Лабораторная работа № 2

Исследование протокола TCP и алгоритма управления очередью RED

Мугари Абдеррахим

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

- Цель лабораторной работы — исследование поведения различных версий протокола TCP (NewReno, Reno) в сети с управлением перегрузками с использованием алгоритма RED. Анализируется изменение размера окна перегрузки и влияние параметров очереди на эффективность передачи данных. Результаты визуализируются с помощью xgraph, с настройкой цвета и подписей графиков.

2 Выполнение лабораторной работы

2.1 Пример с дисциплиной RED

2.1.1 Алгоритм управления очередью RED:

- здесь мы познакомились с алгоритмом RED (случайного раннего обнаружения) и его функцией сброса (Схема и распределение) (рис. 2.1).

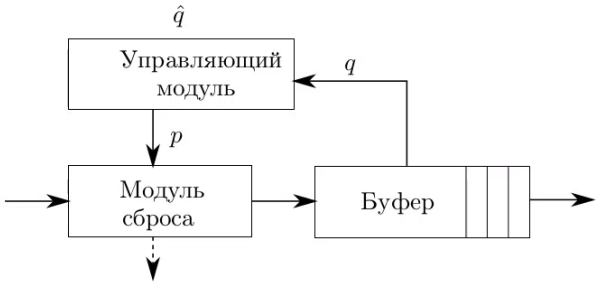

Рис. 2.5. Схема работы модуля RED

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

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

Рис. 2.1: Алгоритм управления очередью RED

2.1.2 Реализация модели задачи

2.1.3 Создание симудятора

- Создается объект симулятора (ns).
- Определяются узлы сети: s1, s2, s3, s4 (отправители) и r1, r2 (маршрутизаторы) (рис. 2.2).

```
set ns [new Simulator]

set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]

$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
```

Рис. 2.2: Алгоритм управления очередью RED

2.1.4 Создание соединений между узлами

- Создаются дуплексные (двусторонние) соединения между узлами с заданной пропускной способностью, задержкой и алгоритмом управления очередью (DropTail или RED).
- Задается лимит очереди на связи r1-r2 и r2-r1 (по 25 пакетов). (рис. 2.3).

```
$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
```

Рис. 2.3: Создание соединений между узлами

2.1.5 Настройка TCP-соединений

- Создаются два TCP-соединения: одно использует TCP/NewReno, другое — TCP/Reno.
- Каждое соединение имеет размер окна 15.
- На TCP-соединения привязываются источники трафика (FTP) (рис. 2.4).

```
set tcp1 [$ns create-connection TCP/Newreno $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]
```

Рис. 2.4: Настройка TCP-соединений

2.1.6 Настройка мониторинга очередей и логирования

- Открывается файл WindowVsTimeReno для записи данных о размере окна перегрузки.
- Включается мониторинг очереди между r1 и r2 (файл qm.out).
- Настраивается трассировка состояния очереди RED (рис. 2.5).

```
set windowVsTime [open WindowVsTimeNewreno 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;

set redq [[$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_
```

Рис. 2.5: Настройка мониторинга очередей и логирования

2.1.7 Запуск передачи данных

FTP1 (TCP NewReno) начинает передачу в **0.0 сек**. Через **1.1 сек**. начинается запись данных о размере окна перегрузки. **FTP2 (TCP Reno)** стартует через **3.0 сек**. Симуляция заканчивается в **10 сек**.(рис. 2.6).

```
$ns at 0.0 "$ftp1 start"
$ns at 1.1 "plotWindow $tcp1 $windowVsTime"
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"
```

Рис. 2.6: Запуск передачи данных

2.1.8 Функция для записи изменения размера окна TCP

Функция записывает размер окна TCP в файл каждые 0.01 сек (рис. 2.7).

```
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"
}
```

Рис. 2.7: Функция для записи изменения размера окна TCP

2.1.9 Функция завершения симуляции и построения графиков

- Код обрабатывает данные о длине очереди и создает временные файлы.
- Генерирует два графика с помощью xgraph TCPNewrenoCWND и queue (рис. 2.8).

```
proc finish {} {
        global tchan_

        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

        exec xgraph -bb -tk -x time -t "TCPNewrenoCWND" WindowVsTimeNewreno &
        exec xgraph -bb -tk -x time -y queue temp.queue &
        exit 0
}

$ns run
```

Рис. 2.8: Функция завершения симуляции и построения графиков

2.1.10 Запуск симуляции

```
$ns run
```

2.1.11 Визуализация:

- Построение графиков с помощью xgraph после завершения симуляции. (рис. 2.9).

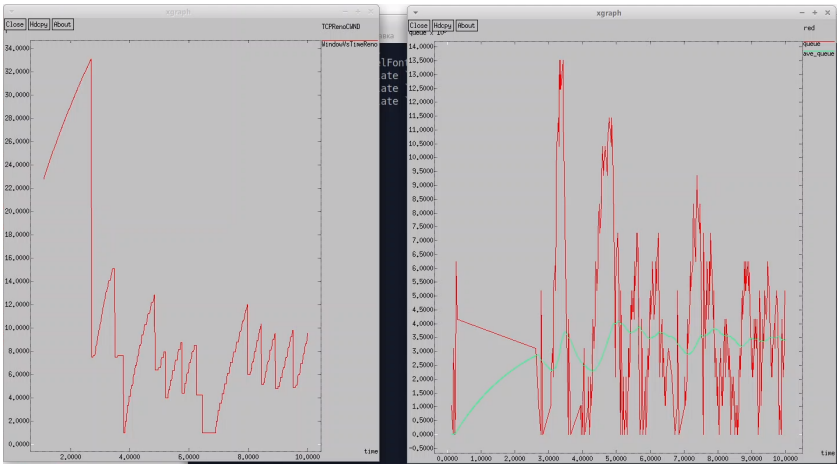

Рис. 2.9: построения графиков

2.1.12 Сравнение результатов и пояснения

2.1.12.1 TCP Reno

- **Механизм работы:** Основан на классическом алгоритме управления перегрузкой с фазами **slow start, congestion avoidance и механизмами fast retransmit/fast recovery**.
- **Характеристика поведения:** При потере пакетов окно резко уменьшается, что может приводить к значительным колебаниям в размере окна (**cwnd**)
- **Особенности:** Может испытывать проблемы при возникновении множественных потерь в одном цикле, что приводит к задержкам в восстановлении (рис. 2.10).

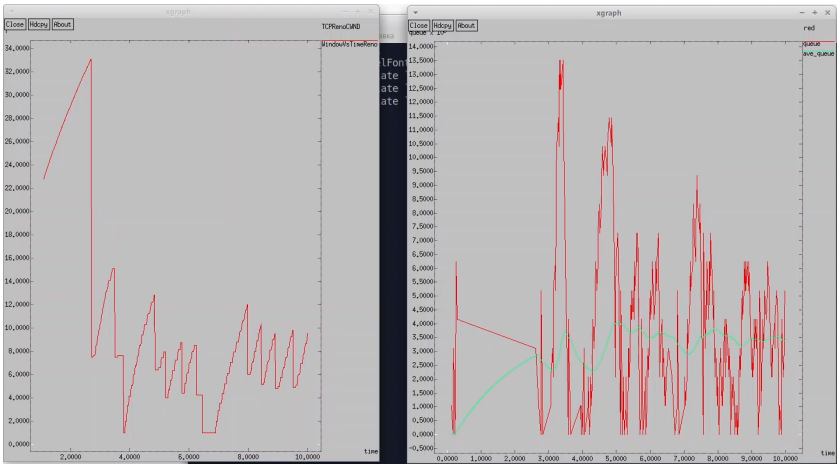

Рис. 2.10: Изменение окна перегрузки TCP Reno во времени и длины очереди во времени (RED-очередь на маршрутизаторе)

2.1.12.2 TCP NewReno

Здесь мы просто изменили тип TCP-агента на NewReno (рис. 2.11).

```
set tcp1 [$ns create-connection TCP/Neweno $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno $node_(s2) TCPSink $node_(s3) 1]
```

Рис. 2.11: TCP NewReno

- **Механизм работы:** Улучшенная версия Reno, где алгоритм **fast recovery** остаётся активным до восстановления всех потерянных пакетов в текущем цикле.
- **Характеристика поведения:** Более устойчив к множественным потерям в одном цикле. В графиках cwnd можно наблюдать менее резкие падения по сравнению с **Reno**.
- **Особенности:** В условиях высокой вероятности потерь (например, в сочетании с **RED-очередью**) может обеспечить более стабильный пропускной потенциал за счёт быстрейшего восстановления (рис. 2.12).

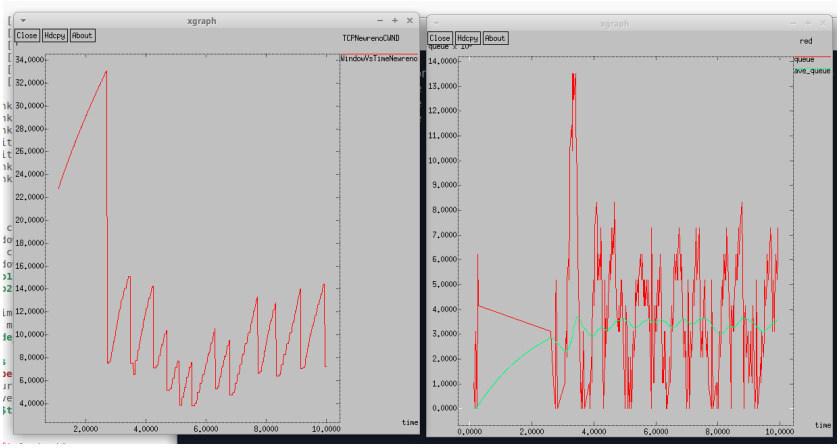

Рис. 2.12: Изменение окна перегрузки TCP NewReno во времени и длины очереди во времени (RED-очередь на маршрутизаторе)

2.1.12.3 TCP Vegas

Пототм изменили тип TCP-агента на Vegas (рис. 2.13).

- TCP Vegas часто приводит к более стабильной и «аккуратной» загрузке сети, снижая переполнение очереди и потери. Однако в смешанных сетях (где есть Reno/NewReno и Vegas одновременно) Vegas может проигрывать по пропускной способности более «агрессивным» вариантам, так как раньше снижает скорость при признаках роста задержек.

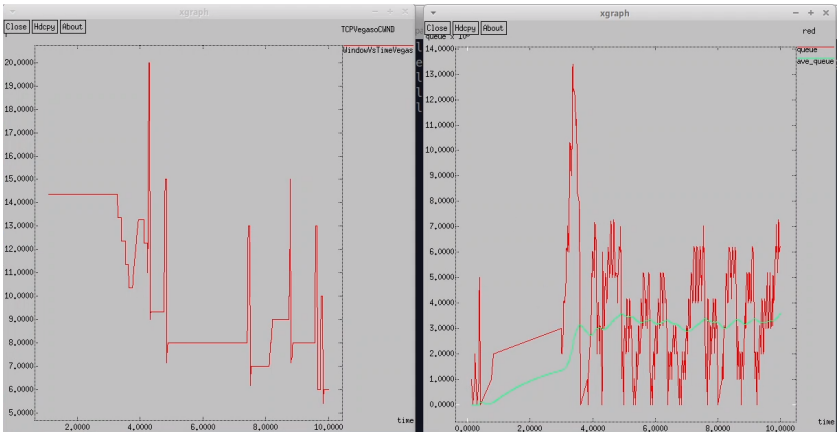

Рис. 2.13: TCP Vegas

2.1.12.4 Вывод

- **Reno**: наглядные колебания очереди и cwnd, резкие падения окна при потерях.
- **NewReno**: в целом схож с Reno, но лучше восстанавливается после множественных потерь, что делает его чуть более стабильным и эффективным.
- **Vegas**: стремится минимизировать потери и задержки, регулируя окно заранее на основе измерений **RTT**, поэтому обычно показывает более низкую и стабильную очередь и менее резкие скачки cwnd, но при этом может иметь более низкую пропускную способность в конкурентной среде.
- С точки зрения графиков, у Reno/NewReno красная линия (длина очереди) часто уходит довольно высоко, а затем падает, сопровождаясь резким снижением cwnd. У Vegas кривая очереди более сглажена, а **cwnd** меняется плавнее.

2.1.13 Внести изменения при отображении окон с графиками

- **Цвета траекторий:**
- `"0.Color: orange" — первая линия (размер очереди).`
- `"1.Color: cyan" — вторая линия (средняя длина очереди).`
- **Фон и цвет графиков:**
- `Фон (-bg black) — чёрный.`
- `Цвет линий (-fg gold) — золотой (но это влияет только на оси, заголовок и подписи).`
- **Заголовки:**
- `"Queue_Stats" — добавлен заголовок для второго графика.`

- "TCPNewrenoCWND" — название графика для размера окна TCP.
- **подписи к осям, подпись траектории в легенде**
- puts $f \n\"lenghtofochered"
- -x time -y queue

3 Выводы

- В ходе работы изучены механизмы управления окном перегрузки в TCP NewReno и динамика очереди. Графики показали зависимость этих параметров от времени, а изменения оформления улучшили их интерпретацию. Анализ подтвердил, что управление окном и очередью повышает эффективность сети.

Более подробно про RED см. в [[1]; [2];]

Список литературы

1. Stevens W.R. TCP/IP Illustrated, Volume 1: The Protocols. 1st изд. Addison-Wesley, 1994.
2. Tanenbaum A.S., Wetherall D.J. Computer Networks. 5th изд. Pearson, 2010.