Лабораторная работа 2. Исследование протокола TCP и алгоритма управления очередью RED

2.1. Предварительные сведения.

2.1.1. Протокол TCP

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

Рис. 2.1. Формат заголовка пакета TCP

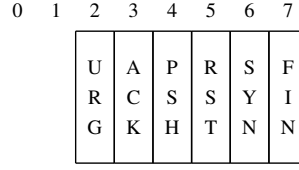

Рис. 2.2. Поле *Флаги* заголовка пакета TCP

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.

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;
- если значение этого поля равно нулю, это означает, что все байты, 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 с перезагрузкой 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

TCP Tahoe:
- медленный старт (Slow-Start);
- контроль перегрузки (Congestion Avoidance);
- быстрый повтор передачи (Fast Retransmit);
- метод оценки длительности цикла передачи (Round Trip Time, RTT), используемой для установки таймера повторной передачи (Retransmission TimeOut, RTO).

Схема работы TCP Tahoe:
- при переполнении буфера все сегменты теряются;
- при потере сегмента или с наступлением таймаута запускается *процедура медленного старта* — потерянный пакет и все, посланные после него пакеты (вне зависимости от того, подтверждено их получение или нет) пересылаются повторно;
- контроль перегрузки и оценка RTT: окно перегрузки увеличивается на 1 пакет с каждым ACK, полученным в течение медленного старта (`cwnd_ < ssthresh_`), и увеличивается на `1/cwnd_` для каждого нового ACK, полученного при избежании перегрузки (когда `cwnd_ >= ssthresh_`);

– реакция на перегрузку: при получении трёх дублированных ACK устанавливается `ssthresh_=min(cwnd_,window_)/2`

TCP Reno:
– 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;
  – 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.

2.1.2. Мониторинг очередей

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

Рис. 2.3. Звено с объектами мониторинга очереди

SnoopQ/In — объект мониторинга очереди на входе.
SnoopQ/Out — объект мониторинга очереди на выходе.
SnoopQ/Drop — объект мониторинга отбрасываемых из очереди пакетов.
RecvT (receive tracing) — объект мониторинга принятых узлом пакетов.
Объекты 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.

Объекты мониторинга очереди:

– 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.

```
# Пример задания множества объектов мониторинга:
SimpleLink instproc \
  attach-monitors { insnoop outsnoop dropsnoop qmon } {
   $self instvar queue_ head_ snoopIn_ snoopOut_ snoopDrop_
   $self instvar drophead_ qMonitor_
   set snoopIn_ $insnoop
   set snoopOut_ $outsnoop
   set snoopDrop_ $dropsnoop
   $snoopIn_ target $head_
   set head_ $snoopIn_
   $snoopOut_ target [$queue_ target]
   $queue_ target $snoopOut_
   $snoopDrop_ target [$drophead_ target]
   $drophead_ target $snoopDrop_
   $snoopIn_ set-monitor $qmon
   $snoopOut_ set-monitor $qmon
   $snoopDrop_ set-monitor $qmon
   set qMonitor_ $qmon
}
```

```
# Пример использования объектов мониторинга очереди соединения.
# Возвращает имя объекта, требуемого для определения
# среднего размера очереди
SimpleLink instproc init-monitor { ns qtrace sampleInterval} {
    $self instvar qMonitor_ ns_ qtrace_ sampleInterval_

    set ns_ $ns
    set qtrace_ $qtrace
    set sampleInterval_ $sampleInterval
    set qMonitor_ [new QueueMonitor]

    $self attach-monitors [new SnoopQueue/In]
        [new SnoopQueue/Out] [new SnoopQueue/Drop] $qMonitor_

    set bytesInt_ [new Integrator]
    $qMonitor_ set-bytes-integrator $bytesInt_
    set pktsInt_ [new Integrator]
    $qMonitor_ set-pkts-integrator $pktsInt_
    return $qMonitor_ }
```

2.2. 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 (см. рис. 2.4);
- 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.

На рис. 2.4 приведена схема моделируемой сети.

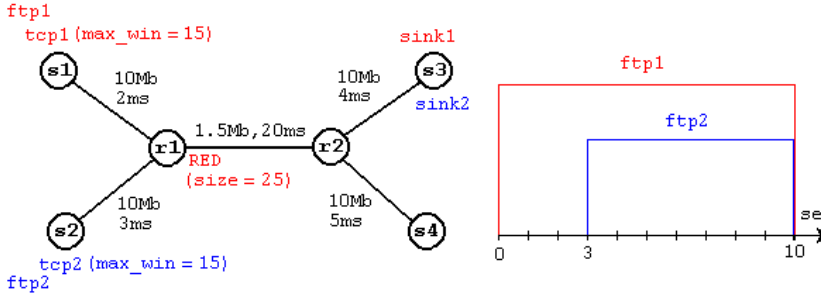

Рис. 2.4. Схема сети

На рис. 2.5 приведена схема работы модуля RED.

На рис. 2.5 q — число пакетов в очереди, $\hat{q}$ — экспоненциально взвешенное скользящее среднее значение длины очереди, $p(\hat{q})$ — функция сброса пакетов.

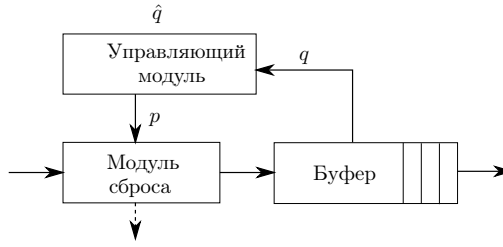

Рис. 2.5. Схема работы модуля RED

Функция сброса алгоритма RED имеет вид (рис. 2.6):

$$p^{\text{RED}}(\hat{q}) = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ \dfrac{\hat{q} - q_{\min}}{q_{\max} - q_{\min}} p_{\max}, & q_{\min} < \hat{q} \leqslant q_{\max}, \\ 1, & \hat{q} > q_{\max}, \end{cases}$$

где $q_{\min}$, $q_{\max}$ — пороговые значения очереди; $p_{\max}$ — параметр максимального сброса.

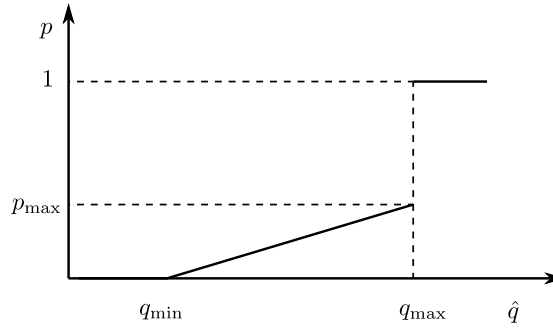

Рис. 2.6. Функция сброса алгоритма RED

Требуется разработать сценарий, реализующий модель согласно рис. 2.4, построить в Xgraph график изменения TCP-окна, график изменения длины очереди и средней длины очереди.

Реализация модели

```
# 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]
```

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 # выполнение кода AWK
 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
}
```

Упражнение

- 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).

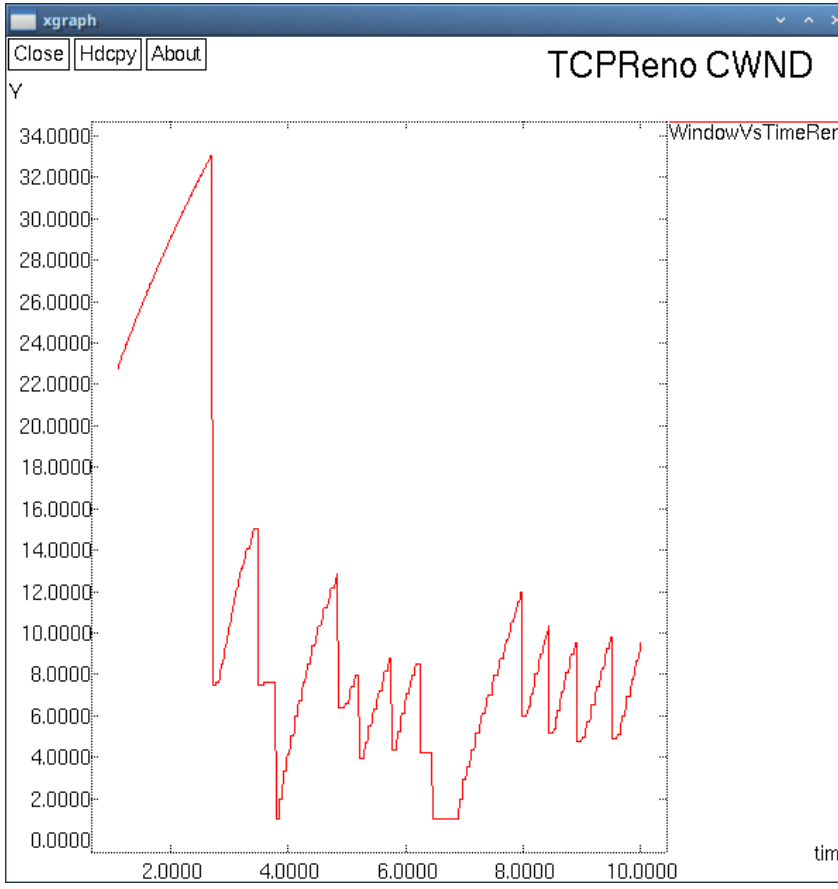

Рис. 2.7. График динамики размера окна TCP

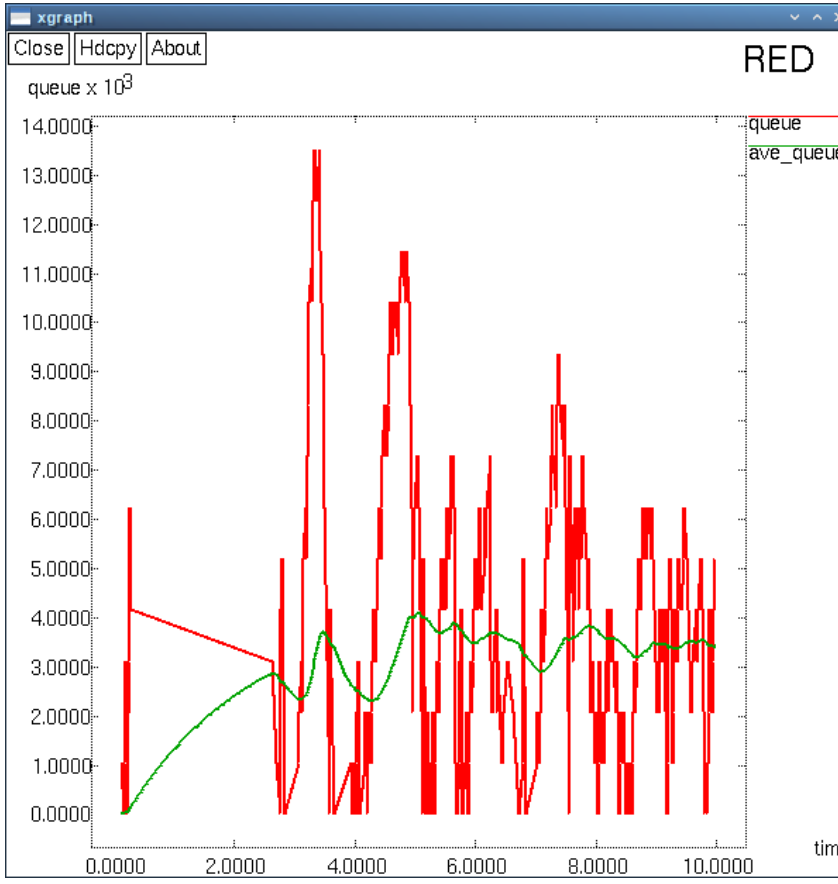

Рис. 2.8. График динамики длины очереди и средней длины очереди