Лабораторная работа №2

Исследование протокола TCP и алгоритма управления очередью RED

Ибатулина Дарья Эдуардовна, НФИбд-01-22

Содержание

# 1 Цель работы

Исследовать протокол TCP и алгоритм управления очередью RED.

# 2 Задание

1. Выполнить пример с дисциплиной RED;
2. Изменить в модели на узле s1 тип протокола TCP с Reno на NewReno, затем на Vegas. Сравнить и пояснить результаты;
3. Внести изменения при отображении окон с графиками (изменить цвет фона, цвет траекторий, подписи к осям, подпись траектории в легенде).

# 3 Теоретическое введение

Протокол управления передачей (Transmission Control Protocol, TCP) имеет средства управления потоком и коррекции ошибок, ориентирован на установление соединения.

* Флаг Указатель срочности (Urgent Pointer, URG) устанавливается в 1 в случае использования поля Указатель на срочные данные.
* Флаг Подтверждение (Acknowledgment, ACK) устанавливается в 1 в случае, если поле Номер подтверждения (Acknowledgement Number) содержит данные. В противном случае это поле игнорируется.
* Флаг Выталкивание (Push, PSH) означает, что принимающий стек TCP должен немедленно информировать приложение о поступивших данных, а не ждать, пока буфер заполнится.
* Флаг Сброс (Reset, RST) используется для отмены соединения из-за ошибки приложения, отказа от неверного сегмента, попытки создать соединение при отсутствии затребованного сервиса.
* Флаг Синхронизация (Synchronize, SYN) устанавливается при инициировании соединения и синхронизации порядкового номера.
* Флаг Завершение (Finished, FIN) используется для разрыва соединения. Он указывает, что отправитель закончил передачу данных.

Управление потоком в протоколе TCP осуществляется при помощи скользящего окна переменного размера: поле Размер окна (Window) (длина 16 бит) содержит количество байт, которое может быть послано после байта, получение которого уже подтверждено; еслизначениеэтогополяравнонулю,этоозначает,чтовсебайты,вплоть до байта с номером Номер подтверждения - 1, получены, но получатель отказывается принимать дальнейшие данные; разрешение на дальнейшую передачу может быть выдано отправкой сегмента с таким же значением поля Номер подтверждения и ненулевым значением поля Размер окна.

Регулирование трафика в TCP: контроль доставки — отслеживает заполнение входного буфера получателя с помощью параметра Размер окна (Window); контроль перегрузки — регистрирует перегрузку канала и связанные с этим потери, а также понижает интенсивность трафика с помощью Окна перегрузки (Congestion Window, CWnd) и Порога медленного старта (Slow Start Threshold, SSThreth).

В ns-2 поддерживает следующие TCP-агенты односторонней передачи:

* Agent/TCP
* Agent/TCP/Reno
* Agent/TCP/Newreno
* Agent/TCP/Sack1 — TCP с выборочным повтором (RFC2018)
* Agent/TCP/Vegas
* Agent/TCP/Fack — Reno TCP с «последующим подтверждением»
* Agent/TCP/Linux — TCP-передатчик с поддержкой SACK, который использует TCP сперезагрузкой контрольных модулей из ядра Linux

Односторонние агенты приёма:

* Agent/TCPSink
* Agent/TCPSink/DelAck
* Agent/TCPSink/Sack1
* Agent/TCPSink/Sack1/DelAck

Двунаправленный агент:

* Agent/TCP/FullTcp

TCPReno: - медленный старт (Slow-Start);

* контроль перегрузки (Congestion Avoidance);
* быстрый повтор передачи (Fast Retransmit);
* процедура быстрого восстановления (Fast Recovery);
* метод оценки длительности цикла передачи (Round Trip Time, RTT), используемой для установки таймера повторной передачи (Retransmission TimeOut, RTO).

Схема работы TCP Reno:

* размер окна увеличивается до тех пор, пока не произойдёт потеря сегмента (аналогично TCP Tahoe) – фаза медленного старта и фаза избежания перегрузки;
* алгоритм не требует освобождения канала и его медленного (slow-start) заполнения после потери одного пакета;
* отправитель переходит в режим быстрого восстановления, после получения некоторого предельного числа дублирующих подтверждений — отправитель повторяет передачу одного пакета и уменьшает окно перегрузки (cwnd) в два раза и устанавливает ssthresh\_ в соответствии с этим значением.

# 4 Выполнение лабораторной работы

## 4.1 Пример с дисциплиной RED

**Постановка задачи**

Описание моделируемой сети:

* сеть состоит из 6 узлов;
* между всеми узлами установлено дуплексное соединение с различными пропускной способностью и задержкой 10 мс;
* узел r1 использует очередь с дисциплиной RED для накопления пакетов, максимальный размер которой составляет 25;
* TCP-источники на узлах s1 и s2 подключаются к TCP-приёмнику на узле s3;
* генераторы трафика FTP прикреплены к TCP-агентам.

Требуется разработать сценарий, реализующий модель согласно описанию, построить в Xgraph график изменения TCP-окна, график изменения длины очереди и средней длины очереди. Запускаю скрипт, выполняющий данную задачу (рис. [[1](#fig:001)]).

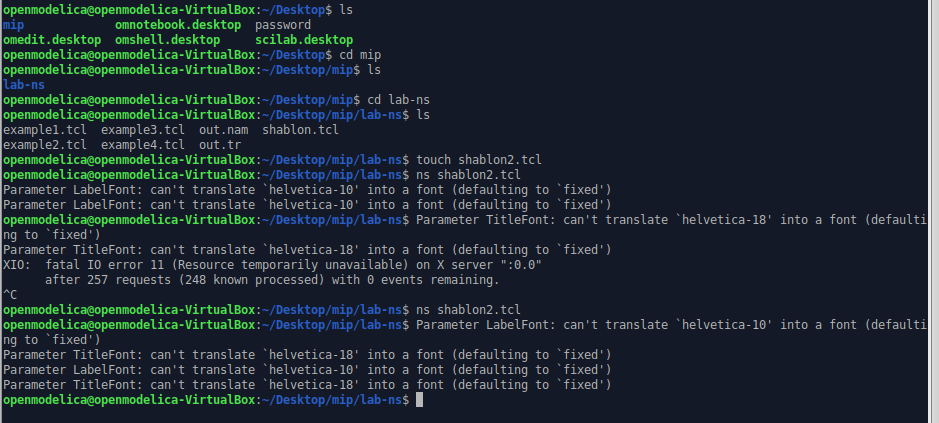


Figure 1: Запуск скрипта

Скрипт приведен ниже:

# создание объекта Simulator  
set ns [new Simulator]  
  
# открытие на запись файла out.nam для визуализатора nam  
set nf [open out.nam w]  
  
# все результаты моделирования будут записаны в переменную nf  
$ns namtrace-all $nf  
  
# открытие на запись файла трассировки out.tr  
# для регистрации всех событий  
set f [open out.tr w]  
# все регистрируемые события будут записаны в переменную f  
$ns trace-all $f  
  
# Процедура finish:  
proc finish {} {  
 global tchan\_  
 # подключение кода AWK:  
 set awkCode {  
 {  
 if ($1 == "Q" && NF>2) {  
 print $2, $3 >> "temp.q";  
 set end $2  
 }  
 else if ($1 == "a" && NF>2)  
 print $2, $3 >> "temp.a";  
 }  
 }  
 set f [open temp.queue w]  
 puts $f "TitleText: red"  
 puts $f "Device: Postscript"  
 if { [info exists tchan\_] } {  
 close $tchan\_   
 }  
 exec rm -f temp.q temp.a  
 exec touch temp.a temp.q  
 exec awk $awkCode all.q  
 puts $f \"queue  
 exec cat temp.q >@ $f  
 puts $f \n\"ave\_queue  
 exec cat temp.a >@ $f  
 close $f  
 # Запуск xgraph с графиками окна TCP и очереди:  
 exec xgraph -bb -tk -x time -t "TCPRenoCWND" WindowVsTimeReno &  
 exec xgraph -bb -tk -x time -y queue temp.queue &  
 exit 0  
}  
  
# Формирование файла с данными о размере окна TCP:  
proc plotWindow {tcpSource file} {  
 global ns  
 set time 0.01  
 set now [$ns now]  
 set cwnd [$tcpSource set cwnd\_]  
 puts $file "$now $cwnd"  
 $ns at [expr $now+$time] "plotWindow $tcpSource $file"  
}  
  
# Узлы сети:  
set N 5  
for {set i 1} {$i < $N} {incr i} {  
 set node\_(s$i) [$ns node]  
}  
set node\_(r1) [$ns node]  
set node\_(r2) [$ns node]  
# Соединения:  
$ns duplex-link $node\_(s1) $node\_(r1) 10Mb 2ms DropTail  
$ns duplex-link $node\_(s2) $node\_(r1) 10Mb 3ms DropTail  
$ns duplex-link $node\_(r1) $node\_(r2) 1.5Mb 20ms RED  
$ns queue-limit $node\_(r1) $node\_(r2) 25  
$ns queue-limit $node\_(r2) $node\_(r1) 25  
$ns duplex-link $node\_(s3) $node\_(r2) 10Mb 4ms DropTail  
$ns duplex-link $node\_(s4) $node\_(r2) 10Mb 5ms DropTail  
  
# Агенты и приложения:  
set tcp1 [$ns create-connection TCP/Reno $node\_(s1) TCPSink $node\_(s3) 0]  
$tcp1 set window\_ 15  
set tcp2 [$ns create-connection TCP/Reno $node\_(s2) TCPSink $node\_(s3) 1]  
$tcp2 set window\_ 15  
set ftp1 [$tcp1 attach-source FTP]  
set ftp2 [$tcp2 attach-source FTP]  
  
# Мониторинг размера окна TCP:  
set windowVsTime [open WindowVsTimeReno w]  
set qmon [$ns monitor-queue $node\_(r1) $node\_(r2) [open qm.out w] 0.1];  
[$ns link $node\_(r1) $node\_(r2)] queue-sample-timeout;  
# Мониторинг очереди:  
set redq [[$ns link $node\_(r1) $node\_(r2)] queue]  
set tchan\_ [open all.q w]  
$redq trace curq\_  
$redq trace ave\_  
$redq attach $tchan\_  
  
  
# Добавление at-событий:  
$ns at 0.0 "$ftp1 start"  
$ns at 1.1 "plotWindow $tcp1 $windowVsTime"  
$ns at 3.0 "$ftp2 start"  
$ns at 10 "finish"  
  
# запуск модели  
$ns run

И вот, какой результат получился после запуска (рис. [[2](#fig:002)], [[3](#fig:003)]).

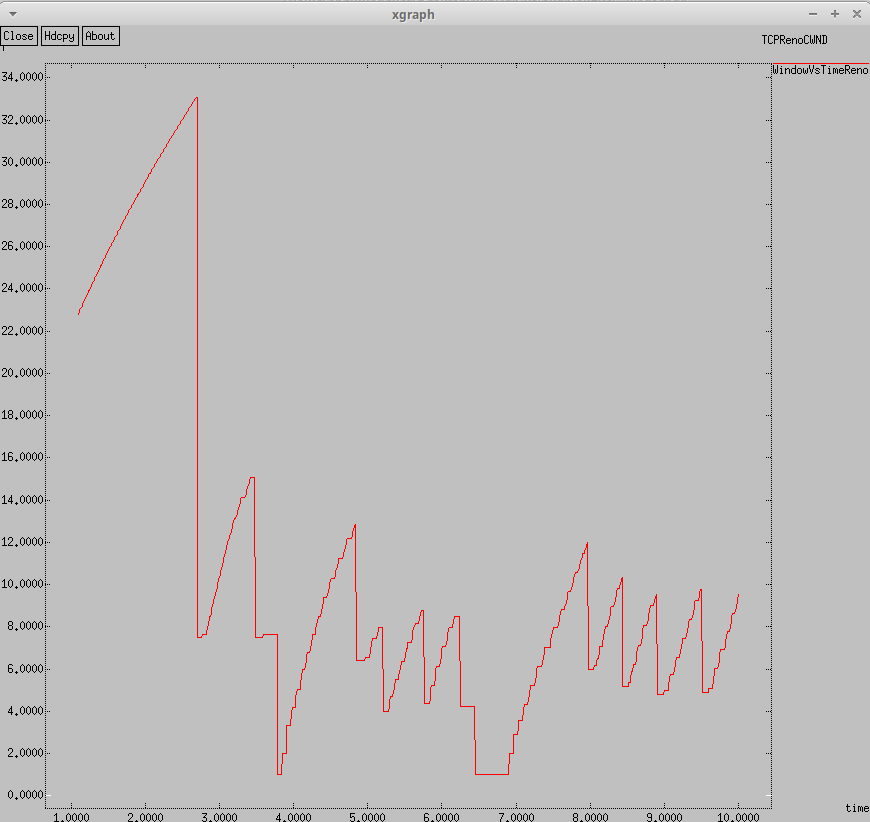


Figure 2: График динамики размера окна TCP

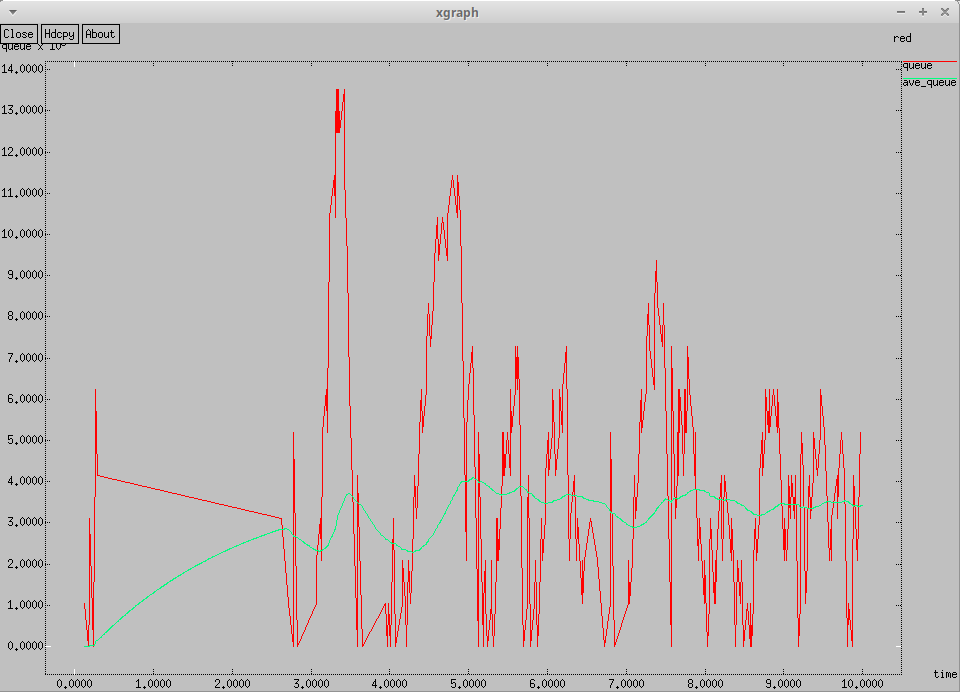


Figure 3: График динамики длины очереди и средней длины очереди

По графику видно, что средняя длина очереди находится в диапазоне от 2 до 4. Максимальная длина почти достигает значения 14.

## 4.2 Упражнение 1

Измените в модели на узле s1 тип протокола TCP с Reno на NewReno, затем на Vegas. Сравните и поясните результаты.

1. Заменим на NewReno тип протокола s1 (рис. [[4](#fig:004)]). Посмотрим, какой получился результат (рис. [[5](#fig:005)], [[6](#fig:006)])

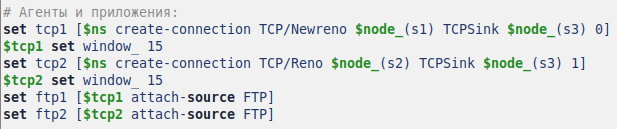


Figure 4: Изменение типа протокола на NewReno

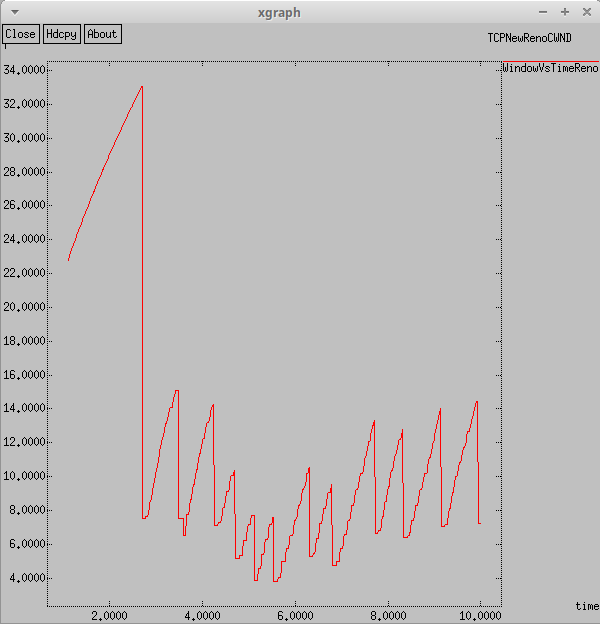


Figure 5: График динамики размера окна TCP

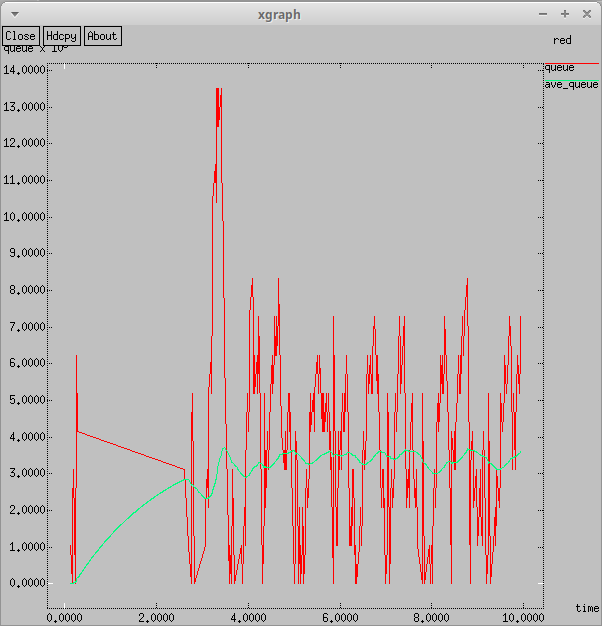


Figure 6: График динамики длины очереди и средней длины очереди

Так же, как было в графике с типом Reno значение средней длины очереди находится в пределах от 2 до 4, а максимальное значение длины не достигает 14. Графики достаточно похожи. В обоих алгоритмах размер окна увеличивается до тех пор, пока не произойдёт потеря сегмента.

1. Заменим на Vegas тип протокола s1 (рис. [[7](#fig:007)]). Посмотрим, какой получился результат (рис. [[8](#fig:008)], [[9](#fig:009)])



Figure 7: Изменение типа протокола на Vegas

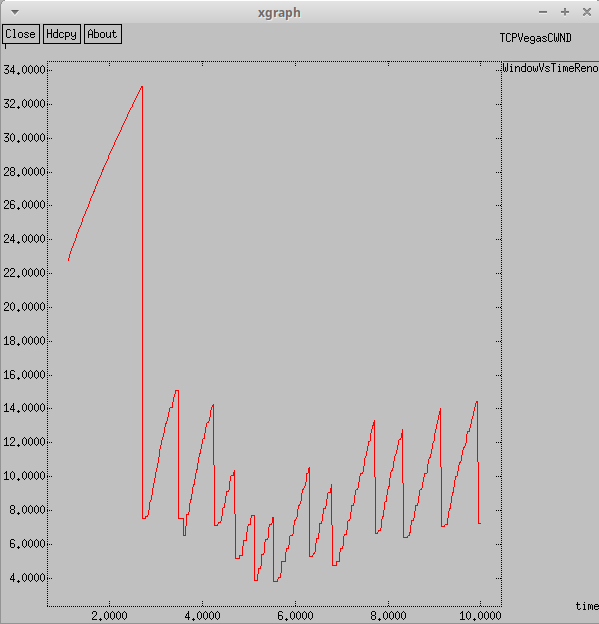


Figure 8: График динамики размера окна TCP

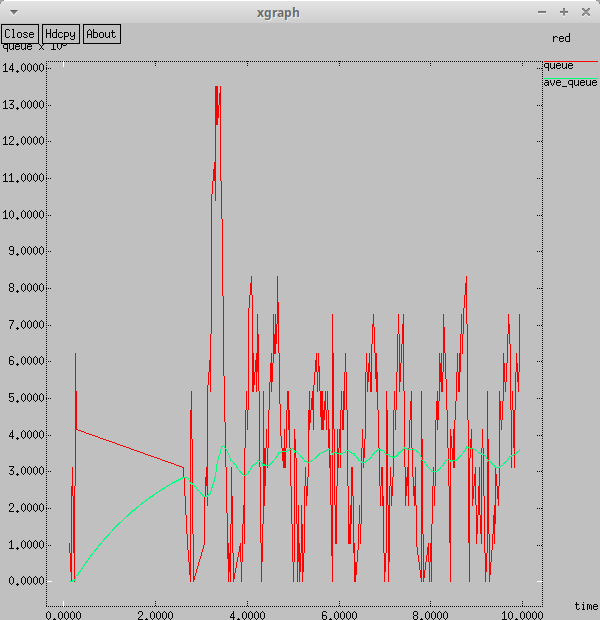


Figure 9: График динамики длины очереди и средней длины очереди

По графику видно, что средняя длина очереди опять находится в диапазоне от 2 до 4 (но можно заметить, что значение длины чаще бывает меньшим, чем при типе Reno/NeReno). Максимальная длина достигает значения 14. Сильные отличия можно заметить по графикам динамики размера окна. При Vegas максимальный размер окна составляет 20, а не 34, как в NewReno. TCP Vegas обнаруживает перегрузку в сети до того, как случайно теряется пакет, и мгновенно уменьшается размер окна.Таким образом, TCP Vegas обрабатывает перегрузку без каких-либо потерь пакета.

## 4.3 Упражнение 2

Внесите изменения при отображении окон с графиками (измените цвет фона, цвет траекторий, подписи к осям, подпись траектории в легенде). Заменим это в коде (В процедуре finish изменим цвет траекторий, подписи легенд, а также добавив опции *-fg* и *-bg* изменим цвет текста и фона в *xgraph*) и посмотрим, какие результаты получились (рис. [[10](#fig:010)], [[11](#fig:011)]).

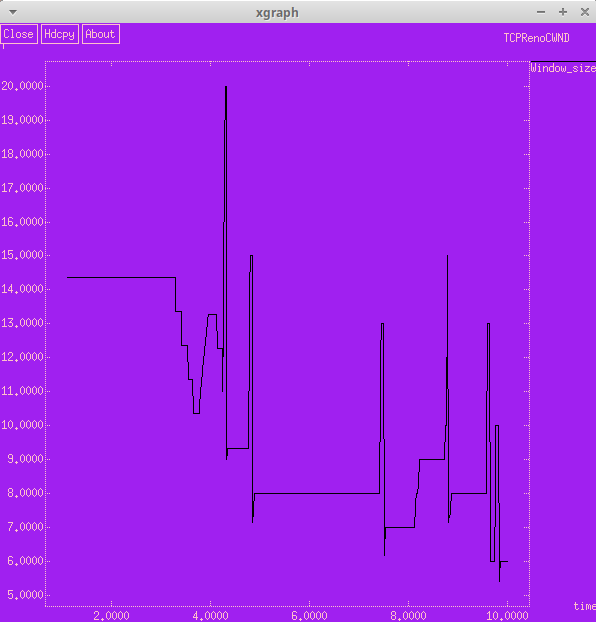


Figure 10: График динамики размера окна TCP

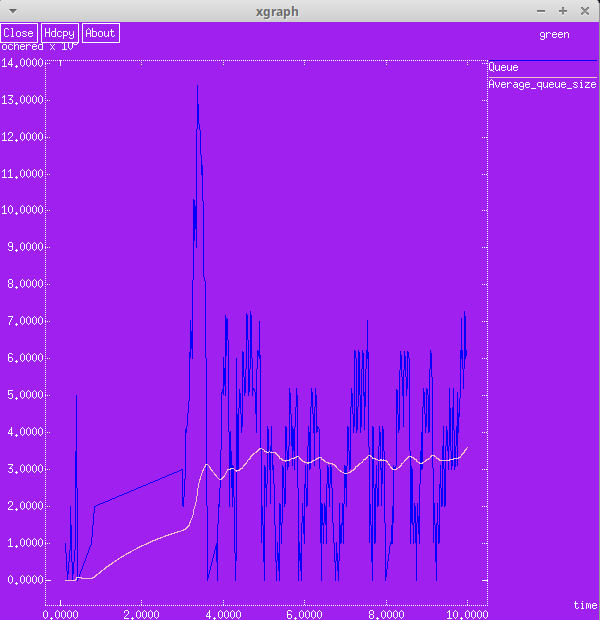


Figure 11: График динамики длины очереди и средней длины очереди

Так выглядит скрипт:

# создание объекта Simulator  
set ns [new Simulator]  
  
# открытие на запись файла out.nam для визуализатора nam  
set nf [open out.nam w]  
  
# все результаты моделирования будут записаны в переменную nf  
$ns namtrace-all $nf  
  
# открытие на запись файла трассировки out.tr  
# для регистрации всех событий  
set f [open out.tr w]  
# все регистрируемые события будут записаны в переменную f  
$ns trace-all $f  
  
# Процедура finish:  
proc finish {} {  
 global tchan\_  
 # подключение кода AWK:  
 set awkCode {  
 {  
 if ($1 == "Q" && NF>2) {  
 print $2, $3 >> "temp.q";  
 set end $2  
 }  
 else if ($1 == "a" && NF>2)  
 print $2, $3 >> "temp.a";  
 }  
 }  
 set f [open temp.queue w]  
 puts $f "TitleText: green"  
 puts $f "Device: Postscript"  
 puts $f "0.Color: Blue"  
 puts $f "1.Color: Pink"  
 if { [info exists tchan\_] } {  
 close $tchan\_  
 }  
 exec rm -f temp.q temp.a  
 exec touch temp.a temp.q  
  
 exec awk $awkCode all.q  
 puts $f \"Queue"  
 exec cat temp.q >@ $f  
 puts $f \n\"Average\_queue\_size"  
 exec cat temp.a >@ $f  
 close $f  
 # Запуск xgraph с графиками окна TCP и очереди:  
 exec xgraph -fg pink -bg purple -bb -tk -x time -t "TCPRenoCWND" WindowVsTimeReno &  
 exec xgraph -fg white -bg purple -bb -tk -x time -y ochered temp.queue &  
 exit 0  
}  
  
# Формирование файла с данными о размере окна TCP:  
proc plotWindow {tcpSource file} {  
 global ns  
 set time 0.01  
 set now [$ns now]  
 set cwnd [$tcpSource set cwnd\_]  
 puts $file "$now $cwnd"  
 $ns at [expr $now+$time] "plotWindow $tcpSource $file"  
}  
  
# Узлы сети:  
set N 5  
for {set i 1} {$i < $N} {incr i} {  
 set node\_(s$i) [$ns node]  
}  
set node\_(r1) [$ns node]  
set node\_(r2) [$ns node]  
# Соединения:  
$ns duplex-link $node\_(s1) $node\_(r1) 10Mb 2ms DropTail  
$ns duplex-link $node\_(s2) $node\_(r1) 10Mb 3ms DropTail  
$ns duplex-link $node\_(r1) $node\_(r2) 1.5Mb 20ms RED  
$ns queue-limit $node\_(r1) $node\_(r2) 25  
$ns queue-limit $node\_(r2) $node\_(r1) 25  
$ns duplex-link $node\_(s3) $node\_(r2) 10Mb 4ms DropTail  
$ns duplex-link $node\_(s4) $node\_(r2) 10Mb 5ms DropTail  
  
# Агенты и приложения:  
set tcp1 [$ns create-connection TCP/Vegas $node\_(s1) TCPSink $node\_(s3) 0]  
$tcp1 set window\_ 15  
set tcp2 [$ns create-connection TCP/Reno $node\_(s2) TCPSink $node\_(s3) 1]  
$tcp2 set window\_ 15  
set ftp1 [$tcp1 attach-source FTP]  
set ftp2 [$tcp2 attach-source FTP]  
  
# Мониторинг размера окна TCP:  
set windowVsTime [open WindowVsTimeReno w]  
puts $windowVsTime "0.Color: Black"  
puts $windowVsTime \"Window\_size"  
set qmon [$ns monitor-queue $node\_(r1) $node\_(r2) [open qm.out w] 0.1];  
[$ns link $node\_(r1) $node\_(r2)] queue-sample-timeout;  
  
# Мониторинг очереди:  
set redq [[$ns link $node\_(r1) $node\_(r2)] queue]  
set tchan\_ [open all.q w]  
$redq trace curq\_  
$redq trace ave\_  
$redq attach $tchan\_  
  
  
# Добавление at-событий:  
$ns at 0.0 "$ftp1 start"  
$ns at 1.1 "plotWindow $tcp1 $windowVsTime"  
$ns at 3.0 "$ftp2 start"  
$ns at 10 "finish"  
  
# запуск модели  
$ns run

# 5 Выводы

В результате выполнения лабораторной работы я исследовала протокол TCP и алгоритм управления очередью RED.

# Список литературы

1. Королькова А.В., Кулябов Д.С. Руководство к лабораторной работе №2. Моделирование информационных процессов. - 2025. — 10 с.