Лабораторная работа 1. Простые модели компьютерной сети

Цель работы

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

1.1. Шаблон сценария для NS-2

В своём рабочем каталоге создайте директорию `mip`, к которой будут выполняться лабораторные работы. Внутри `mip` создайте директорию `lab-ns`, а в ней файл `shablon.tcl`:

```
mkdir -p mip/lab-ns
cd mip/lab-ns
touch shablon.tcl
```

Откройте на редактирование файл `shablon.tcl`. Можно использовать любой текстовый редактор типа emacs.

Сначала создадим объект типа Simulator:

```
# создание объекта  Simulator
set ns [new Simulator]
```

Затем создадим переменную `nf` и укажем, что требуется открыть на запись nam-файл для регистрации выходных результатов моделирования:

```
# 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
```

Вторая строка даёт команду симулятору записывать все данные о динамике модели в файл `out.nam`.

Далее создадим переменную `f` и откроем на запись файл трассировки для регистрации всех событий модели:

```
# 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
```

После этого добавим процедуру `finish`, которая закрывает файлы трассировки и запускает nam:

```
# процедура finish закрывает файлы трассировки
# и запускает визуализатор nam
proc finish {} {
  global ns f nf     # описание глобальных переменных
  $ns flush-trace    # прекращение трассировки
  close $f           # закрытие файлов трассировки
  close $nf          # закрытие файлов трассировки nam
```

```
    # запуск nam в фоновом режиме
    exec nam out.nam &
    exit 0
}
```

Наконец, с помощью команды `at` указываем планировщику событий, что процедуру `finish` следует запустить через 5 с после начала моделирования, после чего запустить симулятор `ns`:

```
# at-событие для планировщика событий, которое запускает
# процедуру finish через 5 с после начала моделирования
$ns at 5.0 "finish"

# запуск модели
$ns run
```

Сохранив изменения в отредактированном файле `shablon.tcl` и закрыв его, можно запустить симулятор командой:

```
ns shablon.tcl
```

При этом на экране появится сообщение типа

```
nam: empty trace file out.nam
```

поскольку ещё не определены никакие объекты и действия.

Получившийся шаблон можно использовать в дальнейшем в большинстве разрабатываемых скриптов NS-2, добавляя в него до строки `$ns at 5.0 "finish"` описание объектов и действий моделируемой системы.

1.2. Простой пример описания топологии сети, состоящей из двух узлов и одного соединения

Постановка задачи. Требуется смоделировать сеть передачи данных, состоящую из двух узлов, соединённых дуплексной линией связи с полосой пропускания 2 Мб/с и задержкой 10 мс, очередью с обслуживанием типа DropTail. От одного узла к другому по протоколу UDP осуществляется передача пакетов, размером 500 байт, с постоянной скоростью 200 пакетов в секунду.

Реализация модели. Скопируем содержимое созданного шаблона в новый файл:

```
cp shablon.tcl example1.tcl
```

и откроем `example1.tcl` на редактирование. Добавим в него до строки `$ns at 5.0 "finish"` описание топологии сети:

```
# создание 2-х узлов:
set N 2
for {set i 0} {$i < $N} {incr i} {
     set n($i) [$ns node]
}

# соединение 2-х узлов дуплексным соединением
# с полосой пропускания 2 Мб/с и задержкой 10 мс,
# очередью с обслуживанием типа  DropTail
$ns duplex-link $n(0) $n(1) 2Mb 10ms DropTail
```

Создадим агенты для генерации и приёма трафика:

```
# создание агента UDP и присоединение его к узлу n0
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# создание источника трафика CBR (constant bit rate)
set cbr0 [new Application/Traffic/CBR]

# устанавливаем размер пакета в 500 байт
$cbr0 set packetSize_ 500

#задаем интервал между пакетами равным 0.005 секунды,
#т.е. 200 пакетов в секунду
$cbr0 set interval_ 0.005

# присоединение источника трафика CBR к агенту udp0
$cbr0 attach-agent $udp0
```

Создаётся агент UDP и присоединяется к узлу n0. В узле агент сам не может генерировать трафик, он лишь реализует протоколы и алгоритмы транспортного уровня. Поэтому к агенту присоединяется приложение. В данном случае — это источник с постоянной скоростью (Constant Bit Rate, CBR), который каждые 5 мс посылает пакет $R = 500$ байт. Таким образом, скорость источника:

$$R = \frac{500 \cdot 8}{0,005} = 800000 \text{ бит/с}.$$

Далее создадим Null-агент, который работает как приёмник трафика, и прикрепим его к узлу `n1`:

```
# Создание агента-приёмника и присоединение его к узлу n(1)
set null0 [new Agent/Null]
$ns attach-agent $n(1) $null0
```

Соединим агенты между собой:

```
# Соединение агентов между собой
$ns connect $udp0 $null0
```

Для запуска и остановки приложения CBR добавляются at-события в планировщик событий (перед командой `$ns at 5.0 "finish"`)

```
# запуск приложения через 0,5 с
$ns at 0.5 "$cbr0 start"

# остановка приложения через 4,5 с
$ns at 4.5 "$cbr0 stop"
```

Сохранив изменения в отредактированном файле и запустив симулятор:

```
ns example1.tcl
```

получим в качестве результата запуск аниматора nam в фоновом режиме (рис. 1.1).

При нажатии на кнопку play в окне nam через 0.5 секунды из узла 0 данные начнут поступать к узлу 1. Это процесс можно замедлить, выбирая шаг отображения в nam. Можно осуществлять наблюдение за отдельным пакетом, щёлкнув по нему в окне nam, а щёлкнув по соединению, можно получить о нем некоторую информацию.

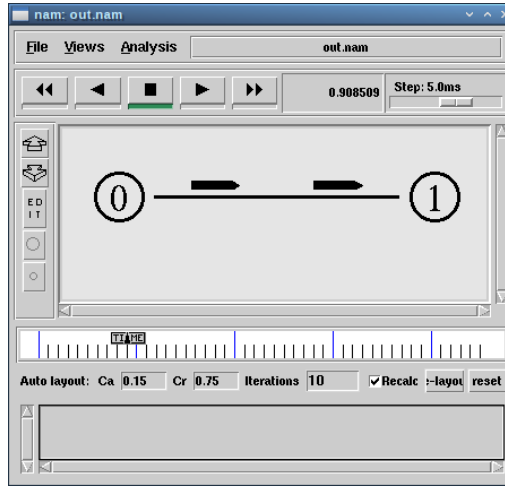

Рис. 1.1. Визуализация простой модели сети с помощью nam

1.3. Пример с усложнённой топологией сети

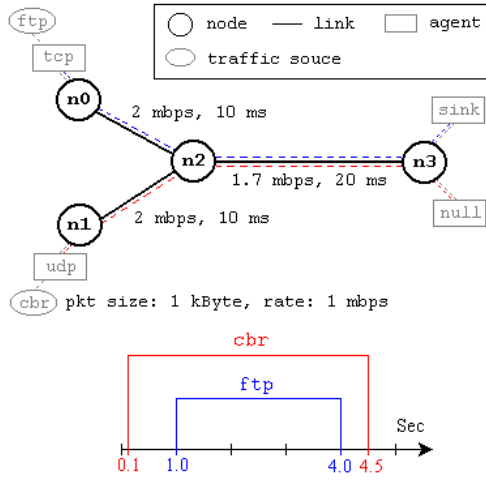

Рис. 1.2. Схема моделируемой сети

Постановка задачи. Описание моделируемой сети (рис. 2.4):
– сеть состоит из 4 узлов (n0, n1, n2, n3);

– между узлами n0 и n2, n1 и n2 установлено дуплексное соединение с пропускной способностью 2 Мбит/с и задержкой 10 мс;
– между узлами n2 и n3 установлено дуплексное соединение с пропускной способностью 1,7 Мбит/с и задержкой 20 мс;
– каждый узел использует очередь с дисциплиной DropTail для накопления пакетов, максимальный размер которой составляет 10;
– TCP-источник на узле n0 подключается к TCP-приёмнику на узле n3 (по-умолчанию, максимальный размер пакета, который TCP-агент может генерировать, равняется 1KByte)
– TCP-приёмник генерирует и отправляет ACK пакеты отправителю и откидывает полученные пакеты;
– UDP-агент, который подсоединён к узлу n1, подключён к null-агенту на узле n3 (null-агент просто откидывает пакеты);
– генераторы трафика ftp и cbr прикреплены к TCP и UDP агентам соответственно;
– генератор cbr генерирует пакеты размером 1 Кбайт со скоростью 1 Мбит/с;
– работа cbr начинается в 0,1 секунду и прекращается в 4,5 секунды, а ftp начинает работать в 1,0 секунду и прекращает в 4,0 секунды.

Реализация модели. Скопируем содержимое созданного шаблона в новый файл:

```
cp shablon.tcl example2.tcl
```

и откроем `example2.tcl` на редактирование.

Создадим 4 узла и 3 дуплексных соединения с указанием направления:

```
set N 4
for {set i 0} {$i < $N} {incr i} {
  set n($i) [$ns node]
}

$ns duplex-link $n(0) $n(2) 2Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 2Mb 10ms DropTail
$ns duplex-link $n(3) $n(2) 2Mb 10ms DropTail

$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(1) $n(2) orient right-up
$ns duplex-link-op $n(2) $n(3) orient right
```

Создадим агент UDP с прикреплённым к нему источником CBR и агент TCP с прикреплённым к нему приложением FTP:

```
# создание агента UDP и присоединение его к узлу n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# создание источника CBR-трафика
# и присоединение его к агенту udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

# создание агента TCP и присоединение его к узлу n(1)
set tcp1 [new Agent/TCP]
$ns attach-agent $n(1) $tcp1
```

```
# создание приложения FTP
# и присоединение его к агенту tcp1
set ftp [new Application/FTP]
$ftp attach-agent $tcp1
```

Создадим агенты-получатели:

```
# создание агента-получателя для udp0
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

# создание агента-получателя для tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n(3) $sink1
```

Соединим агенты udp0 и tcp1 и их получателей:

```
$ns connect $udp0 $null0
$ns connect $tcp1 $sink1
```

Зададим описание цвета каждого потока:

```
$ns color 1 Blue
$ns color 2 Red

$udp0 set class_ 1
$tcp1 set class_ 2
```

Отслеживание событий в очереди:

```
$ns duplex-link-op $n(2) $n(3) queuePos 0.5
```

Наложение ограничения на размер очереди:

```
$ns queue-limit $n(2) $n(3) 20
```

Добавление at-событий:

```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr0 stop"
```

Сохранив изменения в отредактированном файле и запустив симулятор, получим анимированный результат моделирования (рис. 1.3).

При запуске скрипта можно заметить, что по соединениям между узлами n(0)–n(2) и n(1)–n(2) к узлу n(2) передаётся данных больше, чем способно передаваться по соединению от узла n(2) к узлу n(3). Действительно, мы передаём 200 пакетов в секунду от каждого источника данных в узлах n(0) и n(1), а каждый пакет имеет размер 500 байт. Таким образом, полоса каждого соединения $0,8$ Mb, а суммарная — $1,6$ Mb. Но соединение n(2)–n(3) имеет полосу лишь 1 Mb. Следовательно, часть пакетов должна теряться. В окне аниматора можно видеть пакеты в очереди, а также те пакеты, которые отбрасываются при переполнении.

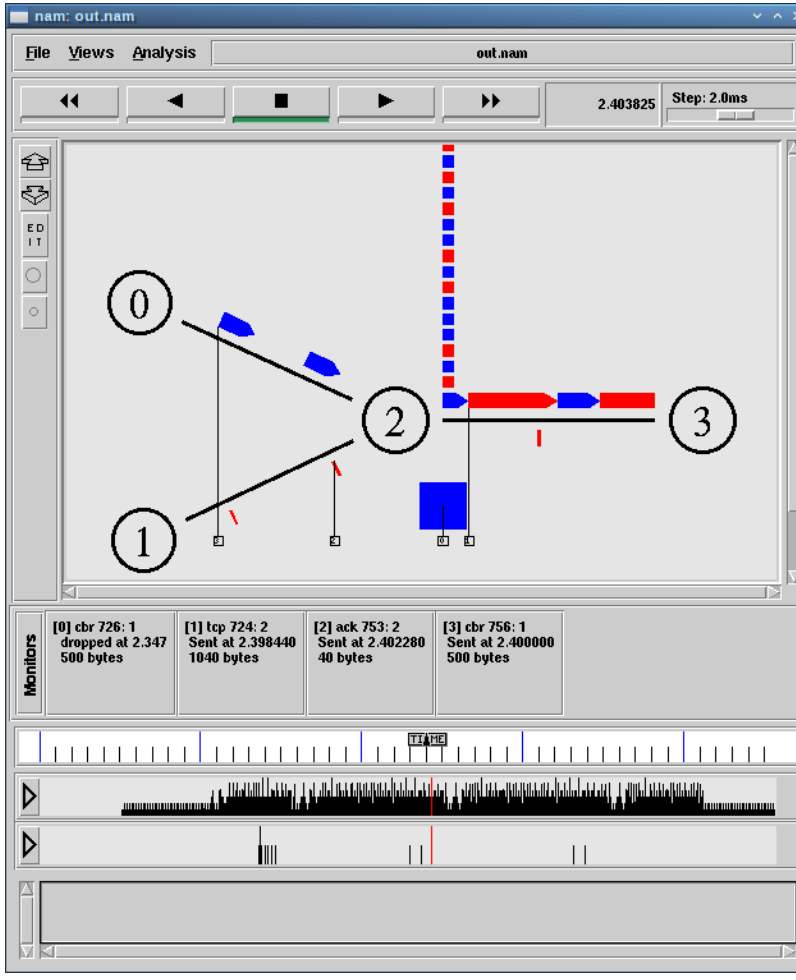

Рис. 1.3. Мониторинг очереди в визуализаторе nam

1.4. Пример с кольцевой топологией сети

Постановка задачи. Требуется построить модель передачи данных по сети с кольцевой топологией и динамической маршрутизацией пакетов:

- сеть состоит из 7 узлов, соединённых в кольцо;
- данные передаются от узла `n(0)` к узлу `n(3)` по кратчайшему пути;
- с 1 по 2 секунду модельного времени происходит разрыв соединения между узлами `n(1)` и `n(2)`;

– при разрыве соединения маршрут передачи данных должен измениться на резервный.

Реализация модели. Скопируем содержимое созданного шаблона в новый файл:

```
cp shablon.tcl example3.tcl
```

и откроем `example3.tcl` на редактирование.

Опишем топологию моделируемой сети:

```
set N 7
for {set i 0} {$i < $N} {incr i} {
  set n($i) [$ns node]
}
```

Далее соединим узлы так, чтобы создать круговую топологию:

```
for {set i 0} {$i < $N} {incr i} {
  $ns duplex-link $n($i) $n([expr ($i+1)%$N]) 1Mb 10ms DropTail
}
```

Каждый узел, за исключением последнего, соединяется со следующим, последний соединяется с первым. Для этого в цикле использован оператор %, означающий остаток от деления нацело.

Зададим передачу данных от узла `n(0)` к узлу `n(3)`:

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Agent/CBR]
$ns attach-agent $n(0) $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

$ns connect $cbr0 $null0
```

Данные передаются по кратчайшему маршруту от узла `n(0)` к узлу `n(3)`, через узлы `n(1)` и `n(2)` (рис. 1.4).

Добавим команду разрыва соединения между узлами `n(1)` и `n(2)` на время в одну секунду, а также время начала и окончания передачи данных:

```
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
```

Передача данных при кольцевой топологии сети в случае разрыва соединения представлена на рис. 1.5.

Добавив в начало скрипта после команды создания объекта Simulator:

```
$ns rtproto DV
```

увидим, что сразу после запуска в сети отправляется небольшое количество маленьких пакетов, используемых для обмена информацией, необходимой для маршрутизации между узлами (рис. 1.6). Когда соединение будет разорвано, информация о топологии будет обновлена, и пакеты будут отсылаться по новому маршруту через узлы `n(6)`, `n(5)` и `n(4)`.

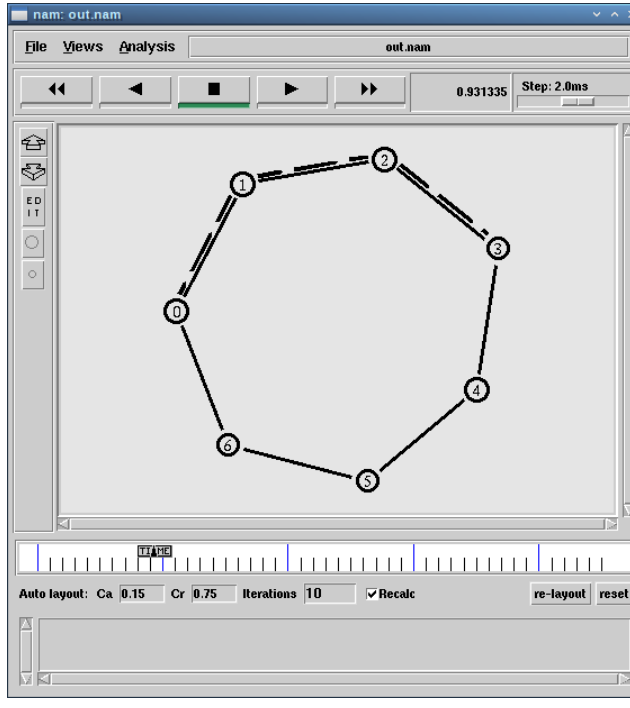

Рис. 1.4. Передача данных по кратчайшему пути сети с кольцевой топологией

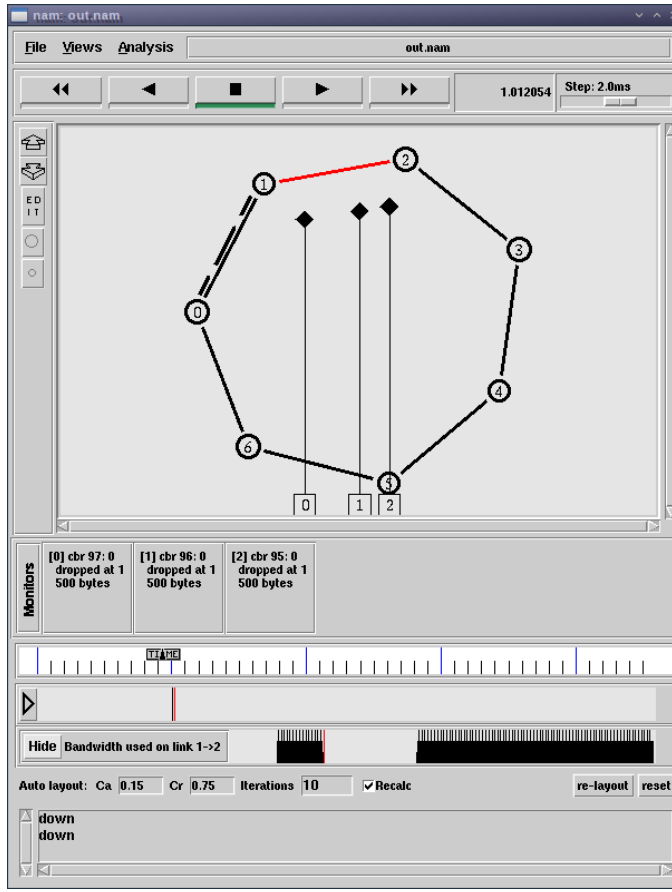

Рис. 1.5. Передача данных по сети с кольцевой топологией в случае разрыва соединения

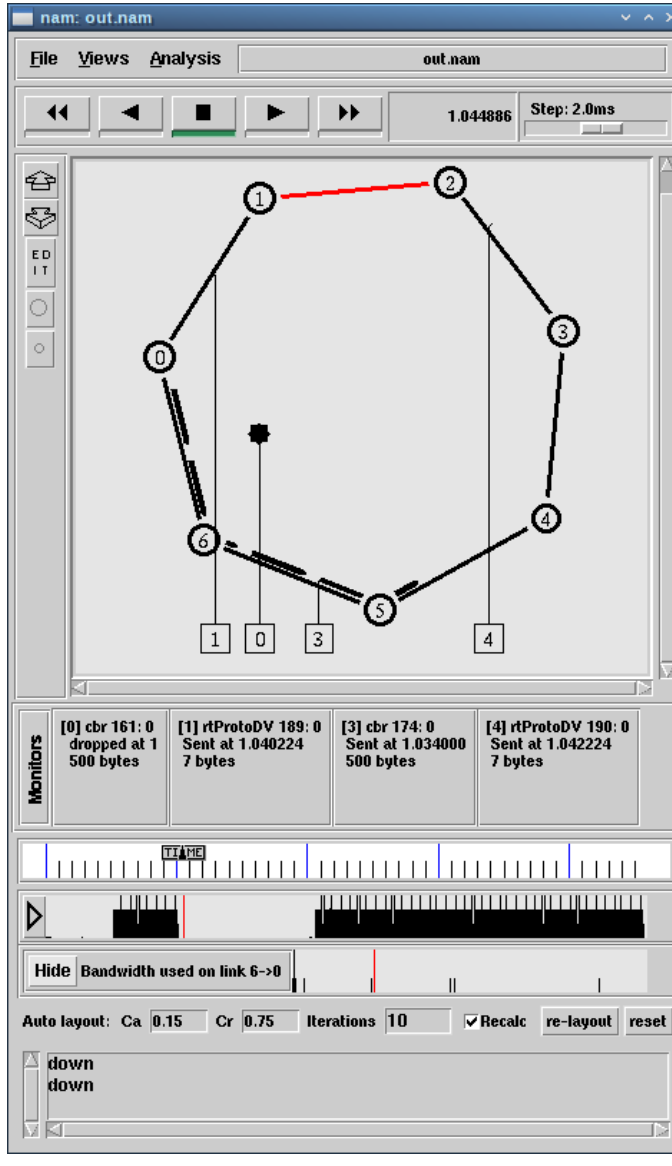

Рис. 1.6. Маршрутизация данных по сети с кольцевой топологией в случае разрыва соединения

Упражнение Внесите следующие изменения в реализацию примера с кольцевой топологией сети:

– топология сети должна соответствовать представленной на рис. 1.7;

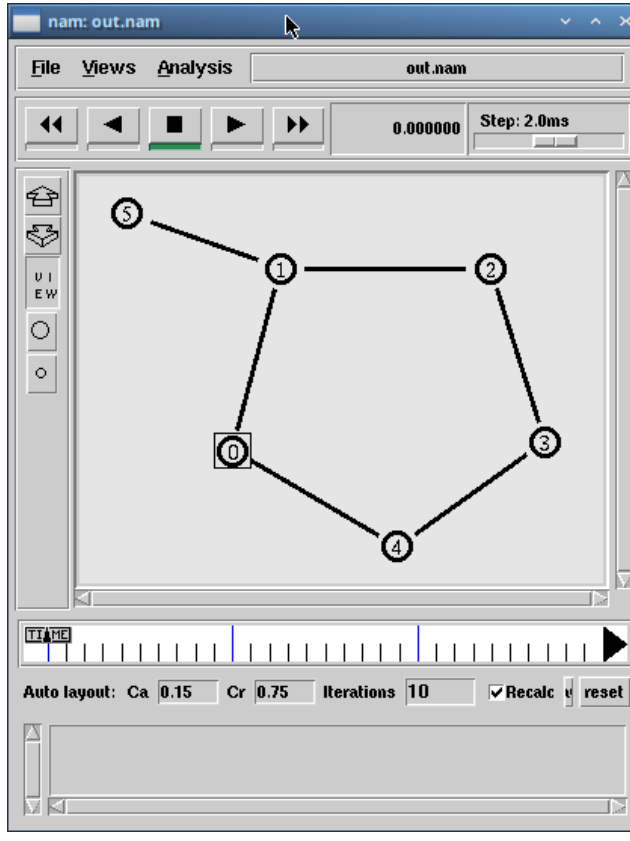

Рис. 1.7. Изменённая кольцевая топология сети

– передача данных должна осуществляться от узла `n(0)` до узла `n(5)` по кратчайшему пути в течение 5 секунд модельного времени;
– передача данных должна идти по протоколу TCP (тип Newreno), на принимающей стороне используется TCPSink-объект типа DelAck; поверх TCP работает протокол FTP с 0,5 до 4,5 секунд модельного времени;
– с 1 по 2 секунду модельного времени происходит разрыв соединения между узлами `n(0)` и `n(1)`;
– при разрыве соединения маршрут передачи данных должен измениться на резервный, после восстановления соединения пакеты снова должны пойти по кратчайшему пути.