

Лабораторная работа 1. Простые модели компьютерной сети

Абакумова Олеся Максимовна, НФИбд-02-22

Содержание

1	Цель работы	6
2	Теоретическое введение	7
3	Выполнение лабораторной работы	8
3.1	Шаблон сценария для NS-2	8
3.2	Простой пример описания топологии сети, состоящей из двух узлов и одного соединения	10
3.3	Пример с усложнённой топологией сети	13
3.4	Пример с кольцевой топологией сети	17
3.5	Упражнение	20
4	Выводы	24
	Список литературы	25

Список иллюстраций

3.1	Создание рабочих директорий и файла	8
3.2	Создание объекта и переменной вместе с требованием	8
3.3	Создание переменной f и открытие на запись файла трассировки .	9
3.4	Добавление процедуры	9
3.5	Добавление процедуры(продолжение)	9
3.6	Указания для планировщика событий	10
3.7	Запуск симулятора	10
3.8	Копирование шаблона	11
3.9	Добавление строк для описания новой топологии сети	11
3.10	Создание агентов	11
3.11	Создание агента-приемника и соединение между ними	12
3.12	Добавление at-событий	12
3.13	Запуск example1.tcl	13
3.14	Создание example2.tcl	14
3.15	Создание узлов и соединения	14
3.16	Создание агента UDP и TCP	15
3.17	Создание агента UDP и TCP(продолжение)	15
3.18	Создание агентов-получателей	15
3.19	Соединение агентов и получателей,описание цветов и отслежива- ние событий	15
3.20	Наложение ограничения и добавление at-событий	16
3.21	Запуск example2.tcl	16
3.22	Создание example3.tcl	17
3.23	Создание example3.tcl	17
3.24	Задание передачи данных от узла к узлу	18
3.25	Добавление команды разрыва и времени начала и окончания для передачи данных	18
3.26	Добавление строки в начало	18
3.27	Передача данных по кратчайшему пути сети с кольцевой топологией	19
3.28	Передача данных по сети с кольцевой топологией в случае разрыва соединения	19
3.29	Маршрутизация данных по сети с кольцевой топологией в случае разрыва соединения	20
3.30	Содержание файла example4.tcl	21
3.31	Пример кольцевой топологии	21
3.32	Путь передачи данных	22
3.33	Разрыв между узлами	22

3.34 Резервный маршрут	23
3.35 Восстановленный маршрут передачи пакетов	23

Список таблиц

1 Цель работы

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

2 Теоретическое введение

Network Simulator (NS-2) — один из программных симуляторов моделирования процессов в компьютерных сетях. NS-2 позволяет описать топологию сети, конфигурацию источников и приёмников трафика, параметры соединений (полосу пропускания, задержку, вероятность потерь пакетов и т.д.) и множество других параметров моделируемой системы. Данные о динамике трафика, состоянии соединений и объектов сети, а также информация о работе протоколов фиксируются в генерируемом trace-файле.

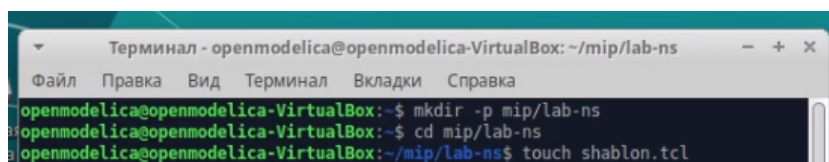
NS-2 является объектно-ориентированным программным обеспечением. Его ядро реализовано на языке C++. В качестве интерпретатора используется язык скриптов (сценариев) OTcl (Object oriented Tool Command Language). NS-2 полностью поддерживает иерархию классов C++ и подобную иерархию классов интерпретатора OTcl. Обе иерархии обладают идентичной структурой, т.е. существует однозначное соответствие между классом одной иерархии и таким же классом другой. Объединение для совместного функционирования C++ и OTcl производится при помощи TclCl (Classes Tcl). В случае, если необходимо реализовать какую-либо специфическую функцию, не реализованную в NS-2 на уровне ядра, для этого используется код на C++.

Более подробно про NS-2 см. в [1].

3 Выполнение лабораторной работы

3.1 Шаблон сценария для NS-2

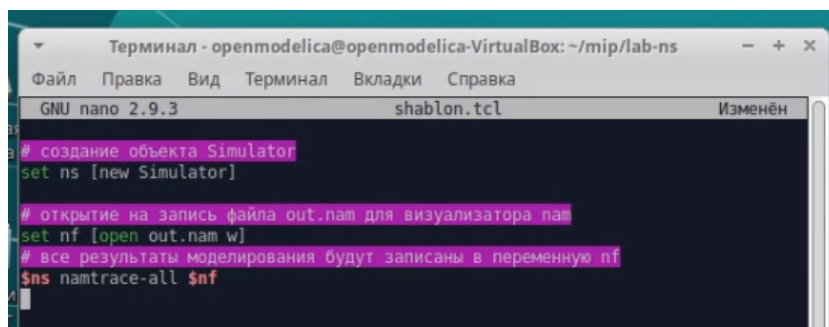
В своём рабочем каталоге создадим директорию `mip`, к которой будут выполняться лабораторные работы. Внутри `mip` создадим директорию `lab-ns`, а в ней файл `shablon.tcl` (рис. 3.1):



```
Терминал - openmodelica@openmodelica-VirtualBox: ~/mip/lab-ns
Файл  Правка  Вид  Терминал  Вкладки  Справка
openmodelica@openmodelica-VirtualBox:~$ mkdir -p mip/lab-ns
openmodelica@openmodelica-VirtualBox:~$ cd mip/lab-ns
openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$ touch shablon.tcl
```

Рис. 3.1: Создание рабочих директорий и файла

Откроем на редактирование файл `shablon.tcl`. Можно использовать любой текстовый редактор типа `emacs`. Я использую `nano`. Создадим объект типа `Simulator`, переменную `nf` и укажем, что требуется открыть на запись `nam`-файл для регистрации выходных результатов моделирования (рис. 3.2):



```
Терминал - openmodelica@openmodelica-VirtualBox: ~/mip/lab-ns
Файл  Правка  Вид  Терминал  Вкладки  Справка
GNU nano 2.9.3      shablon.tcl      Изменён
# 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
```

Рис. 3.2: Создание объекта и переменной вместе с требованием

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 (рис. 3.3):

```
# 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
```

Рис. 3.3: Создание переменной f и открытие на запись файла трассировки

После этого добавим процедуру finish, которая закрывает файлы трассировки и запускает nam (рис. 3.4):

```
$ns trace-all $f
# 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
```

Рис. 3.4: Добавление процедуры

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

Рис. 3.5: Добавление процедуры(продолжение)

Наконец, с помощью команды at указываем планировщику событий, что процедуру finish следует запустить через 5 с после начала моделирования, после чего запустить симулятор ns (рис. 3.6):



Рис. 3.6: Указания для планировщика событий

Сохранив изменения в отредактированном файле shablon.tcl и закрыв его, можно запустить симулятор командой (рис. 3.7):

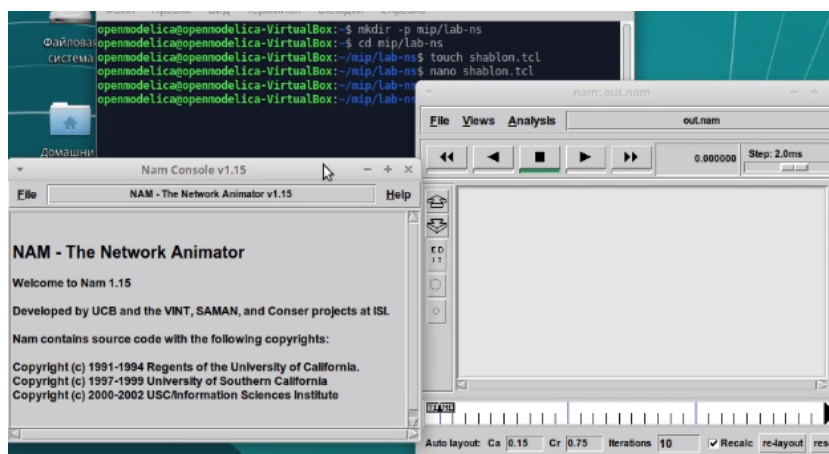


Рис. 3.7: Запуск симулятора

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.

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

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.

Скопируем содержимое созданного шаблона в новый файл (рис. 3.8):

```

openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$ ns shablon.tcl
openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$ cp shablon.tcl example1.tcl
openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$

```

Рис. 3.8: Копирование шаблона

Открыв example1.tcl на редактирование, добавим в него до строки \$ns at 5.0 “finish” описание топологии сети (рис. 3.9):

```

# 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.
# очередь с обслуживанием типа DropTail
$ns duplex-link $n(0) $n(1) 2Mb 10ms DropTail
# at-событие для планировщика событий, которое запускает
# процедуру finish через 5 с после начала моделирования
$ns at 5.0 "finish"
# запуск модели
$ns run

```

Рис. 3.9: Добавление строк для описания новой топологии сети

Создадим агенты для генерации и приёма трафика (рис. 3.10):

```

# 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

```

Рис. 3.10: Создание агентов

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:

$$R = \frac{500 \times 8}{0.005} = 800000$$

Далее создадим Null-агент, который работает как приёмник трафика, и прикрепим его к узлу n1, соединим их между собой (рис. 3.11):

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

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

Рис. 3.11: Создание агента-приемника и соединение между ними

Для запуска и остановки приложения CBR добавляются at-события в планировщик событий (перед командой \$ns at 5.0 “finish”) (рис. 3.12):

```
# запуск приложения через 0,5 с
$ns at 0.5 "$cbr0 start"
# остановка приложения через 4,5 с
$ns at 4.5 "$cbr0 stop"
```

Рис. 3.12: Добавление at-событий

Сохранив изменения в отредактированном файле и запустив симулятор (рис. 3.13):

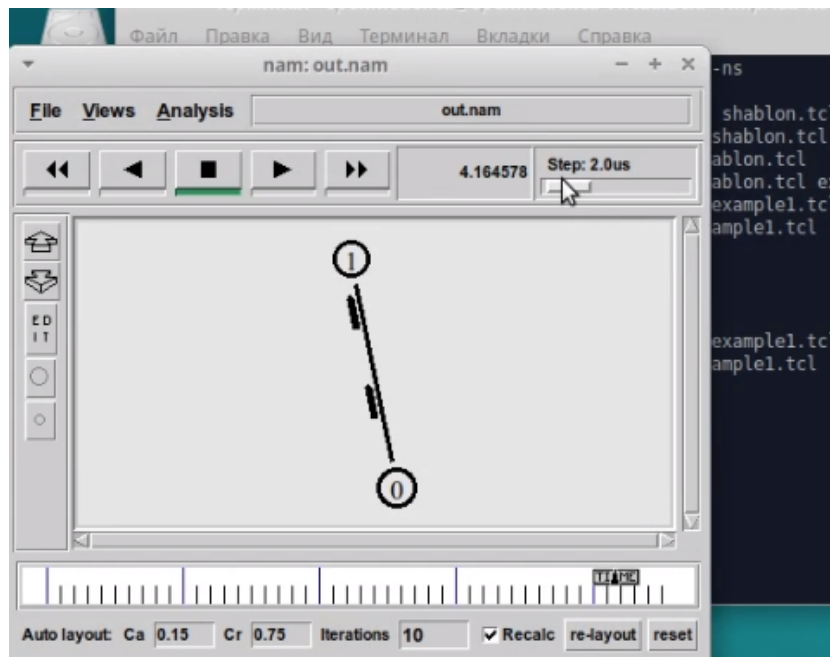


Рис. 3.13: Запуск example1.tcl

получим в качестве результата запуск аниматора nam в фоновом режиме.

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.

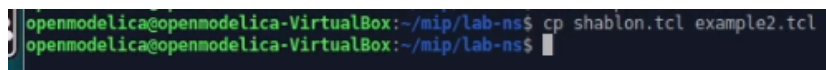
3.3 Пример с усложнённой топологией сети

Описание моделируемой сети:

- 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.

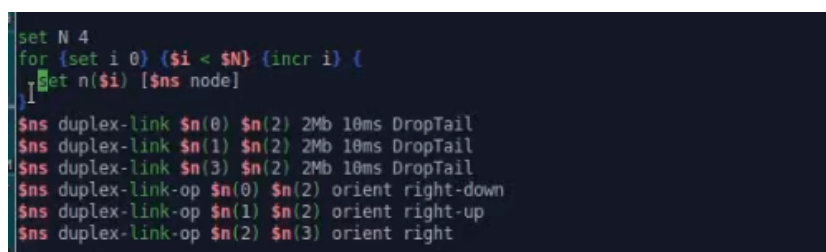
Скопируем содержимое созданного шаблона в новый файл и откроем example2.tcl на редактирование (рис. 3.14):



```
openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$ cp shablon.tcl example2.tcl
openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$
```

Рис. 3.14: Создание example2.tcl

Создадим 4 узла и 3 дуплексных соединения с указанием направления (рис. 3.15):



```
set N 4
for {set i 0} {$i < $N} {incr i} {
    get n($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
```

Рис. 3.15: Создание узлов и соединения

Создадим агент UDP с прикреплённым к нему источником CBR и агент TCP с

прикреплённым к нему приложением FTP (рис. 3.16):

```
# 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
```

Рис. 3.16: Создание агента UDP и TCP

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

Рис. 3.17: Создание агента UDP и TCP(продолжение)

Создадим агенты-получатели (рис. 3.18):

```
# 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

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

Рис. 3.18: Создание агентов-получателей

Соединим агенты udp0 и tcp1 и их получателей, описание цвета каждого потока и отслеживание событий в очереди (рис. 3.19):

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

$ns color 1 Blue
$ns color 2 Red

$udp0 set class_ 1
$tcp1 set class_ 2

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

Рис. 3.19: Соединение агентов и получателей, описание цветов и отслеживание событий

Наложение ограничения на размер очереди и добавление at-событий (рис. 3.20):

```
$ns queue-limit $n(2) $n(3) 20
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr0 stop"
```

Рис. 3.20: Наложение ограничения и добавление at-событий

Сохранив изменения в отредактированном файле и запустив симулятор, получим анимированный результат моделирования (рис. 3.21):

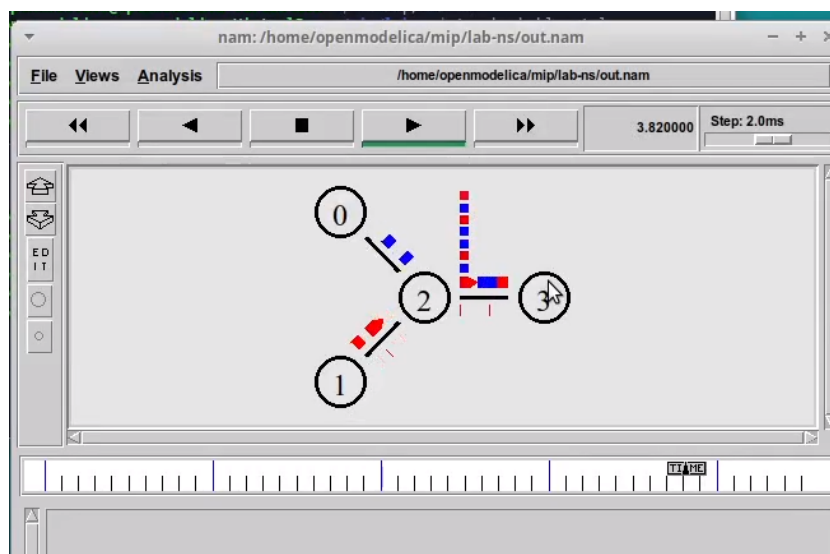


Рис. 3.21: Запуск example2.tcl

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 Мб, а суммарная – 1, 6 Мб. Но соединение $n(2)$ – $n(3)$ имеет полосу лишь 1 Мб. Следовательно, часть пакетов должна теряться. В окне аниматора можно видеть пакеты в очереди, а также те пакеты, которые отбрасываются при переполнении.

3.4 Пример с кольцевой топологией сети

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.

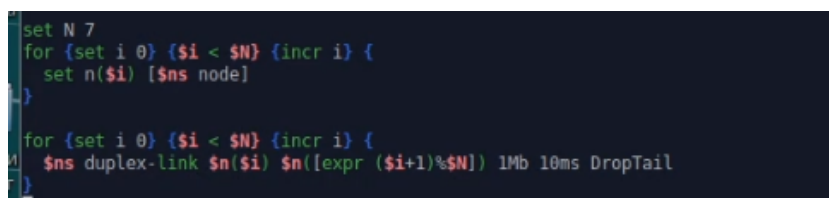
Скопируем содержимое созданного шаблона в новый файл и откроем `example3.tcl` на редактирование (рис. 3.22):



```
openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$ cp shablon.tcl example3.tcl
openmodelica@openmodelica-VirtualBox:~/mip/lab-ns$
```

Рис. 3.22: Создание `example3.tcl`

Опишем топологию моделируемой сети и соединим узлы так, чтобы создать круговую топологию (рис. 3.23):



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

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

Рис. 3.23: Создание `example3.tcl`

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 $n(3)$ (рис. 3.24):

```

a 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

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

  $ns connect $cbr0 $null0

```

Рис. 3.24: Задание передачи данных от узла к узлу

Данные передаются по кратчайшему маршруту от узла $n(0)$ к узлу $n(3)$, через узлы $n(1)$ и $n(2)$ (рис. 3.27).

Добавим команду разрыва соединения между узлами $n(1)$ и $n(2)$ на время в одну секунду, а также время начала и окончания передачи данных (рис. 3.25):

```

а $ns connect $cbr0 $null0

  $ns at 0.5 "$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"

```

Рис. 3.25: Добавление команды разрыва и времени начала и окончания для передачи данных

Передача данных при кольцевой топологии сети в случае разрыва соединения (рис. 3.28).

Добавив в начало скрипта после команды создания объекта Simulator (рис. 3.26):

```

Терминал - openmodelica@openmodelica-VirtualBox: ~/mip/lab-ns
Файл  Правка  Вид  Терминал  Вкладки  Справка
GNU nano 2.9.3  example3.tcl  Изменён

# создание объекта Simulator
set ns [new Simulator]

$ns rtproto DV

```

Рис. 3.26: Добавление строки в начало

увидим, что сразу после запуска в сети отправляется небольшое количество маленьких пакетов, используемых для обмена информацией, необходимой для

маршрутизации между узлами (рис. 3.29). Когда соединение будет разорвано, информация о топологии будет обновлена, и пакеты будут отсылаться по новому маршруту через узлы $n(6)$, $n(5)$ и $n(4)$.

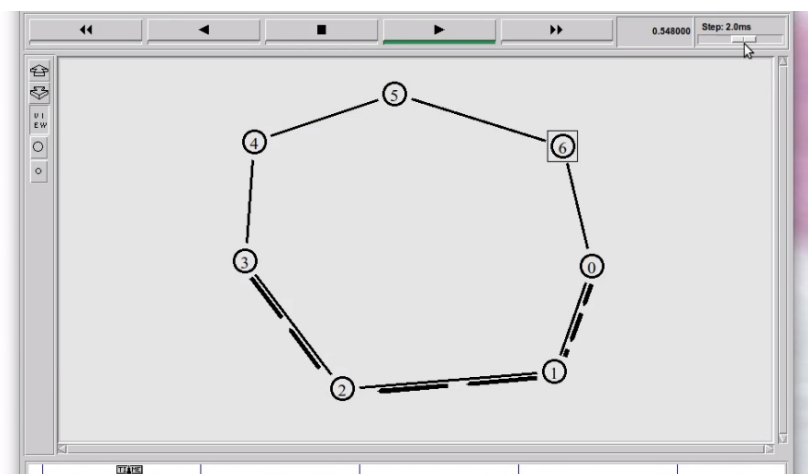


Рис. 3.27: Передача данных по кратчайшему пути сети с кольцевой топологией

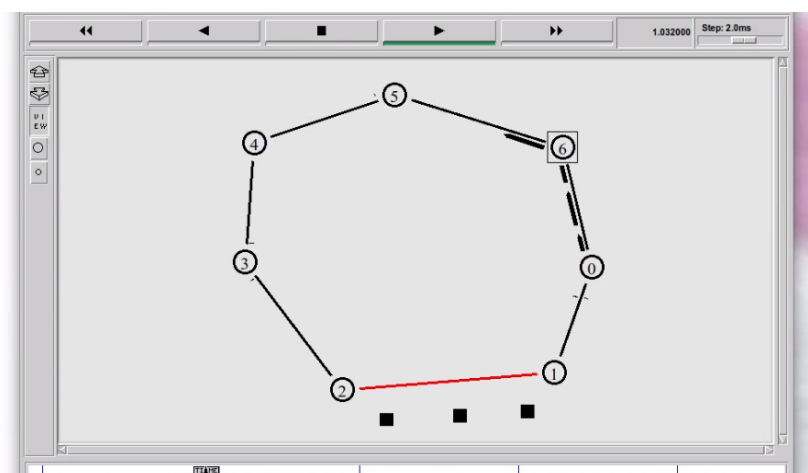


Рис. 3.28: Передача данных по сети с кольцевой топологией в случае разрыва соединения

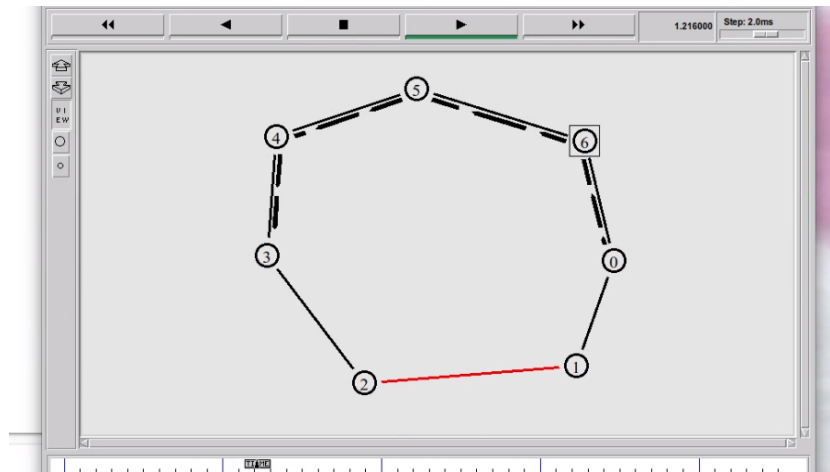


Рис. 3.29: Маршрутизация данных по сети с кольцевой топологией в случае разрыва соединения

3.5 Упражнение

Внесем следующие изменения в реализацию примера с кольцевой топологией сети: – топология сети должна соответствовать представленной на рисунке (рис. 3.31);

- передача данных должна осуществляться от узла $n(0)$ до узла $n(5)$ по кратчайшему пути в течение 5 секунд модельного времени (рис. 3.32);

- 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 $n(1)$ (рис. 3.33);

- при разрыве соединения маршрут передачи данных должен измениться на резервный (рис. 3.34), после восстановления соединения пакеты снова должны пойти по кратчайшему пути (рис. 3.35).

Реализация:

Скопировав шаблон в новый файл `example4.tcl` и открыв его на редактировании, пропишем следующее (рис. 3.30):

```

set N 5
for {set i 0} {$i < $N} {incr i} {
    set n($i) [$ns node]
}

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

set n5 [$ns node]

$ns duplex-link $n5 $n(1) 1Mb 10ms DropTail

set tcp1 [new Agent/TCP/Newreno]
$ns attach-agent $n(0) $tcp1

set ftp [new Application/FTP]
$ftp attach-agent $tcp1

set sink1 [new Agent/TCP/Sink/DelAck]
$ns attach-agent $n5 $sink1
$ns connect $tcp1 $sink1

$ns at 0.5 "$ftp start"
$ns rtmodel-at 1.0 down $n(0) $n(1)
$ns rtmodel-at 2.0 up $n(0) $n(1)
$ns at 4.5 "$ftp stop"
$ns at 5.0 "finish"

```

Рис. 3.30: Содержание файла example4.tcl

При запуске симуляции, мы получаем кольцевую топологию вида (рис. 3.31):

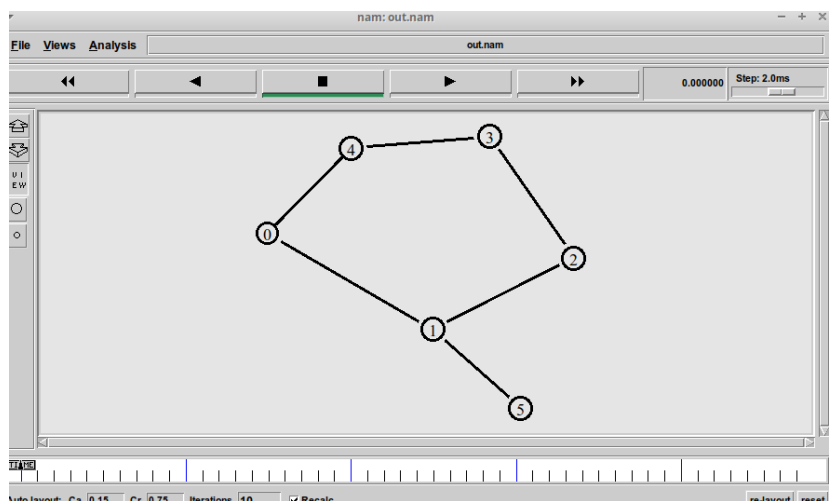


Рис. 3.31: Пример кольцевой топологии

Передача данных от узла $n(0)$ до узла $n(5)$ осуществляется по кратчайшему пути (рис. 3.32):

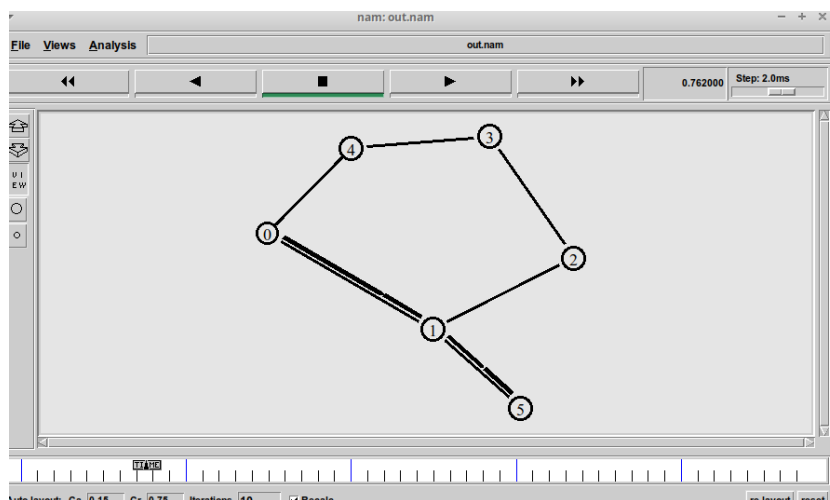


Рис. 3.32: Путь передачи данных

Далее происходит разрыв соединения между узлами $n(0)$ и $n(1)$ с 1 по 2 секунду модельного времени (рис. 3.33):

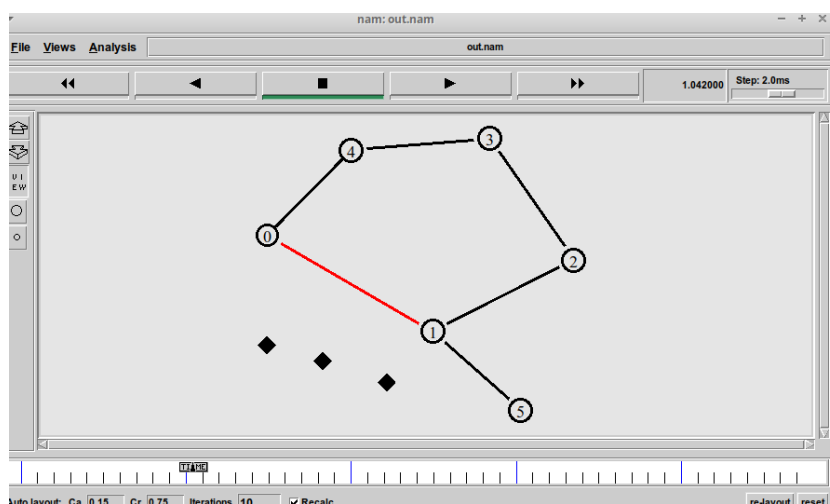


Рис. 3.33: Разрыв между узлами

После разрыва соединения, маршрут перестроен на резервный (рис. 3.34):

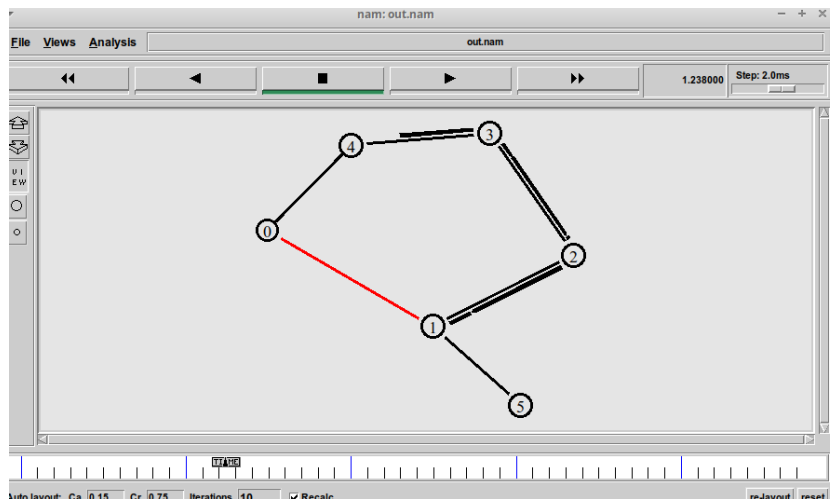


Рис. 3.34: Резервный маршрут

После восстановления соединения, пакеты снова пошла по кратчайшему пути (рис. 3.35):

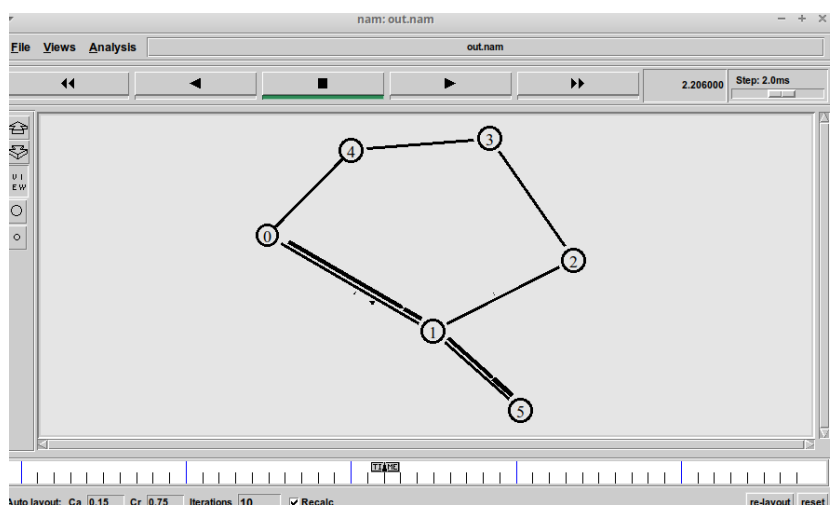


Рис. 3.35: Восстановленный маршрут передачи пакетов

4 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки моделирования сетей передачи данных с помощью средства имитационного моделирования NS-2, а также провела анализ полученных результатов моделирования.

Список литературы

1. Королькова А.В., Кулябов Д.С. Моделирование информационных процессов. МСК.: Российский университет дружбы народов, 2014. 191 с.