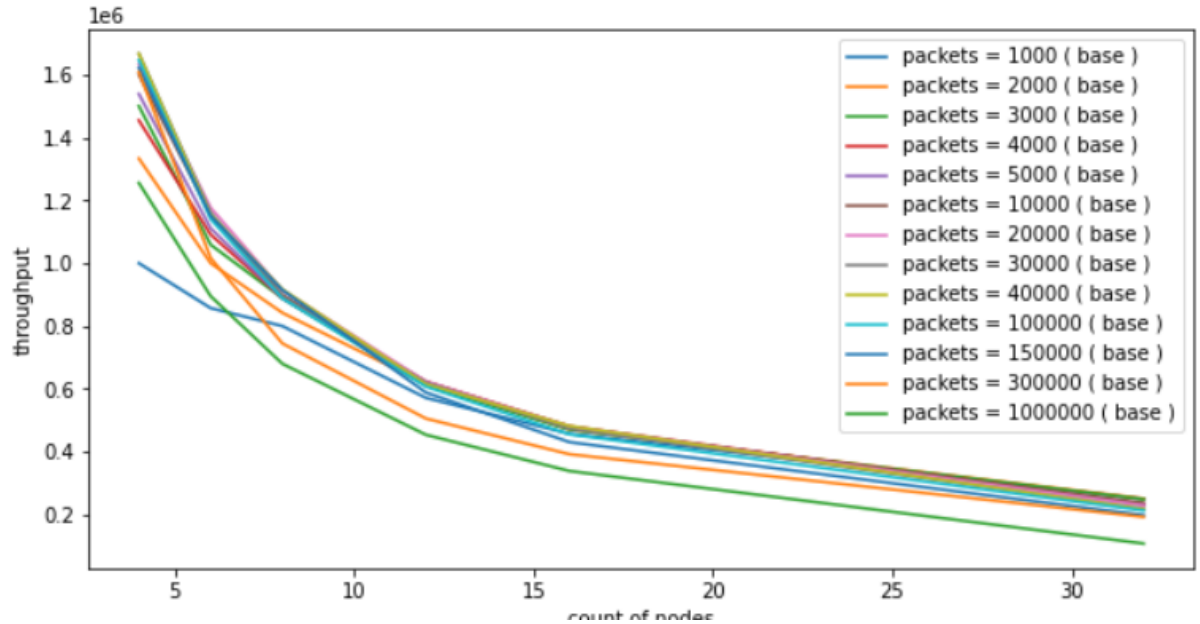
Графики:

Можно заметить большой разрыв между графиками, где у нас 8 и 12 узлов. Такое происходит из-за того, что ядер на устройстве – 8. Поэтому, когда потоков больше, происходит резкое замедление работы программы, соответственно увеличивается latency


600000
500000
400000
300000
200000
100000
0
latency
nodes = 4 (base)
nodes = 6 (base)
nodes = 8 (base)
nodes = 12 (base)
nodes = 16 (base)
0.0
0.2
0.4
0.6
0.8
1.0
1e6
count of packets


1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
throughput
nodes = 4 (base)
nodes = 6 (base)
nodes = 8 (base)
nodes = 12 (base)
nodes = 16 (base)
0.0
0.2
0.4
0.6
0.8
1.0
1e6
count of packets



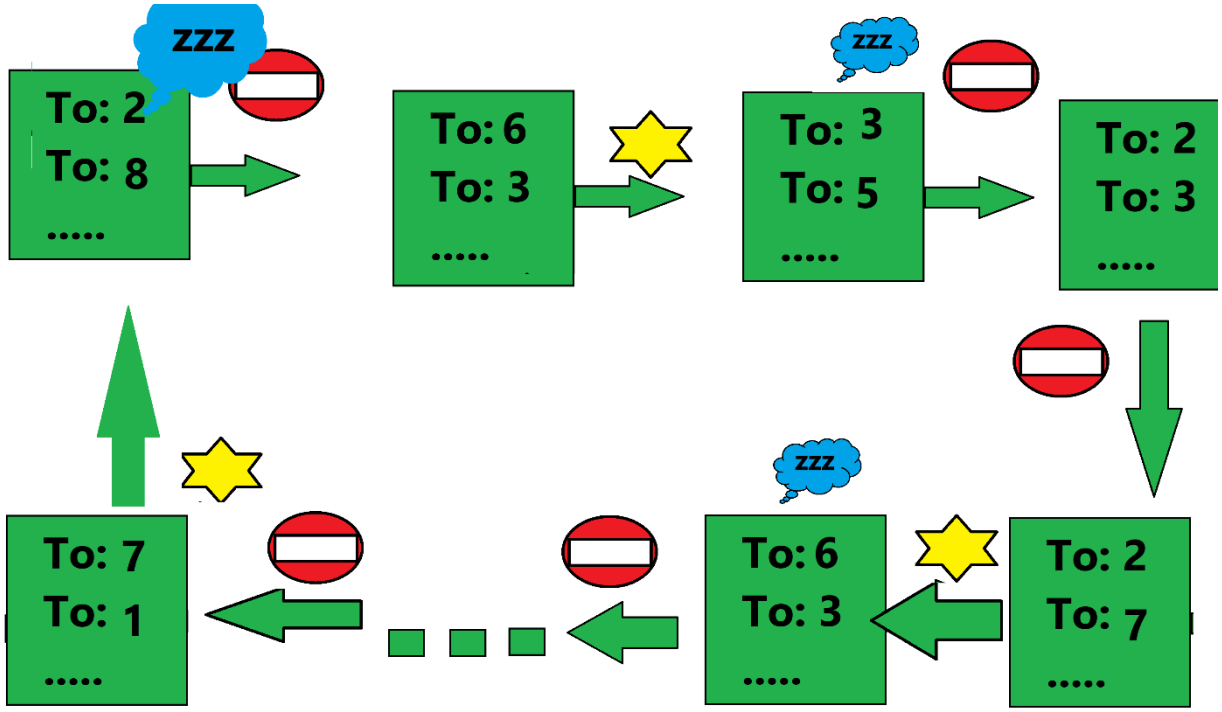


Оптимизация 1

Теперь пусть каждый узел имеет и видит только 2 лока, ведь только ими он пользуется. Теперь все потоки могут работать в произвольном режиме, нет общей точки синхронизации. У нас есть свой лок, который мы захватываем, чтобы вытащить из очереди элемент или делаем wait на нем, чтобы ждать появления нового пакета. Также есть лок соседнего по часовой стрелке узла. Мы его захватываем, чтобы засунуть в него пакет, после чего делаем на нем notify.

Таким образом хотелось повысить качество всех режимов.

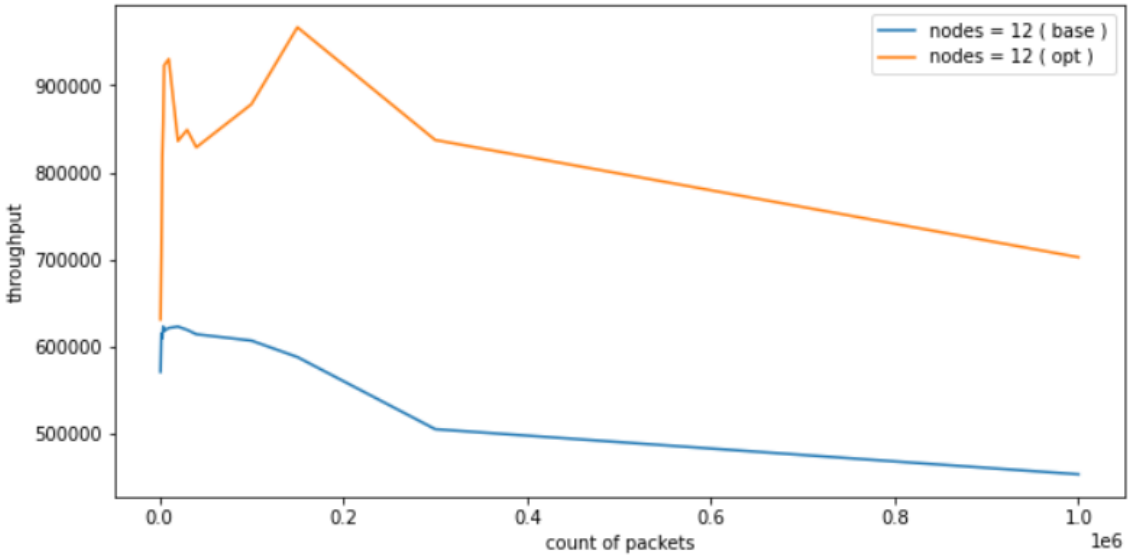
Схема:

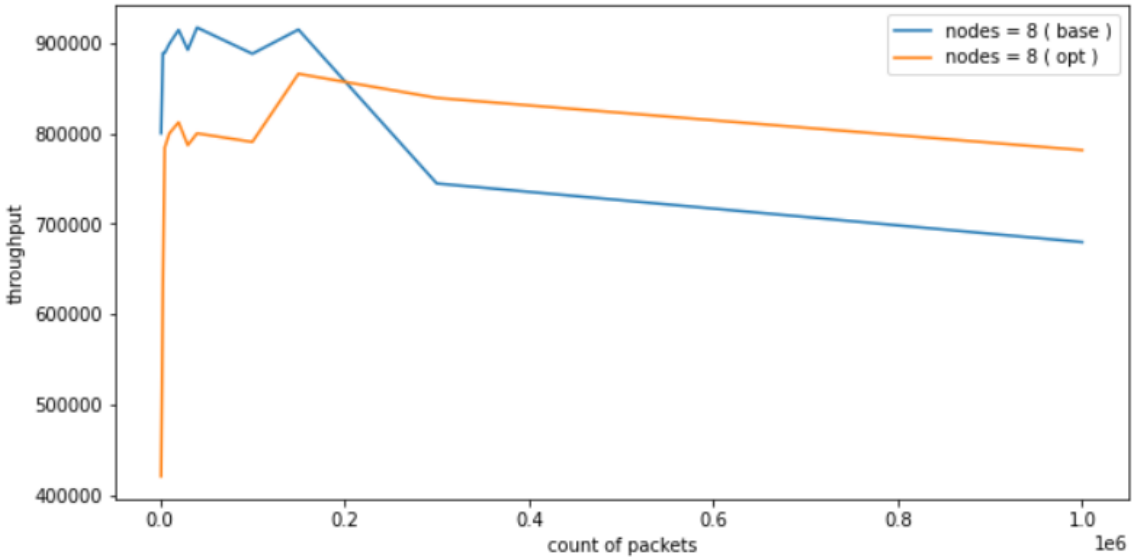


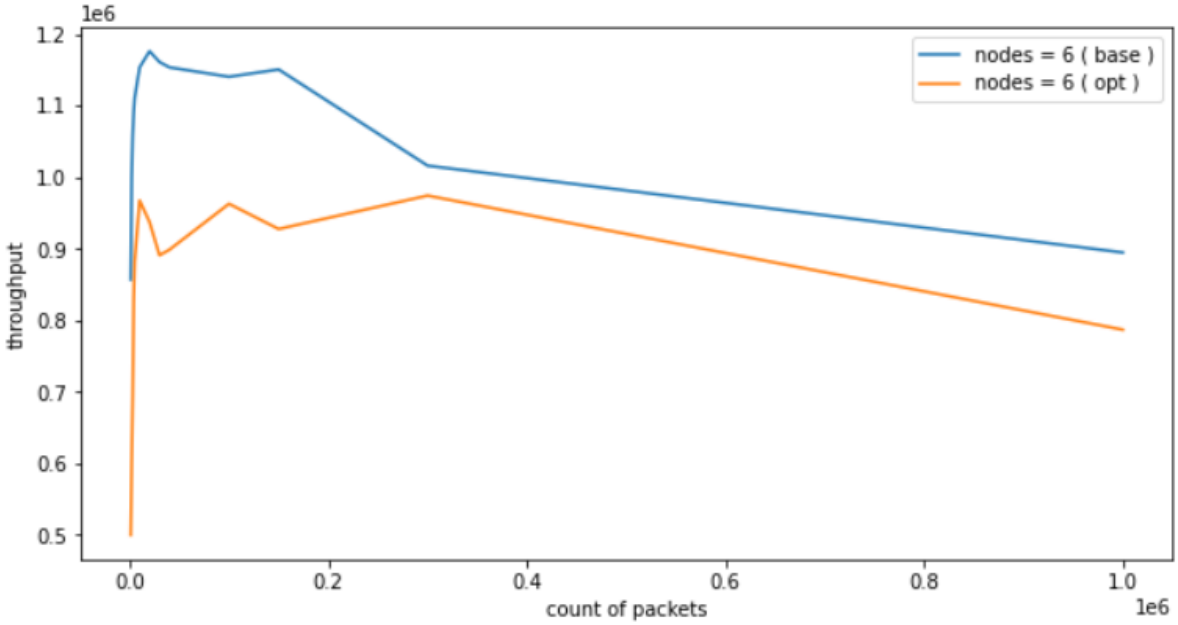
Графики:

Throughput

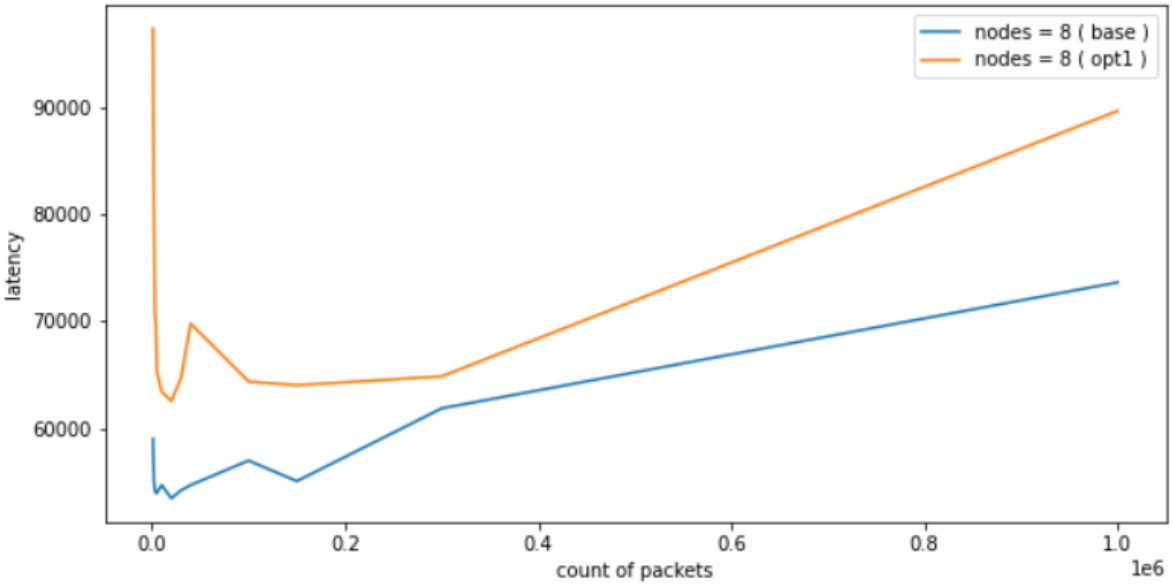
Когда узлов больше, чем ядер, то оптимизация улучшает качество сети. Иначе происходит замедление, из-за дополнительных синхронизаций.

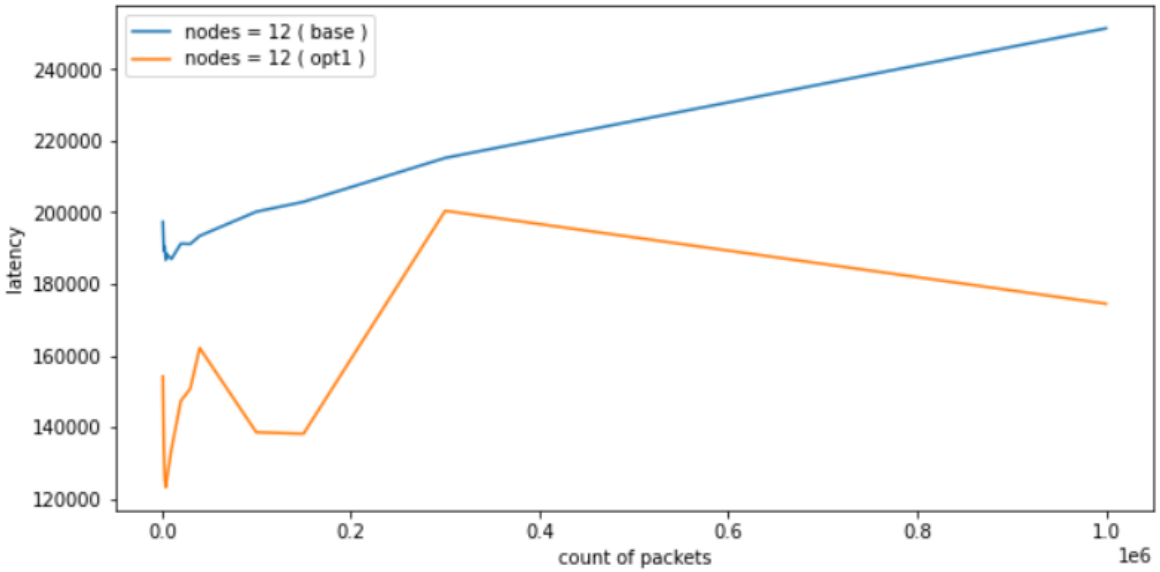

nodes = 12 (base)
nodes = 12 (opt)
throughput
900000
800000
700000
600000
500000
0.0
0.2
0.4
0.6
0.8
1.0
1e6
count of packets


nodes = 8 (base)
nodes = 8 (opt)
throughput
900000
800000
700000
600000
500000
400000
0.0
0.2
0.4
0.6
0.8
1.0
1e6
count of packets

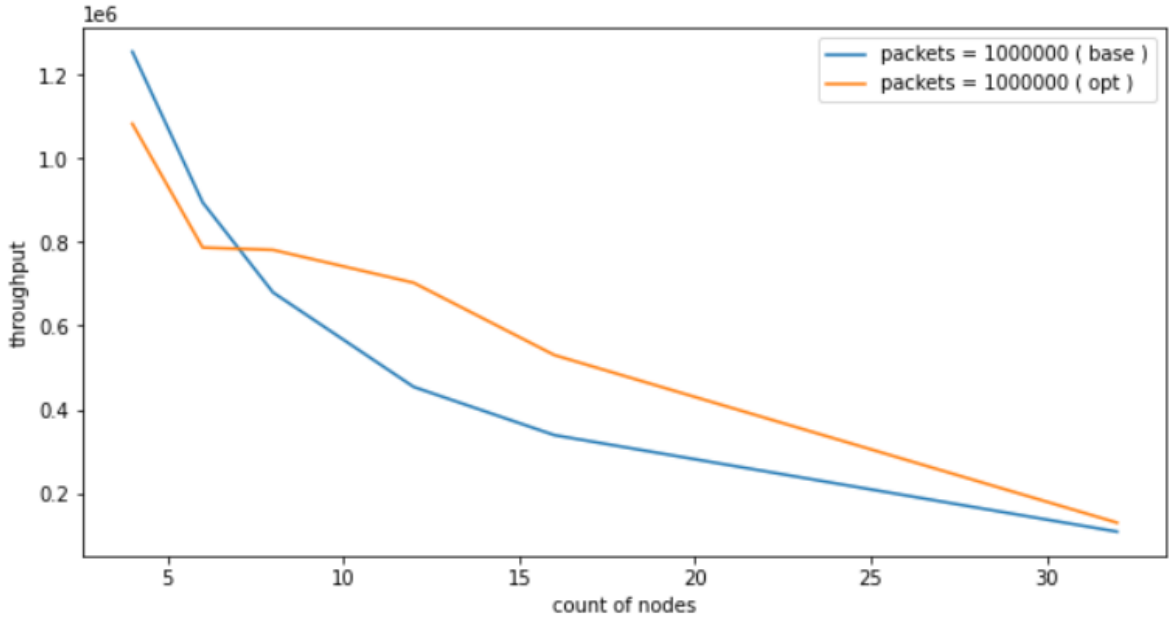

1e6
nodes = 6 (base)
nodes = 6 (opt)
throughput
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0
1e6
count of packets

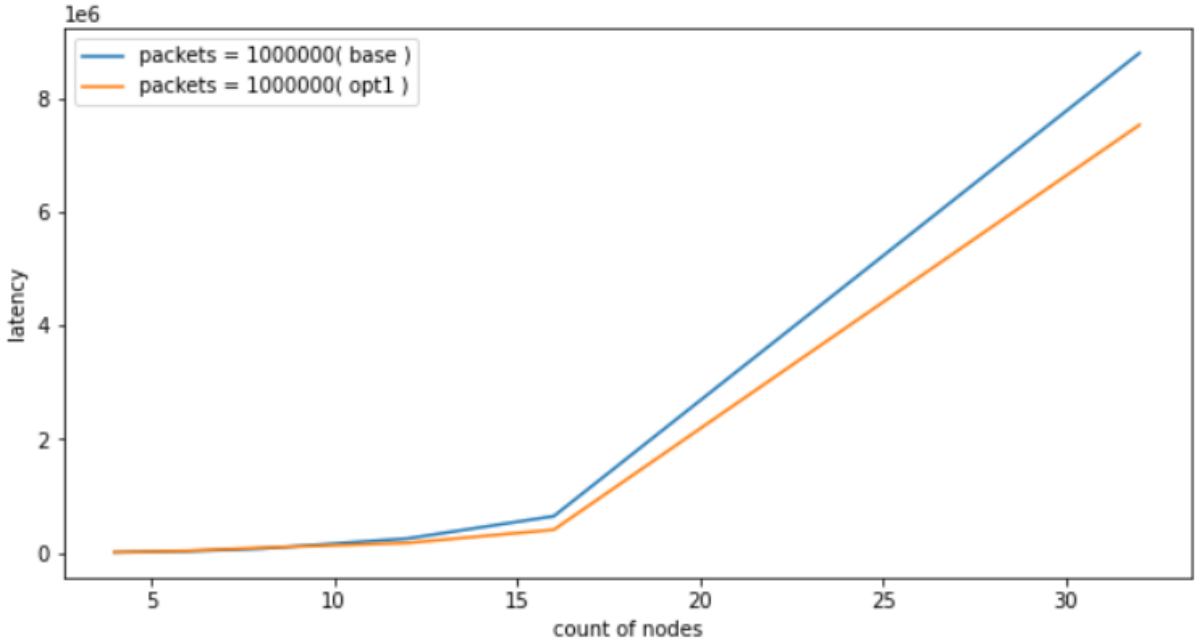
Latency





На следующих графиках видно, что и на latency и на throughput существенно влияет количество узлов. Начиная с 16 узлов происходит резкое увеличение latency. Throughput же уменьшается очень резко, когда увеличиваем количество узлов с 4 до 8.



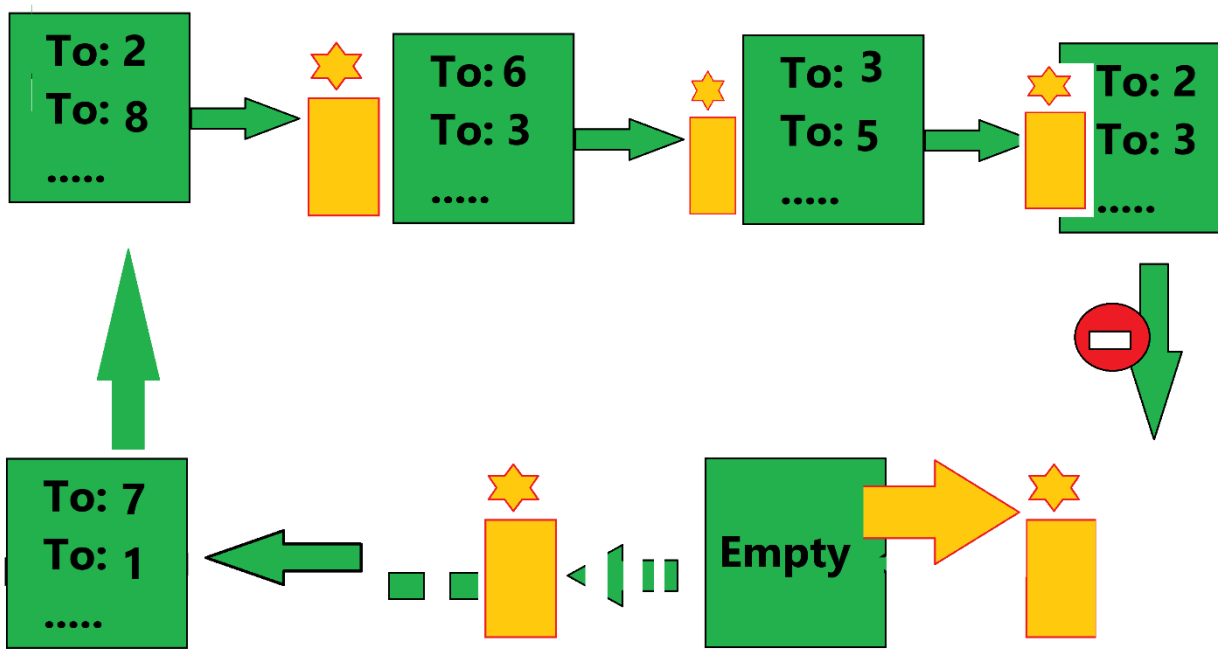


Как видно вторая оптимизация существенно повышает throughput, так как мы уже можем перемещать не синхронизируясь со всеми.

Оптимизация 2

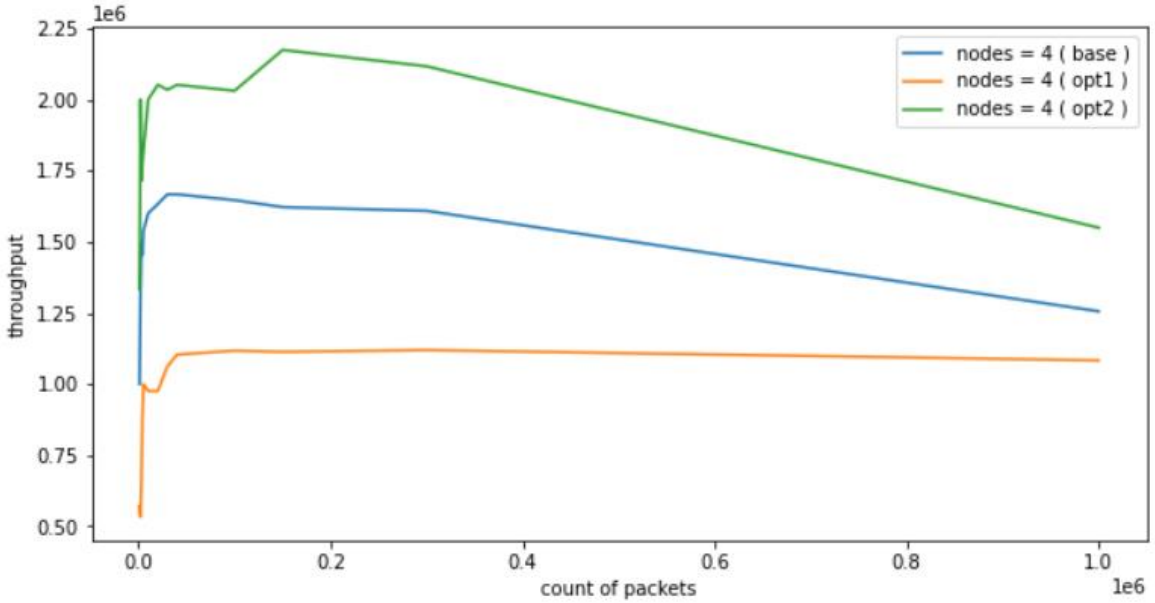
Давайте добавим новую очередь, в которую пишут ноды, чтобы передать данные. Когда наша внутренняя очередь будет пуста, будем блокировать эту очередь и вытаскивать данные из неё

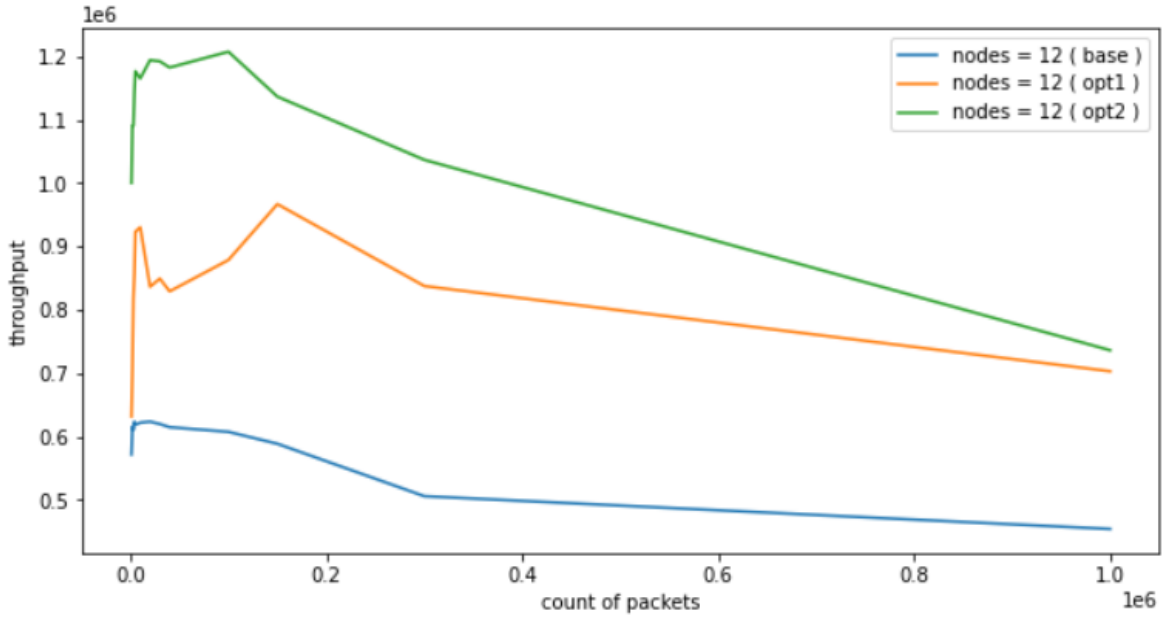
Схема:



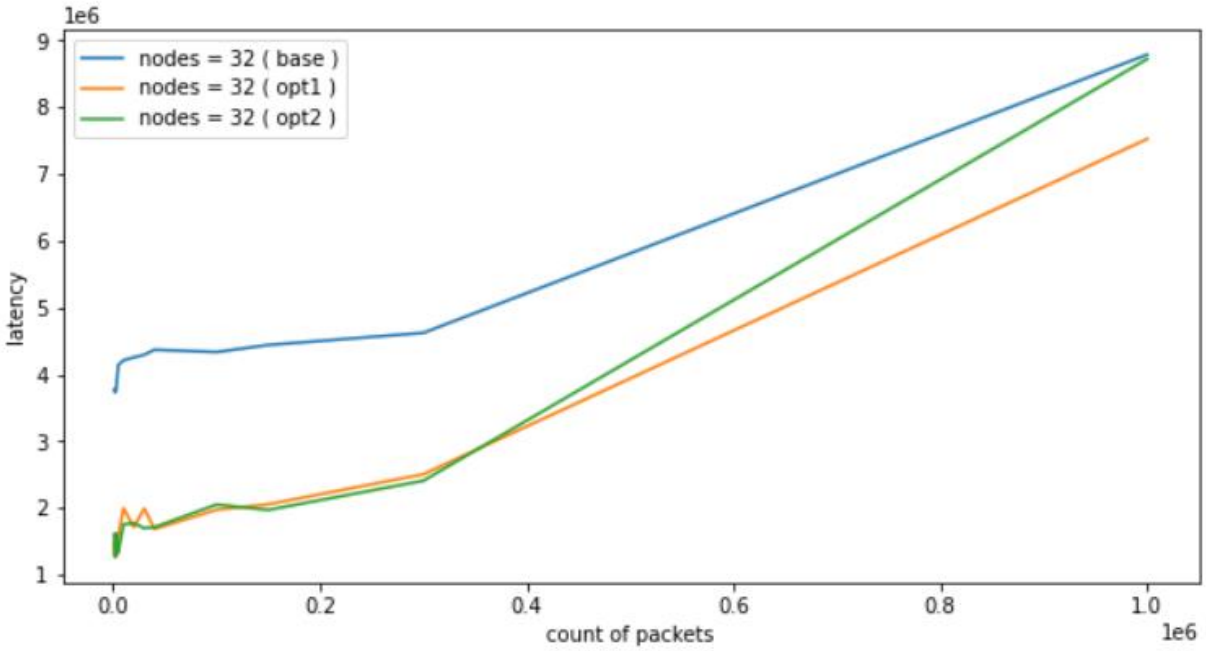
Графики:

Обгоняем даже когда мало узлов

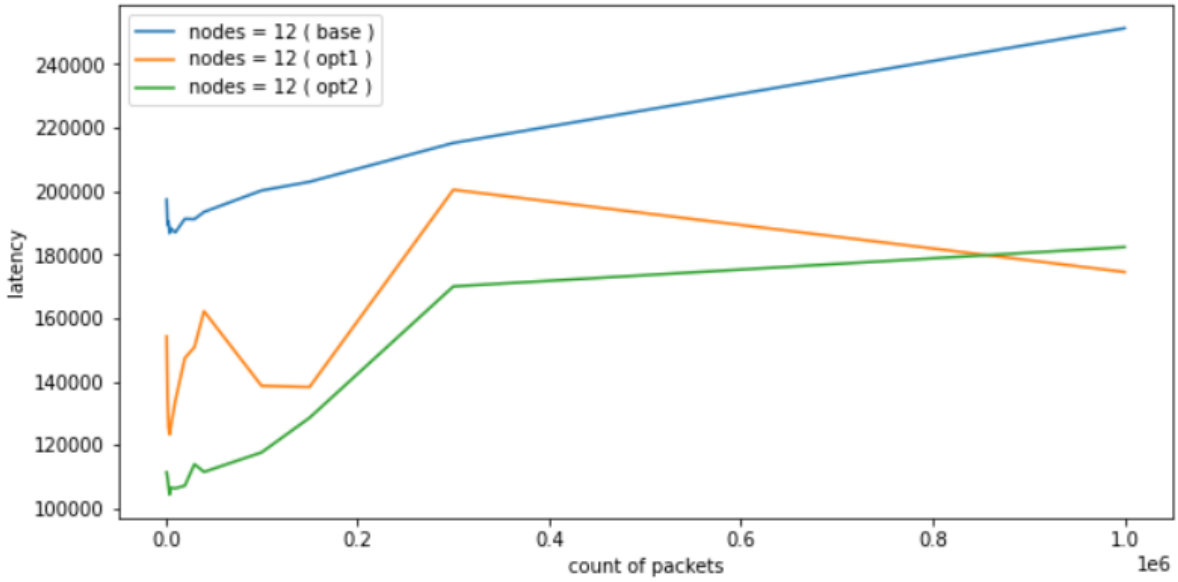


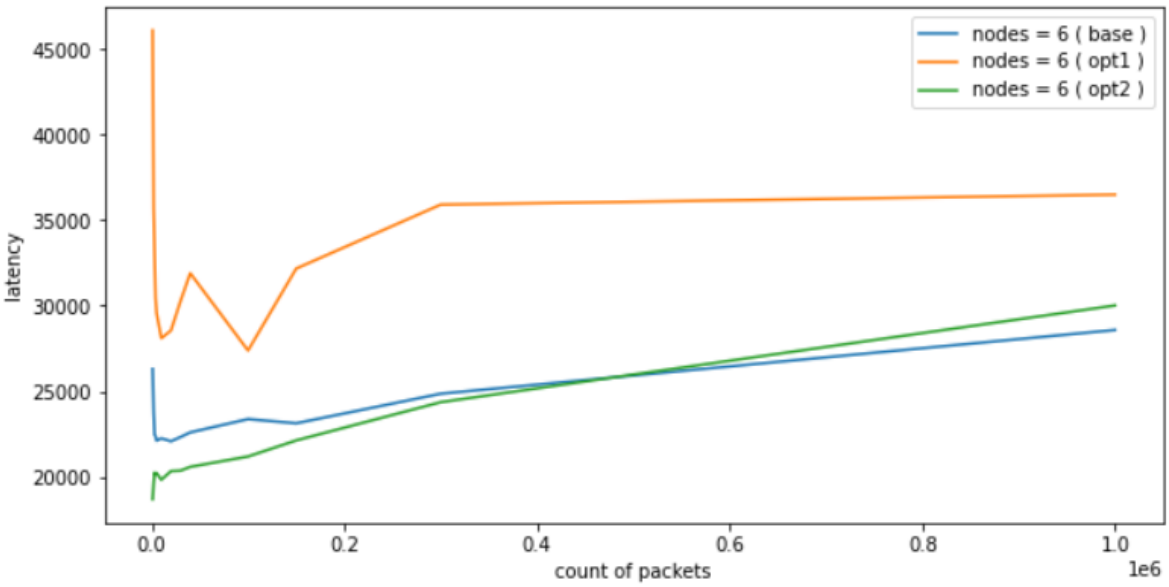


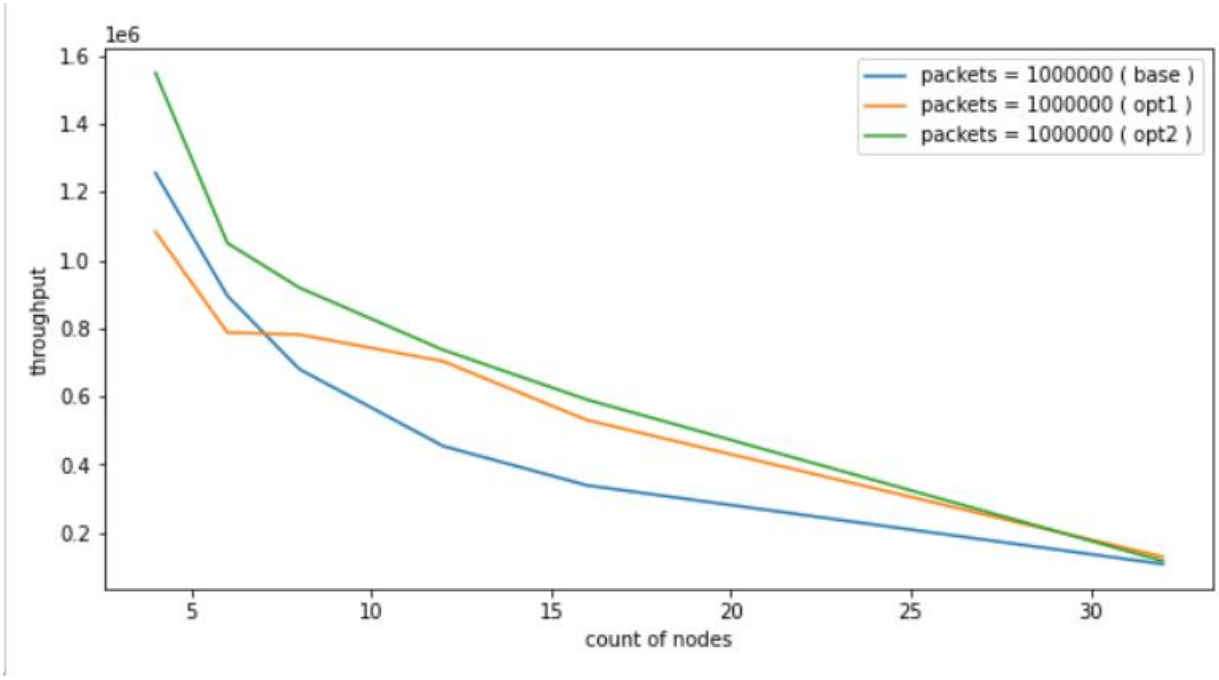
Но мы значительно проигрываем в latency теперь. Потому что пакет очень долго может лежат в очереди и ждать, пока мы его возьмем

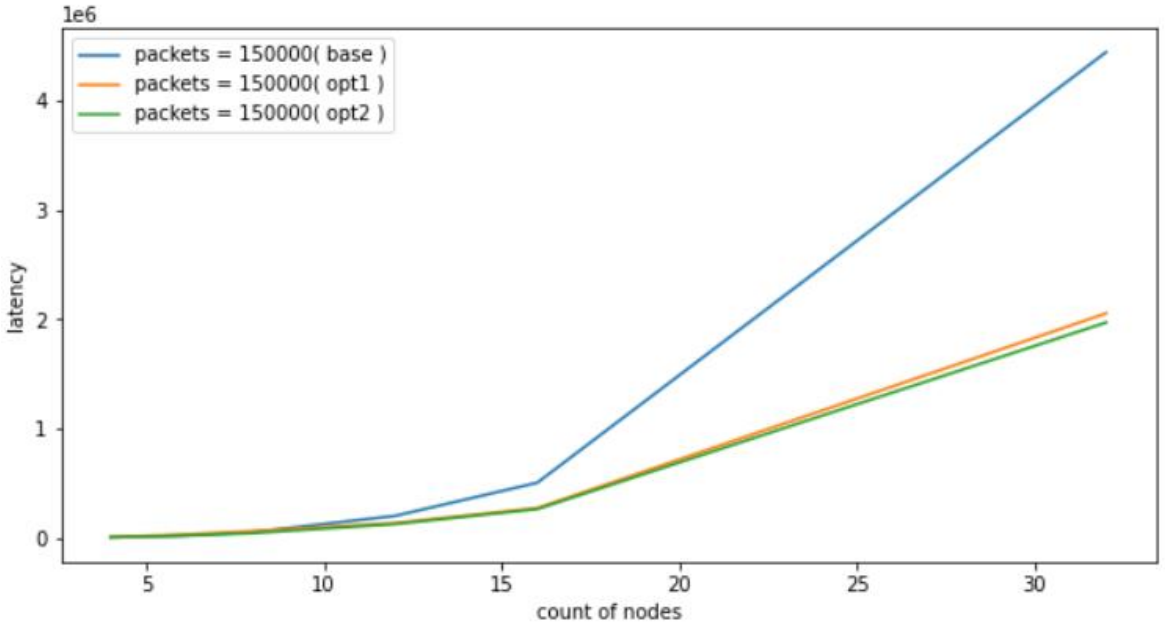


Но когда узлов не так много, наша оптимизация работает лучше, чем оптимизация 1.


nodes = 12 (base)
nodes = 12 (opt1)
nodes = 12 (opt2)
latency
count of packets
1e6


nodes = 6 (base)
nodes = 6 (opt1)
nodes = 6 (opt2)
latency
count of packets
1e6





Вывод:

Были исследованы throughput и latency, также проверены влияние оптимизаций.