Лабораторная работа №2

Имитационное моделирование

Александрова Ульяна Вадимовна

Содержание

# 1 Цель работы

Целью данной работы является исследование протокола TCP и алгоритма управления очередью RED.

# 2 Задание

1. Прорешать пример с дисциплиной RED
2. Сделать упражнение на исследование других моделей протокола TCP

# 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 заполниться.
* 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.

Объект мониторинга очереди оповещает диспетчера очереди о поступлении пакета. Диспетчер очереди осуществляет мониторинг очереди.

* **qlim\_**: Максимально разрешённое число пакетов в очереди.
* **limit\_**: Размер очереди в пакетах.
* **blocked\_**: Принимает значение true, если очередь заблокирована.
* **unblock\_on\_resume\_**: Принимает значение true, указывая, что очередь должна быть разблокирована после отправки последнего пакета.
* **bytes\_**: Принимает значение true, если используется режим передачи в байтах, а не в пакетах.
* **queue-in-bytes\_**: Принимает значение true, если используется режим измерения среднего размера очереди в байтах, а не пакетах.
* **thresh\_**: Минимальный порог среднего размера очереди (в пакетах).
* **maxthresh\_**: Максимальный порог среднего размера очереди (в пакетах).
* **mean\_pktsize\_**: Грубая оценка среднего размера пакета (в байтах).
* **q\_weight\_**: Вес очереди (используется при расчёте экспоненциально взвешенного скользящего среднего размера очереди).
* **wait\_**: Интервал времени между сброшенными пакетами.
* **size\_**: Размер мгновенной длины очереди (в байтах).
* **pkts\_**: Размер мгновенной длины очереди (в пакетах).
* **parrivals\_**: Промежуточная сумма поступивших пакетов.
* **barrivals\_**: Промежуточная сумма байт в поступивших пакетах.
* **pdepartures\_**: Промежуточная сумма обслуженных пакетов (не отброшенных).
* **bdepartures\_**: Промежуточная сумма байт обслуженных пакетов (не отброшенных).
* **pdrops\_**: Общая сумма отброшенных пакетов.
* **bdrops\_**: Общая сумма байт отброшенных пакетов.
* **bytesInt\_**: Заполненность очереди в байтах.
* **pktsInt\_**: Заполненность очереди в пакетах.
* **epdrops\_**: Число сброшенных по алгоритму RED пакетов.
* **ebdrops\_**: Число байт в сброшенных по алгоритму RED пакетах.
* **enable\_in\_**: Устанавливается значение true, если требуется мониторинг потока на входе.
* **enable\_out\_**: Устанавливается значение true, если требуется мониторинг потока на выходе.
* **enable\_drop\_**: Устанавливается значение true, если требуется мониторинг сброшенных из потока пакетов.
* **enable\_edrop\_**: Устанавливается значение true, если требуется мониторинг сброшенных из потока пакетов по алгоритму RED.
* **src\_**: Адрес источника пакетов, принадлежащих потоку.
* **dst\_**: Адрес получателя пакетов, принадлежащих потоку.
* **flowid\_**: Идентификатор потока.

# 4 Выполнение лабораторной работы

## 4.1 Пример

Сначала выполняю пример из методического материала с данными условиями 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.

Реализую модель при помощи данного листинга:

# создание объекта Simulator  
set ns [new Simulator]  
  
set N 5  
for {set i 1} {$i < $N} {incr i} {  
set n(s$i) [$ns node]  
}  
  
set n(r1) [$ns node]  
set n(r2) [$ns node]  
  
$ns duplex-link $n(s1) $n(r1) 10Mb 2ms DropTail  
$ns duplex-link $n(s2) $n(r1) 10Mb 3ms DropTail  
$ns duplex-link $n(r1) $n(r2) 1.5Mb 20ms RED  
$ns queue-limit $n(r1) $n(r2) 25  
$ns queue-limit $n(r2) $n(r1) 25  
$ns duplex-link $n(s3) $n(r2) 10Mb 4ms DropTail  
$ns duplex-link $n(s4) $n(r2) 10Mb 5ms DropTail  
  
  
set tcp1 [$ns create-connection TCP/Reno $n(s1) TCPSink $n(s3) 0]  
$tcp1 set window\_ 15  
set tcp2 [$ns create-connection TCP/Reno $n(s2) TCPSink $n(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 $n(r1) $n(r2) [open qm.out w] 0.1];  
[$ns link $n(r1) $n(r2)] queue-sample-timeout;  
  
# Мониторинг очереди:  
set redq [[$ns link $n(r1) $n(r2)] queue]  
set tchan\_ [open all.q w]  
$redq trace curq\_  
$redq trace ave\_  
$redq attach $tchan\_  
  
$ns at 0.0 "$ftp1 start"  
$ns at 1.1 "plotWindow $tcp1 $windowVsTime"  
$ns at 3.0 "$ftp2 start"  
  
$ns at 10.0 "finish"  
  
# 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"  
}  
  
# 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 touch all.q  
  
 # выполнение кода AWK  
 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  
  
 set tempQueueContent [exec cat temp.q]  
 puts "Содержимое temp.q: $tempQueueContent"  
  
 # 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  
}  
  
 # запуск   
$ns run

Запускаю через симулятор и получаю два графика (рис. 1) (рис. 2).

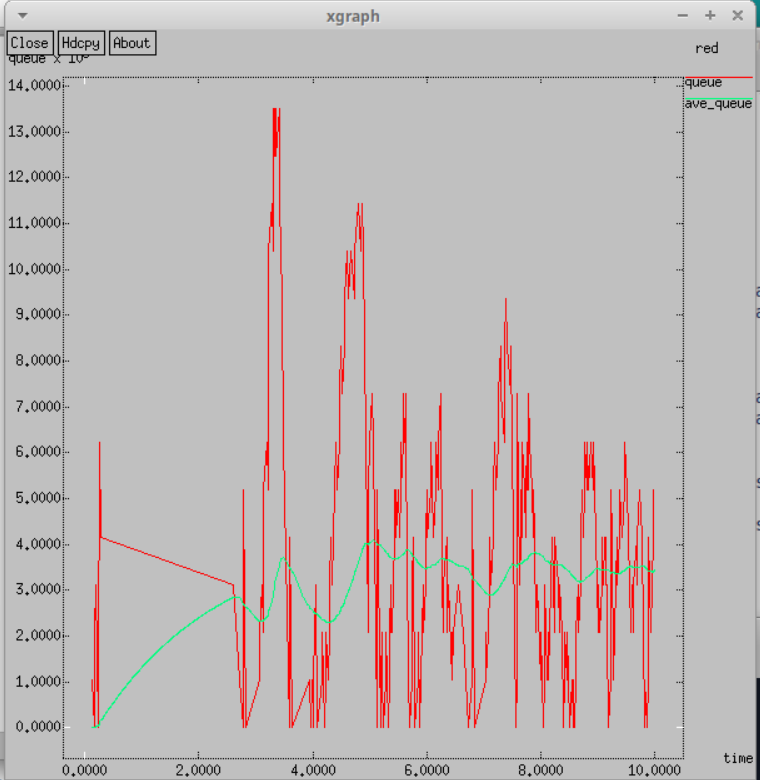


Рис. 1: График очереди

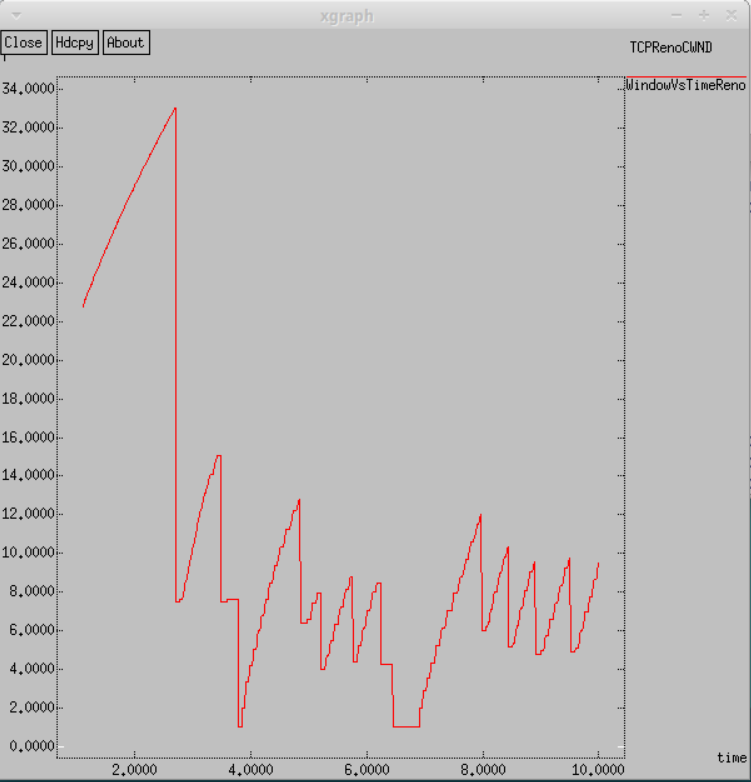


Рис. 2: График динамики размера окна TCP

Смотря на графики, мы можем заметить, что средняя длина очереди колеблется от 2 до 4, при этом максимума достигает в 14, а минимум по этому графику определить нельзя. Размер окна TCP достигает достаточно больших значений, из чего мы можем сделать вывод, что модель не очень хорошо справляется с удержанием пакетов.

## 4.2 Упражнение. NewReno

Чтобы выполнить это задание, редактирую уже имеющийся файл и меняю модель с Reno на NewReno, а также меняю цвет фона:

# создание объекта Simulator  
set ns [new Simulator]  
  
set N 5  
for {set i 1} {$i < $N} {incr i} {  
set n(s$i) [$ns node]  
}  
  
set n(r1) [$ns node]  
set n(r2) [$ns node]  
  
$ns duplex-link $n(s1) $n(r1) 10Mb 2ms DropTail  
$ns duplex-link $n(s2) $n(r1) 10Mb 3ms DropTail  
$ns duplex-link $n(r1) $n(r2) 1.5Mb 20ms RED  
$ns queue-limit $n(r1) $n(r2) 25  
$ns queue-limit $n(r2) $n(r1) 25  
$ns duplex-link $n(s3) $n(r2) 10Mb 4ms DropTail  
$ns duplex-link $n(s4) $n(r2) 10Mb 5ms DropTail  
  
  
set tcp1 [$ns create-connection TCP/Newreno $n(s1) TCPSink $n(s3) 0]  
$tcp1 set window\_ 15  
set tcp2 [$ns create-connection TCP/Reno $n(s2) TCPSink $n(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 WindowVsTimeNewReno w]  
set qmon [$ns monitor-queue $n(r1) $n(r2) [open qm.out w] 0.1];  
[$ns link $n(r1) $n(r2)] queue-sample-timeout;  
  
# Мониторинг очереди:  
set redq [[$ns link $n(r1) $n(r2)] queue]  
set tchan\_ [open all.q w]  
$redq trace curq\_  
$redq trace ave\_  
$redq attach $tchan\_  
  
$ns at 0.0 "$ftp1 start"  
$ns at 1.1 "plotWindow $tcp1 $windowVsTime"  
$ns at 3.0 "$ftp2 start"  
  
$ns at 10.0 "finish"  
  
# 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"  
}  
  
# 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 touch all.q  
  
 # выполнение кода AWK  
 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  
  
 set tempQueueContent [exec cat temp.q]  
 puts "Содержимое temp.q: $tempQueueContent"  
  
 set windowVsTimeContent [exec cat WindowVsTimeNewReno]  
 puts "Содержимое WindowVsTimeNewReno: $windowVsTimeContent"  
  
  
 # Запуск xgraph с графиками окна TCP и очереди:  
 exec xgraph -bb -tk -x time -t "TCPNewRenoCWND" -bg white -lw 1 -x 'Time' WindowVsTimeNewReno &  
 exec xgraph -bb -tk -x time -y queue -bg white 'Time' temp.queue &  
 exit 0  
}  
  
 # запуск   
$ns run

Просматриваю результаты и вижу, что они изменились, но не значительно (рис. 3) (рис. 4).

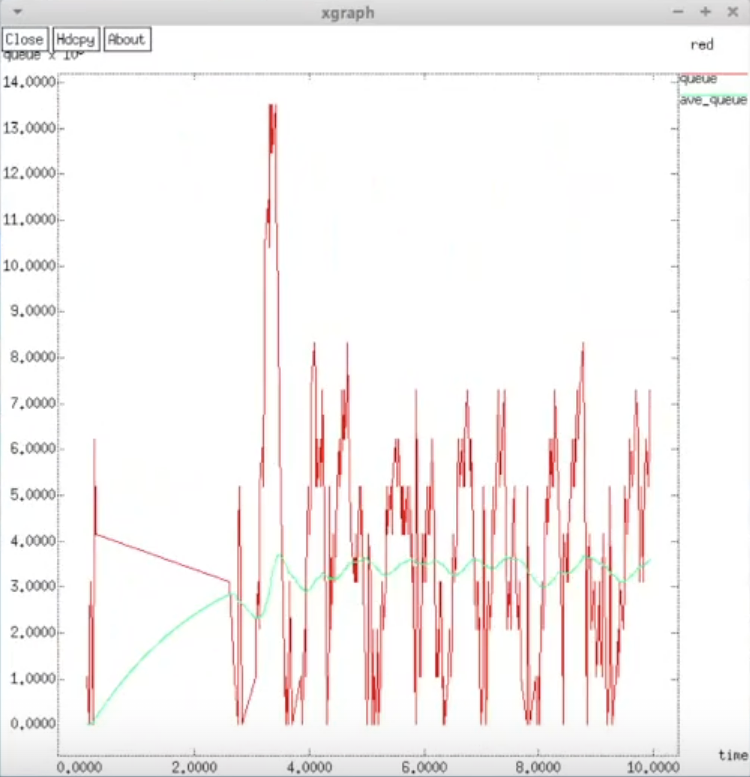


Рис. 3: График очереди для NewReno

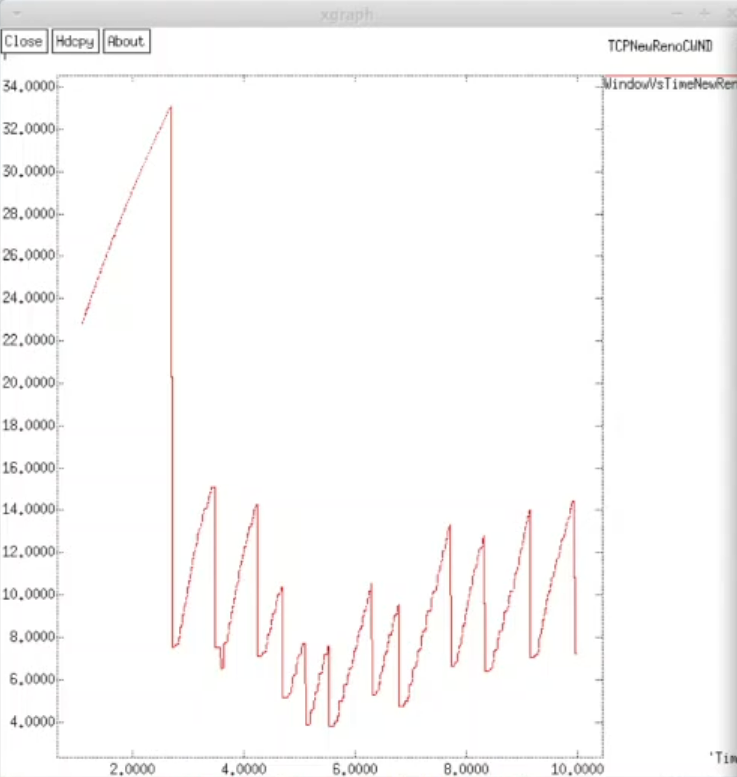


Рис. 4: График динамики размера окна TCP для NewReno

## 4.3 Упражнение. Vegas

Проделываю аналогичные действия для модели Vegas:

# создание объекта Simulator  
set ns [new Simulator]  
  
set N 5  
for {set i 1} {$i < $N} {incr i} {  
set n(s$i) [$ns node]  
}  
  
set n(r1) [$ns node]  
set n(r2) [$ns node]  
  
$ns duplex-link $n(s1) $n(r1) 10Mb 2ms DropTail  
$ns duplex-link $n(s2) $n(r1) 10Mb 3ms DropTail  
$ns duplex-link $n(r1) $n(r2) 1.5Mb 20ms RED  
$ns queue-limit $n(r1) $n(r2) 25  
$ns queue-limit $n(r2) $n(r1) 25  
$ns duplex-link $n(s3) $n(r2) 10Mb 4ms DropTail  
$ns duplex-link $n(s4) $n(r2) 10Mb 5ms DropTail  
  
  
set tcp1 [$ns create-connection TCP/Vegas $n(s1) TCPSink $n(s3) 0]  
$tcp1 set window\_ 15  
set tcp2 [$ns create-connection TCP/Reno $n(s2) TCPSink $n(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 WindowVsTimeVegas w]  
puts $windowVsTime "0.Color: Blue"  
puts $windowVsTime \"window  
set qmon [$ns monitor-queue $n(r1) $n(r2) [open qm.out w] 0.1];  
[$ns link $n(r1) $n(r2)] queue-sample-timeout;  
  
# Мониторинг очереди:  
set redq [[$ns link $n(r1) $n(r2)] queue]  
set tchan\_ [open all.q w]  
$redq trace curq\_  
$redq trace ave\_  
$redq attach $tchan\_  
  
$ns at 0.0 "$ftp1 start"  
$ns at 1.1 "plotWindow $tcp1 $windowVsTime"  
$ns at 3.0 "$ftp2 start"  
  
$ns at 10.0 "finish"  
  
# 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"  
}  
  
# 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 touch all.q  
  
 # выполнение кода AWK  
 exec awk $awkCode all.q   
 puts $f \"main  
 exec cat temp.q >@ $f  
 puts $f \n\"average  
 exec cat temp.a >@ $f  
 close $f  
  
 set tempQueueContent [exec cat temp.q]  
 puts "Содержимое temp.q: $tempQueueContent"  
  
 set windowVsTimeContent [exec cat WindowVsTimeVegas]  
 puts "Содержимое WindowVsTimeVegas: $windowVsTimeContent"  
  
  
 # Запуск xgraph с графиками окна TCP и очереди:  
 exec xgraph -bb -tk -x time -t "TCPVegasCWND" -bg pink -lw 1 -x 'Time' WindowVsTimeVegas &  
 exec xgraph -bb -tk -x time -y queue -bg pink -x 'Time' temp.queue &  
 exit 0  
}  
  
 # запуск   
$ns run

Графики имеют следующий вид (рис. 5) (рис. 6).

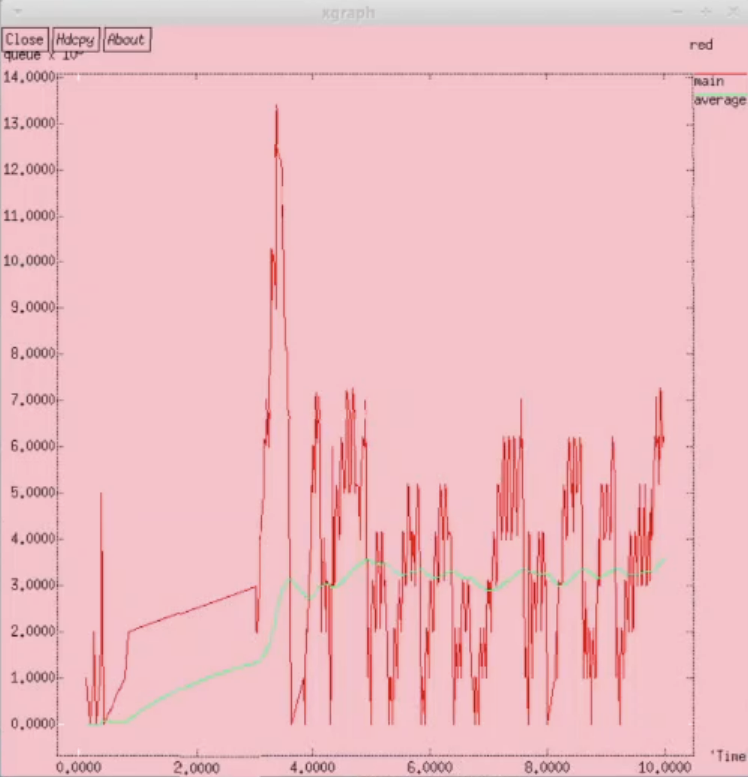


Рис. 5: График очереди для Vegas

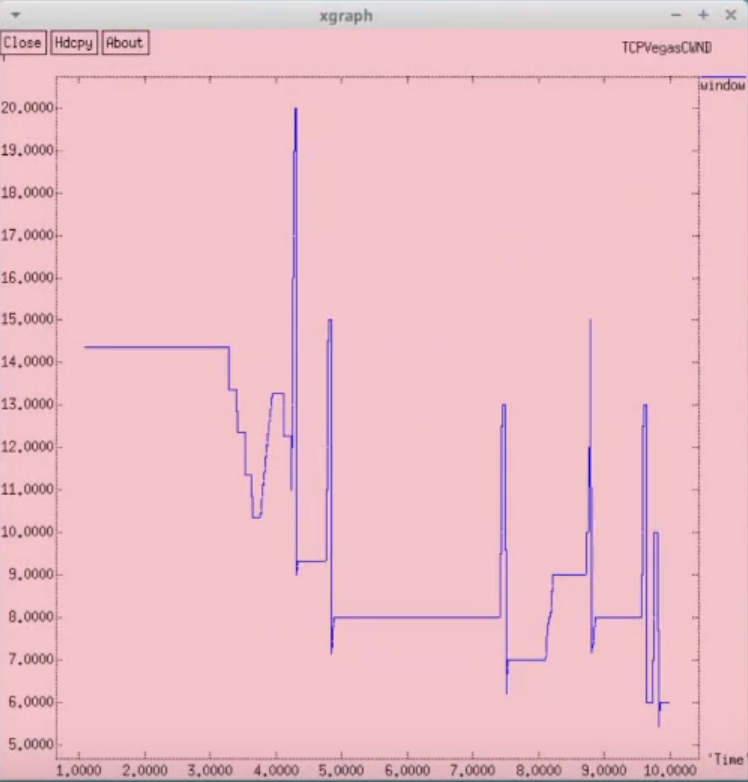


Рис. 6: График динамики размера окна TCP для Vegas

Теперь амплитуда колебания очереди уменьшиалсь с 2 до 4 до 3 до 4, а также максимальное значение теперь достигаеться в точке 13. График динамики размера окна также отличается. С этим методом, модель гораздо эффективнее работает с пакетами, засчет резкого изменения размера окна.

# 5 Выводы

Я исследовала алгоритмы управления очередью RED.