Лабораторная работа № 2

Исследование протокола TCP и алгоритма управления очередью RED

Демидова Екатерина Алексеевна

Содержание

Список иллюстраций

1 Цель работы

Исследование протокола TCP и алгоритма управления очередью RED.

2 Задание

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 описанную модель, построить в Xgraph график изменения TCP-окна, график изменения длины очереди и средней длины очереди.

3 Выполнение лабораторной работы

3.1 Пример с дисциплиной RED

Ниже приведен листинг, который задаёт сеть 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. Также строится в Xgraph график изменения TCP-окна, график изменения длины очереди и средней длины очереди.

```
#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
     # 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
    # 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_

#at-событие для планировщика событий, которое запускает
#процедуру finish через 5 с после начала моделирования
# 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
```

3.2 Результаты моделирования

В результате получим графики для типа Reno протокола TCP(рис. [3.1], [3.2]).

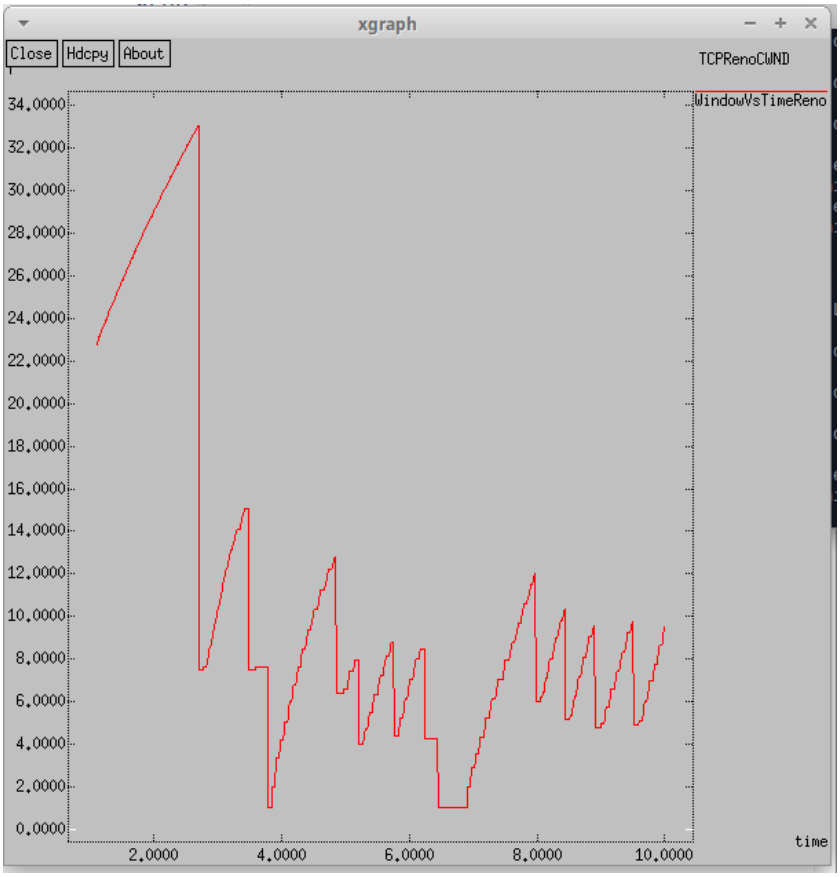

Рис. 3.1: График динамики размера окна TCP. Reno

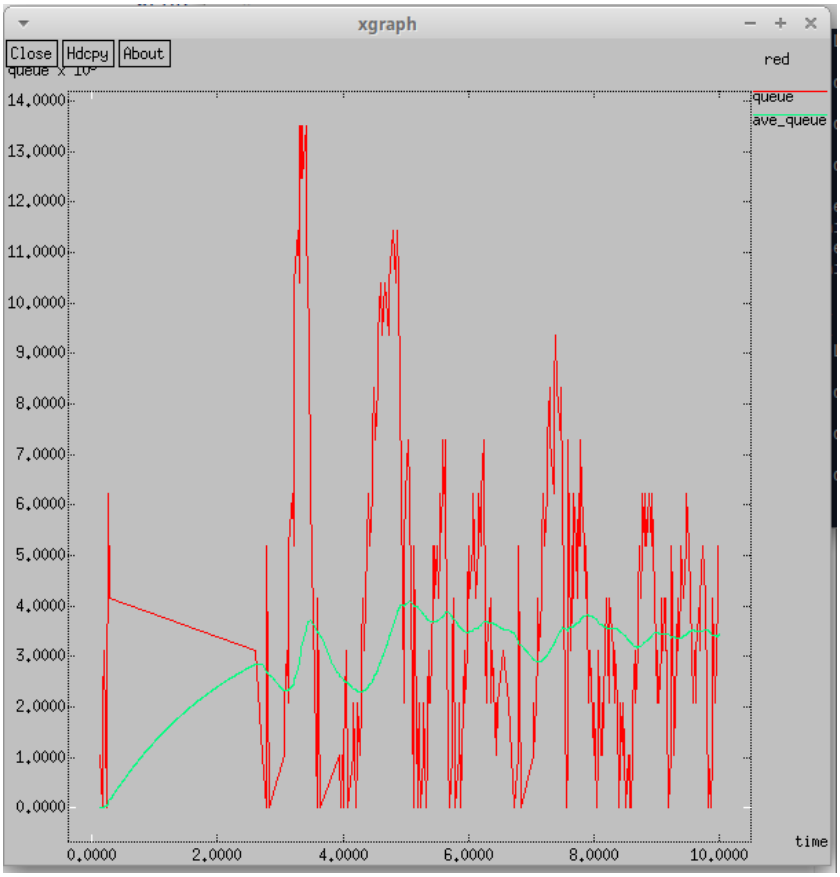

Рис. 3.2: График динамики длины очереди и средней длины очереди. Reno

Можно увидеть, что средняя длина очереди колеблется от 2 до 4, а максимальная длина достигает 14. Динамика размера окна циклична.

3.3 Изменение типа TCP

Теперь изменим тип протокола TCP с Reno на NewReno:

```
 Агенты и приложения:
set tcp1 [$ns create-connection TCP/Newreno $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Newreno $node_(s2) TCPSink $node_(s3) 1]
$tcp2 set window_ 15
```

```
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]
```

Можно увидеть, что средняя длина очереди колеблется от 2 до 4, а максимальная длина достигает 14. Графики Newreno и Reno совпадают. Оба эти алгоритма увеличивают размер окна, пока не произойдет потеря сегмента, а затем уменьшают его, пока не разгрузит очередь. Поэтому динамика размера окна циклична.

(рис. [3.3], [3.4]).

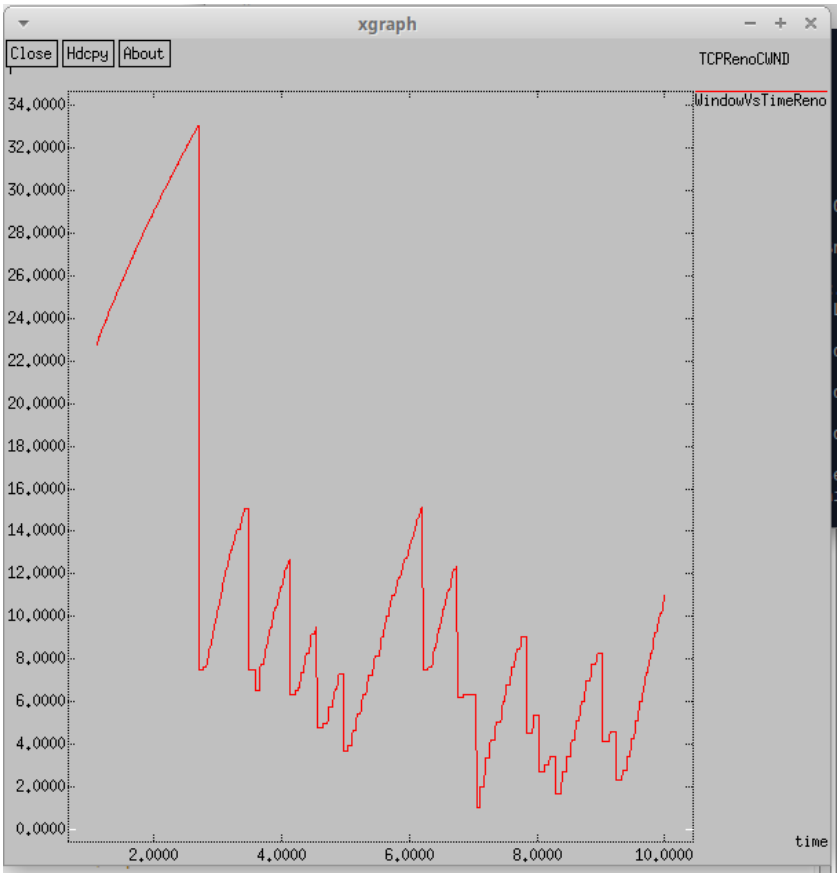

Рис. 3.3: График динамики размера окна TCP. NewReno

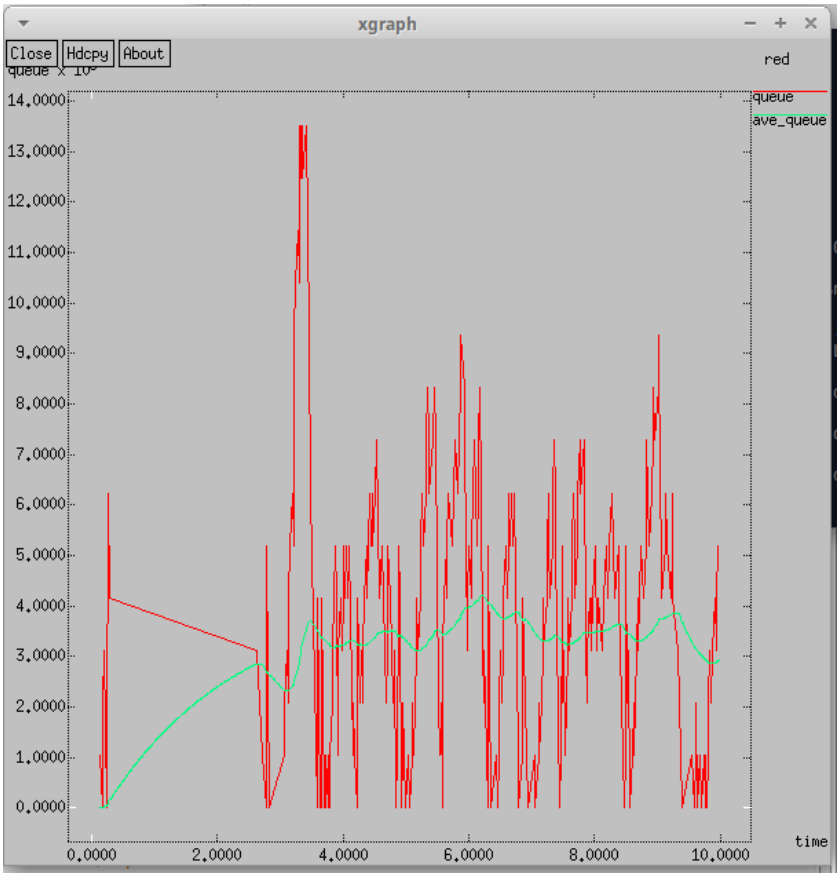

Рис. 3.4: График динамики длины очереди и средней длины очереди. NewReno

Теперь изменим тип протокола TCP на Vegas

```
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/Vegas $node_(s2) TCPSink $node_(s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]
```

Можно увидеть, что средняя длина очереди колеблется от 2 до 3, а максимальная длина достигает 11. По сравнению с двумя предыдущими типами колебания имеют меньшую амплитуду. Это объясняется тем, что этот тип контролирует

размер окна путем мониторивания отправителем RTT(время приема-передачи). Отправитель определяет с помощью этого сеть стремится к перегрузке или нет(за счет увеличения и уменьшения RTT) и соответственно подстраивает размер окна. Это способствует стремлению размера окна к требуемому значению(рис. [3.5], [3.6]).

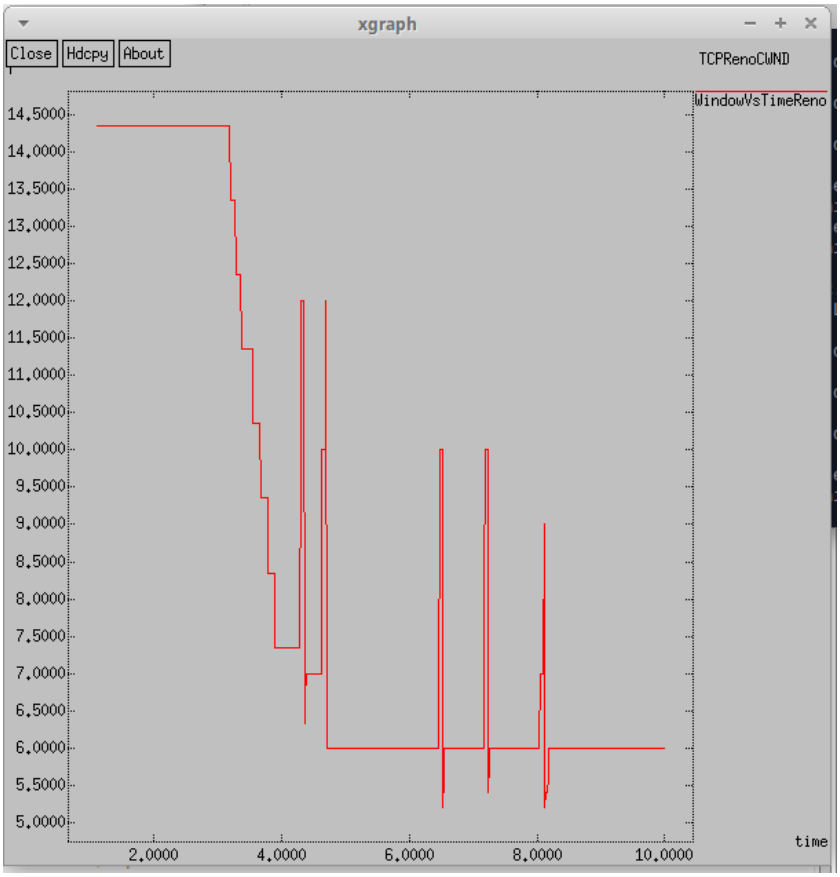

Рис. 3.5: График динамики размера окна TCP. Vegas

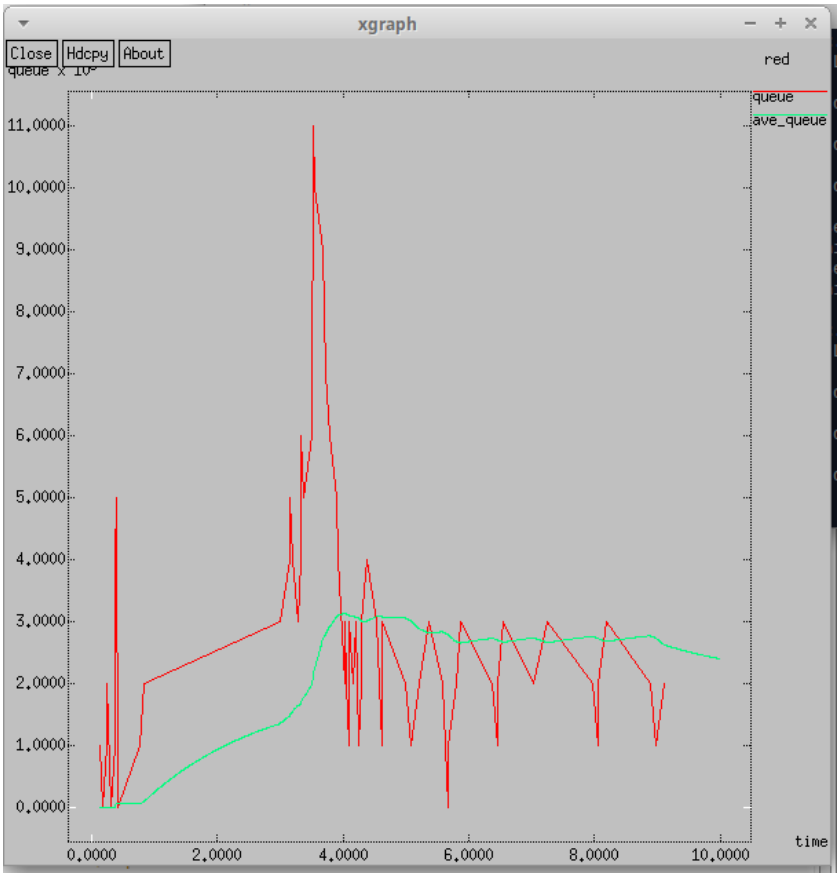

Рис. 3.6: График динамики длины очереди и средней длины очереди. Vegas

3.4 Изменение типа TCP отображения графиков

Внесем изменения при отображении окон с графиками (изменим цвет фона, цвет траекторий, подписи к осям, подпись траектории в легенде). Для этого изменим код в процедуре finish:

```
set f [open temp.queue w]
puts $f "TitleText: red"
puts $f "Device: Postscript"
puts $f "0.Color: Blue"
puts $f "1.Color: Yellow"
if { [info exists tchan_] } {
```

```
    close $tchan_
}
exec rm -f temp.q temp.a
exec touch temp.a temp.q
 # выполнение кода AWK
exec awk $awkCode all.q
puts $f \"Ochered"
exec cat temp.q >@ $f
puts $f \n\"Srednee_ocheredi"
exec cat temp.a >@ $f
close $f
# Запуск xgraph с графиками окна TCP и очереди:
exec xgraph -fg white -bg black -bb -tk -x vremya -t "TCPRenoCWND" WindowVsTimeRe
exec xgraph -fg white -bg black -bb -tk -x vremya -y ochered temp.queue &
```

А название траектории динамики размера окна и её цвет задается вне этой процедуры:

```
# Мониторинг размера окна TCP:
set windowVsTime [open WindowVsTimeReno w]
puts $windowVsTime "0.Color: Blue"
puts $windowVsTime \"DinamikaRazmeraOkna
```

В результате получим измененный график(рис. [3.7], [~ 3.8]).

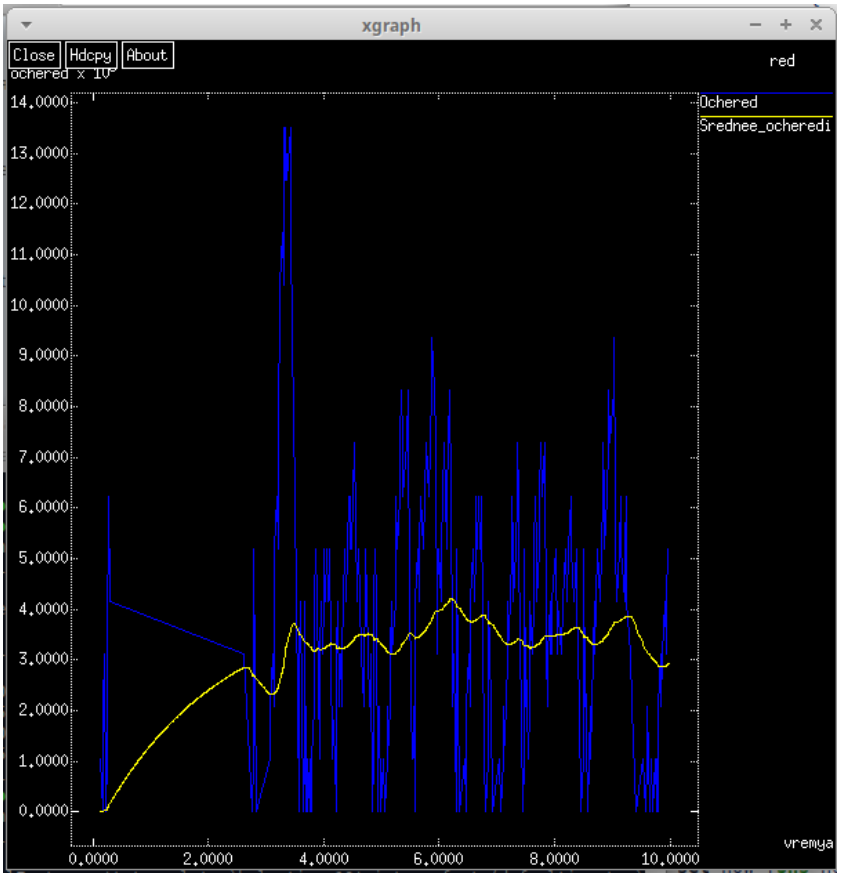

Рис. 3.7: Изменение отображения графика. Длина очереди

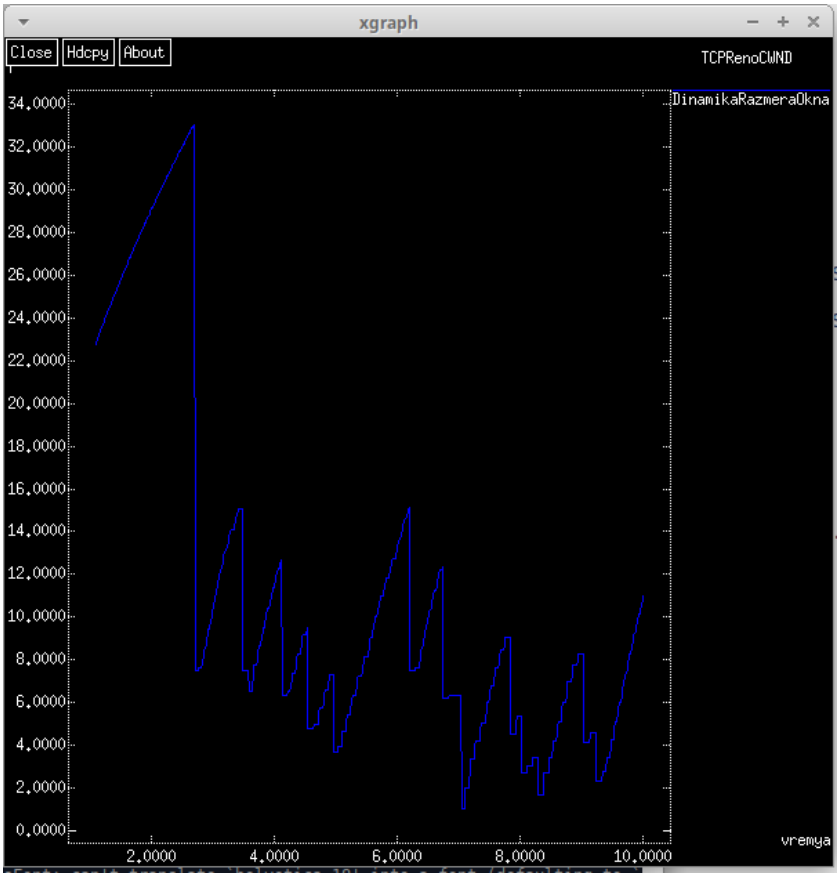

Рис. 3.8: Изменение отображения графика. Динамика размера окна

4 Выводы

В результате выполнения работы был исследован протокола TCP и алгоритм управления очередью RED, нарисованы и проанализированы графики динамики размера окна и длины очереди для разных типов TCP.