Лабораторная работа №2

Исследование протокола TCP и алгоритма управления очередью RED

Абу Сувейлим Мухаммед Мунифович

Содержание

# 1 Цель работы

* Приобретение навыков моделирования сетей передачи данных с помощью средства имитационного моделирования NS-2 на основе алгоритма RED, а также анализ полученных результатов моделирования.

# 2 Задание

* Измените в модели на узле s1 тип протокола TCP с Reno на NewReno, затем на Vegas. Сравните и поясните результаты.
* Внесите изменения при отображении окон с графиками (измените цвет фона, цвет траекторий, подписи к осям, подпись траектории в легенде). [1]

# 3 Теоретическое введение

“Алгоритм Random Early Detection (RED) лежит в основе ряда механизмов предотвращения и контроля перегрузок в очередях маршрутизаторов. Его основное предназначение заключается в сглаживании временных всплесков трафика и предупреждении длительной перегрузки сети посредством уведомления источников трафика о необходимости снижения интенсивности передачи информации.” [2] Алгоритм RED позволяет контролировать нагрузку с помощью выборочного случайного уничтожения некоторых пакетов, что заставляет протоколы, подобные TCP, снижать скорость передачи. При потере хотя бы одного пакета протокол TCP начинает процедуру Slow Start заново; это снижает объем трафика, поступающего в сеть. Наиболее разумно - не дожидаться полной перегрузки сети (тогда будет удален весь трафик), а уже на подступах к опасному порогу начать выборочное уничтожение отдельных пакетов, информируя тем самым источники нагрузки о текущей пропускной способности сети. [3]

# 4 Выполнение лабораторной работы

## 4.1 Шаблон сценария для NS-2

1. Во-первых, скопируем содержимое созданного шаблона в прошлой лабароторной работе в новый файл example.tcl:

* cp shablon.tcl example.tcl

1. и откроем example.tcl на редактирование. Добавим в него до строки $ns run описание топологии сети:

* # 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]

1. Соединим наши узлы и роутеры:

* # 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

1. Далее, создадим агенты ftp на улзах s1 и s2:

* # 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]

1. Подготовиш графический шаблон:

* # Здесь window\_ — верхняя граница окна приёмника (Advertisment Window) TCP соединения.  
   # 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;

1. Для мониторинга очереди. curq\_ — текущий размер очереди, ave\_ — средний размер очереди:

* # 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\_   
   # Здесь curq\_ — текущий размер очереди, ave\_ — средний размер очереди.

1. Добавление at-событий:

* # 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"

1. Формирование файла с данными о размере окна TCP:

* # 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"  
   }  
   # Здесь cwnd\_ — текущее значение окна перегрузки.

1. Процедура finish и запуск модели:

* # 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  
   }  
    
   # запуск модели  
   $ns run

1. График динамики длины очереди и средней длины очереди.

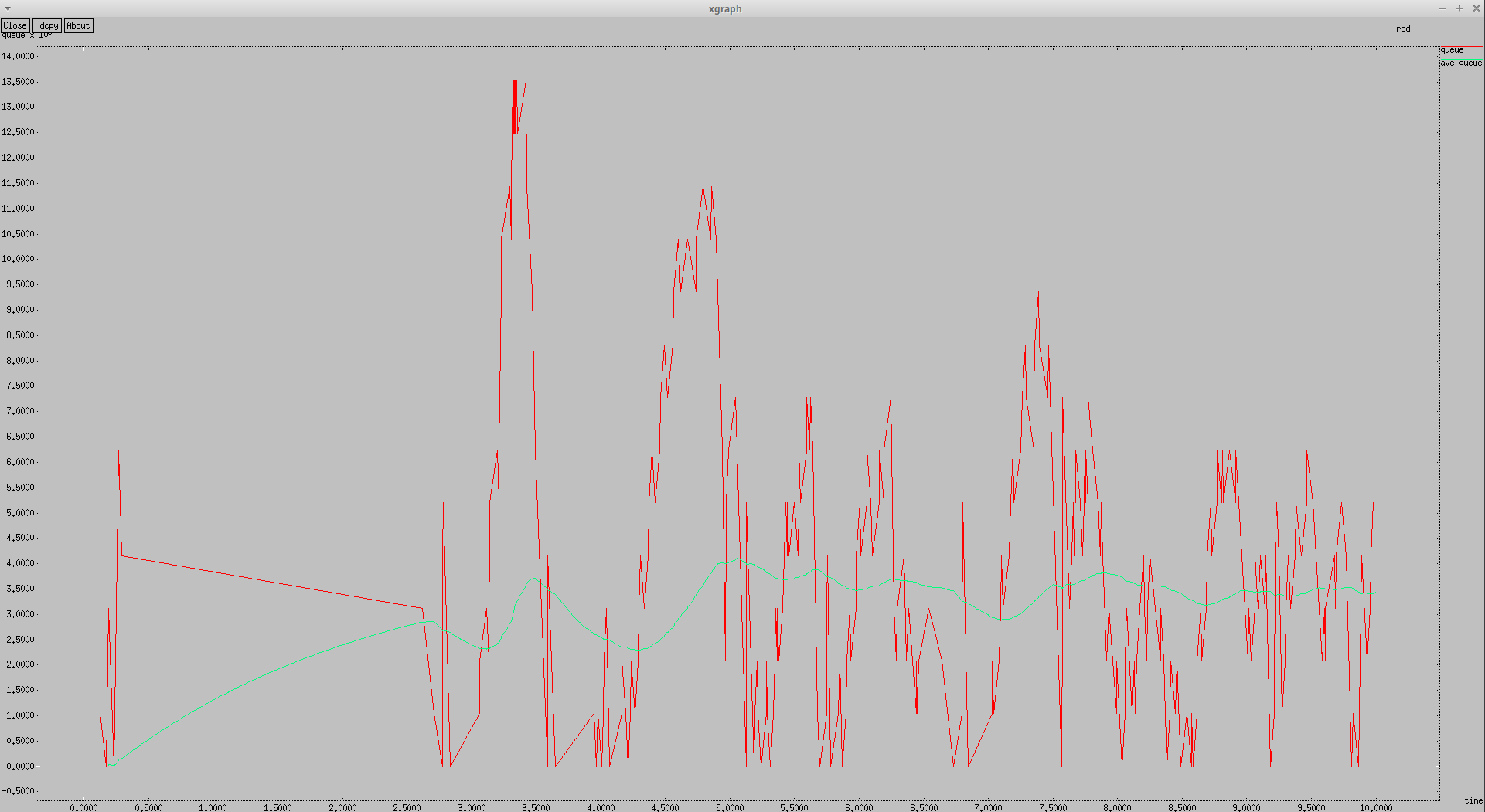


Рис. 1: График динамики длины очереди и средней длины очереди

## 4.2 Упражнение

1. Скопируем содержимое созданного задания 1 в новый файл:

* cp example.tcl exercise.tcl

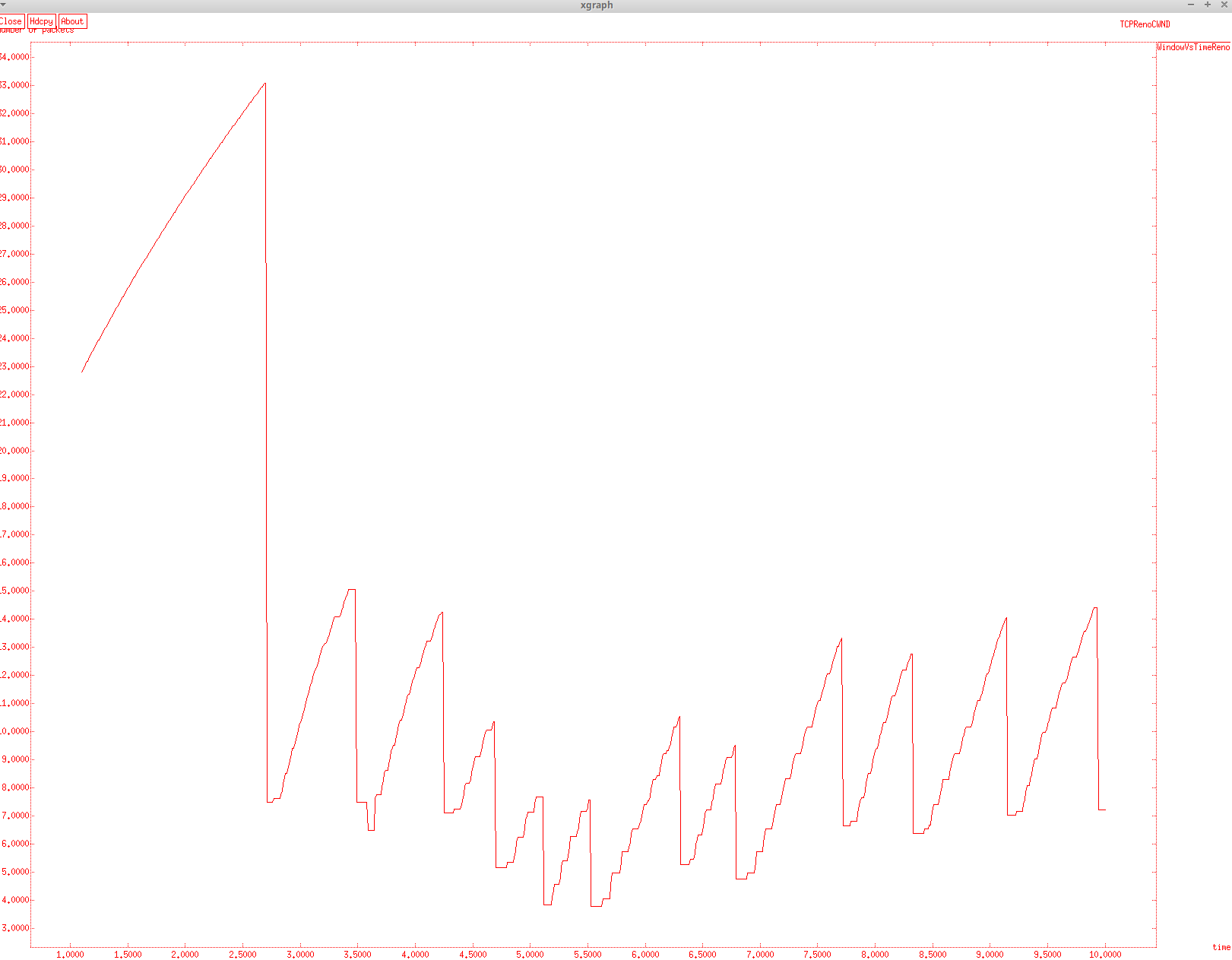
1. В процедуре внесем следующие изменения:

* # Запуск xgraph с графиками окна TCP и очереди:  
   exec xgraph -bb -bg White -fg Red -x "time" -y "number of packets" -tk -x time -t "TCPRenoCWND" WindowVsTimeReno &  
   exec xgraph -bb -bg White -fg Red -x "time" -y "number of packets"-tk -x time -y queue temp.queue &   
   exit 0
* Цвет фона - белый, цвет оси и линии - красный. Название оси х - время, а у - пакеты.

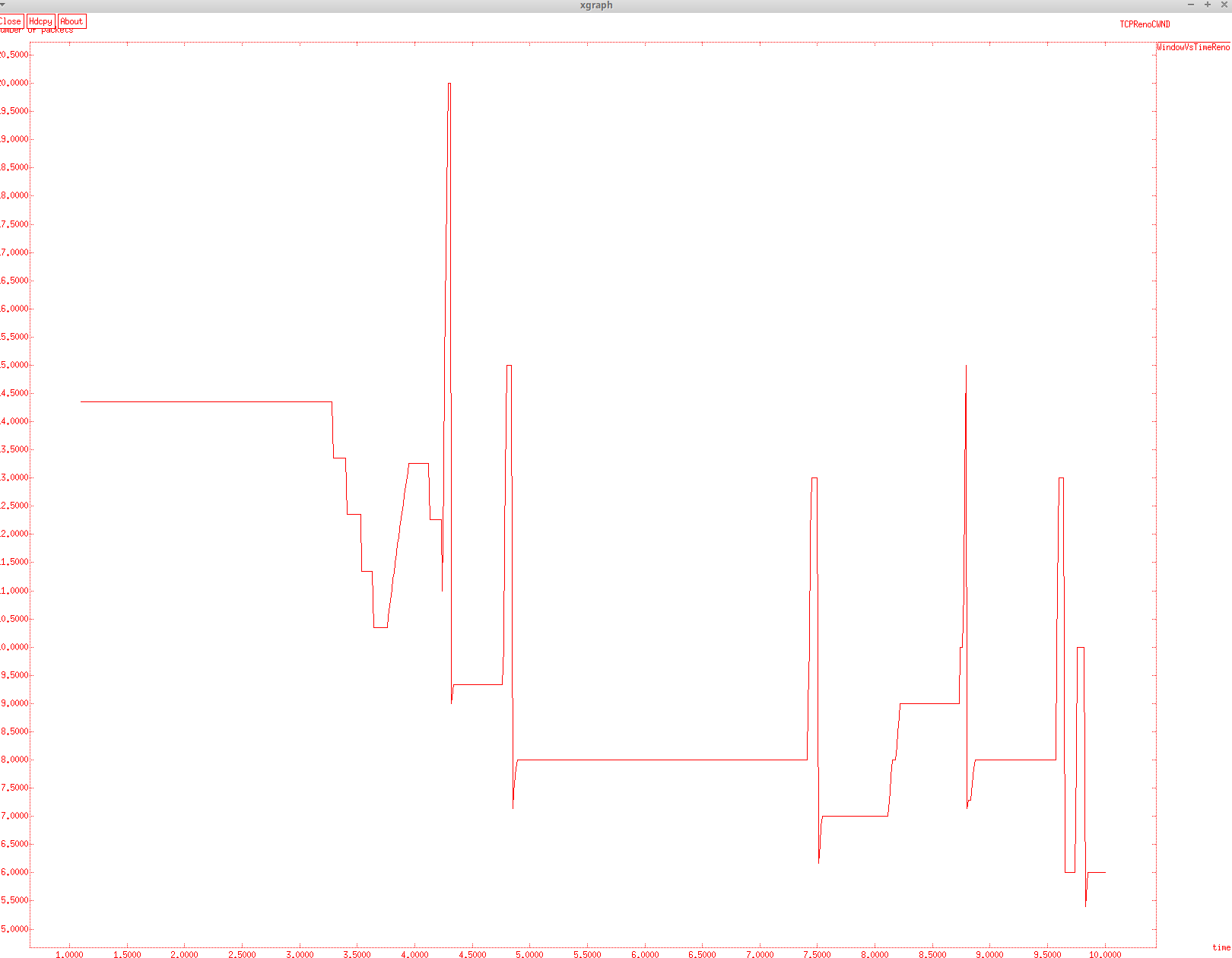
1. Где агенты и приложения вместо TCP/Reno напишим и TCP/Newreno и TCP/Vegas:

* # Агенты и приложения:  
   #set tcp1 [$ns create-connection TCP/Vegas $node\_(s1) TCPSink $node\_(s3) 0]  
   set tcp1 [$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]

1. Сохранив изменения в отредактированном файле и запустив симулятор:

* 
* Рис. 2: Изменённая сети из управжении на TCP/Newreno

1. Изменённая сети из управжении на TCP/Vegas:

* 
* Рис. 3: Изменённая сети из управжении на TCP/Vegas

## 4.3 Исходный код

### 4.3.1 Управжение

# создание объекта Simulator  
set ns [new Simulator]  
  
# 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]  
  
  
# Здесь window\_ — верхняя граница окна приёмника (Advertisment Window) TCP соединения.  
# 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\_  
  
  
# Здесь curq\_ — текущий размер очереди, ave\_ — средний размер очереди.  
# 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"  
  
# 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"  
}  
  
# Здесь cwnd\_ — текущее значение окна перегрузки.  
# 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 -bg White -fg Red -x "time" -y "number of packets" -tk -x time -t "TCPRenoCWND" WindowVsTimeReno &  
 exec xgraph -bb -bg White -fg Red -x "time" -y "number of packets"-tk -x time -y queue temp.queue &   
 exit 0  
}  
  
  
  
# запуск модели  
$ns run

# 5 Вывод

* Изучали как работает алгоритм RED. [1]

# 6 Библиография

1. Korolkova A., Kulyabov D. Моделирование информационных процессов. 2014.

2. Korolkova A., Kulyabov D., Черноиванов А. К вопросу о классификации алгоритмов RED // Вестник РУДН. Серия «Математика. Информатика. Физика». 2009. С. 34–46.

3. Алленов О. Алгоритм RED: красный свет для лишних пакетов [Электронный ресурс]. 1998. URL: <https://www.osp.ru/nets/1998/09/143680>.