

Servicios de Red
Práctica 1
Introducción al Network Simulator

Himar Alonso Díaz

6 de marzo de 2006

Contenido

1. Ejemplo 1	3
1.1. Descripción del programa. Código fuente	3
1.2. Funcionamiento	4
2. Ejemplo 2	5
2.1. Descripción del programa. Código fuente	5
2.2. Funcionamiento	7
3. Ejemplo 3	9
3.1. Descripción del programa. Código fuente	9
3.2. Funcionamiento	10

Introducción

En esta primera práctica se pretende hacer una introducción al simulador de redes de ordenadores *Network Simulator*. El uso de este simulador está basado en la realización de *scripts* en lenguaje TCL. Estos *scripts* son procesados por el *kernel* del programa, mediante la instrucción “ns fichero.tcl”. Luego podremos visualizar la salida haciendo uso del programa “NAM” (*Network Animator*).

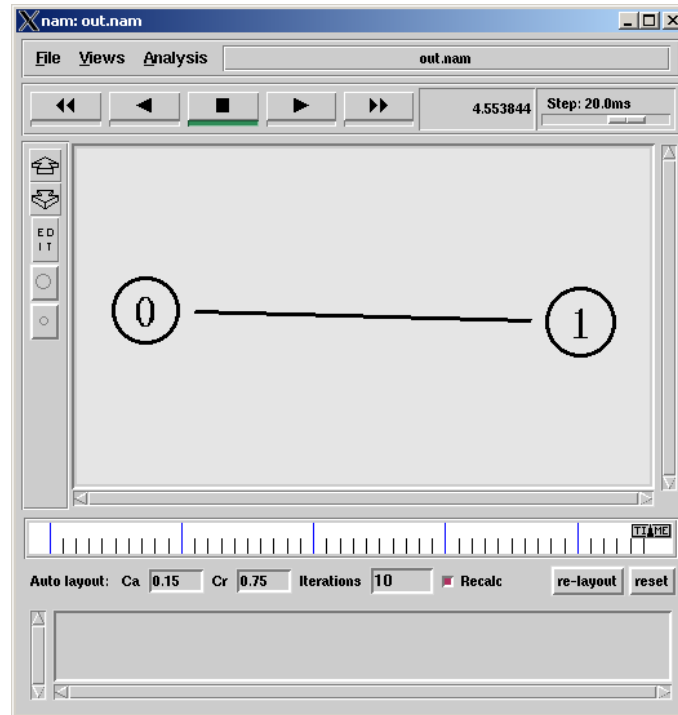


Figura 1: Ejemplo de visualización con NAM

A continuación se describirá el funcionamiento del *Network Simulator* haciendo uso de tres ejemplos sencillos, basados en tráfico UDP.

1. Ejemplo 1

1.1. Descripción del programa. Código fuente

El primero de los *scripts* que vamos a hacer consta de dos nodos unidos por un enlace dúplex, con un ancho de banda e 1Mb, retardo de 10ms y tipo de cola DropTail. El protocolo usado es UDP, el tipo de tráfico entre los dos nodos es CBR, el tamaño de los paquetes es de 500 bytes y la frecuencia con la que se envían es de 200 paquetes por segundo.

Creación de un objeto simulador. Este es el objeto principal del programa, sobre él se realizarán todas las operaciones:

```
set ns [new Simulator]
```

Es necesario crear un fichero para visualizar la salida con "nam". Llamaremos al fichero "out.nam" y permanecerá abierto para volcar la información al final:

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Definición del evento "finish". Este procedimiento se ejecutará para finalizar el programa. Su misión es volcar la información de la simulación en el fichero "out.nam", cerrar el archivo y ejecutar el nam, para mostrar directamente la visualización:

```
proc finish {} {
    global ns nf f0
    $ns flush-trace
    close $nf
    exec nam out.nam &
    exit 0
}
```

Creamos dos nodos para simular una transferencia de uno a otro:

```
set n0 [$ns node]
set n1 [$ns node]
```

Creación de un enlace dúplex para unir los nodos "n0" y "n1":

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

Creamos un Agente UDP y lo asociamos al nodo "n0":

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

El Agente UDP no puede transmitir información a la red por sí solo. Tenemos que asociarle una fuente de tráfico tipo CBR (Constant Bit Rate) para el agente "udp0":

```
set cbr0 [new Application/Traffic/CBR]
```

Especificamos el tamaño tamaño (en bytes) de cada paquete:

```
$cbr0 set packetSize_500
```

Especificamos el período con el que se enviarán los paquetes. Si queremos una frecuen-

cia de 200 paquetes por segundo, el período es justamente la inversa, o sea 0.005:

```
$cbr0 set interval_0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

Conectamos el Agente UDP con el "traffic sink" o "sumidero de tráfico", cuya misión es recoger los paquetes y desecharlos:

```
$ns connect $udp0 $null0
```

Planificación: Con estas sentencias indicaremos al generador de tráfico CBR cuándo debe empezar a emitir paquetes, y cuándo debe parar. El tiempo se indica en segundos:

```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
```

Esta instrucción forma parte de la planificación. Indica que en el instante de tiempo 5 se ejecute el procedimiento "finish", definido anteriormente:

```
$ns at 5.0 "finish"
```

Finalmente, y cuando todos los parámetros de la simulación han sido definidos, la ejecutamos:

```
$ns run
```

1.2. Funcionamiento

El funcionamiento de este primer ejemplo es muy sencillo: Tenemos dos nodos, en uno se encuentra la fuente de datos y en otro el sumidero, de modo que, de acuerdo con la planificación, en el instante $t = 0,5$ s comienzan a transmitirse los paquetes a través del enlace, y en el instante $t = 4,5$ s la fuente para.

Uno de los problemas que se planteaba es que si la simulación terminaba en el instante $t = 5$ s, ¿sería posible que los últimos paquetes no llegaran al destino, ya que la fuente estaría emitiendo hasta justamente 0,5s antes? Al ejecutar la visualización comprobamos que efectivamente sí llegan todos los paquetes, ya que el tiempo que tarda en llegar cada paquete es mucho menor que ese intervalo.

En la Figura 2 se muestra una captura donde se aprecian las flechas que simbolizan la transmisión de paquetes.

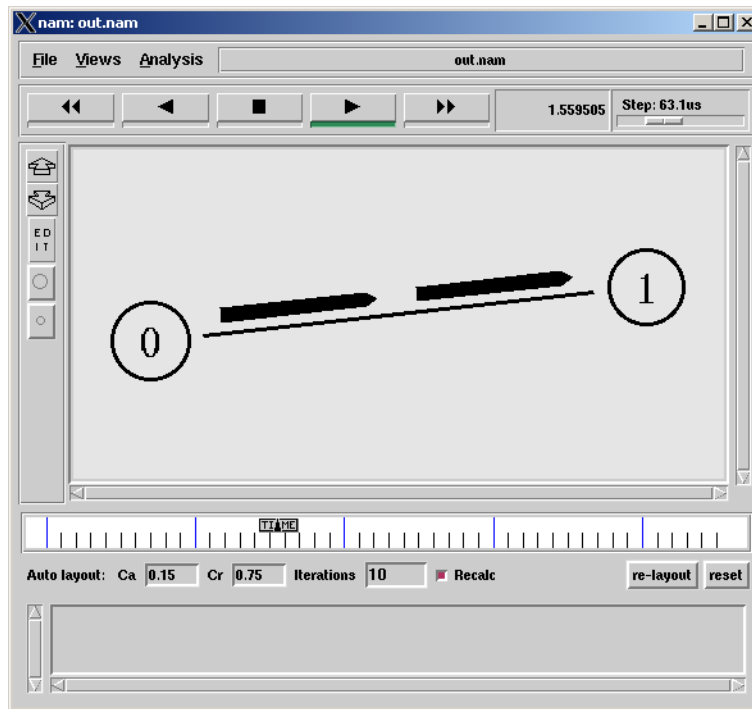


Figura 2: Captura de la visualización del Ejemplo 1 con NAM

2. Ejemplo 2

2.1. Descripción del programa. Código fuente

Este ejemplo está basado en el anterior. Veamos qué efecto produce tener dos nodos fuente. Para ello, además de crear los nodos *fuentes* tendremos que utilizar un nodo *router* y un nodo final (sumidero).

Una de las diferencias importantes es que el nodo “n2” tendrá una *cola SFQ* donde irá almacenando los paquetes procedentes de los nodos fuente, a modo de *buffer*. Finalmente los enviará hacia el nodo sumidero.

Creamos el objeto simulador:
`set ns [new Simulator]`

Definimos los colores que luego utilizaremos para distinguir la procedencia de los paquetes:
`$ns color 1 Blue`
`$ns color 2 Red`

Abrimos el fichero donde volcaremos el resultado para luego visualizarlo con el nam. Definimos también el procedimiento "finish" para volcar el resultado de la simulación y ejecutar el nam:
`set nf [open out.nam w]`
`$ns namtrace-all $nf`

`proc finish {} {`

```

    global ns nf f0
    $ns flush-trace
    close $nf
    exec nam out.nam &
    exit 0
}

```

En este ejemplo necesitaremos definir cuatro nodos:

```

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

```

Creamos los enlaces de los nodos según la definición del enunciado:

```

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link $n2 $n3 1Mb 10ms SFQ
$ns duplex-link-op $n2 $n3 orient right

```

La siguiente instrucción nos permitirá monitorizar las colas:

```

$ns duplex-link-op $n2 $n3 queuePos 0.5

```

Creamos los Agentes UDP de cada uno de los nodos emisores, y para distinguir los paquetes, le asignamos un color diferente:

```

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
$udp0 set fid.1 # Color Azul
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
$udp1 set fid.2 # Color Rojo

```

Creamos las fuentes de tráfico de tipo CBR:

```

set cbr0 [new Application/Traffic/CBR]
set cbr1 [new Application/Traffic/CBR]

```

El tamaño de los paquetes es igual para ambas fuentes de tráfico:

```

$cbr0 set packetSize_500
$cbr1 set packetSize_500

```

Una frecuencia de 200 paquetes por segundo equivale a un intervalo de 0.005 segundos:

```

$cbr0 set interval_0.005
$cbr0 attach-agent $udp0
$cbr1 set interval_0.005
$cbr1 attach-agent $udp1

```

El sumidero irá situado en el nodo "n3":

```
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

Conectamos la fuente de tráfico con el "traffic sink":

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

Planificación. Las dos fuentes empezarán y acabarán en el mismo instante:

```
$ns at 0.5 "$cbr1 start"
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

Llamamos al procedimiento finish, una vez terminada la simulación:

```
$ns at 5.0 "finish"
```

Ejecución de la simulación:

```
$ns run
```

2.2. Funcionamiento

En este caso no hay un solo nodo fuente, de modo que al intentar transmitir los dos a la vez, y a la misma velocidad, y dado que el ancho de banda es igual en los tres enlaces, se origina *congestión* en el enlace que va desde el router al nodo final.

Si el *buffer* del router fuese capaz de almacenar toda la información (es decir, si la cola tuviese un tamaño *infinito*), no habría problema para mandar la información, aunque más lentamente. Este es el caso en el que aún no se ha llenado la cola (Figura 3).

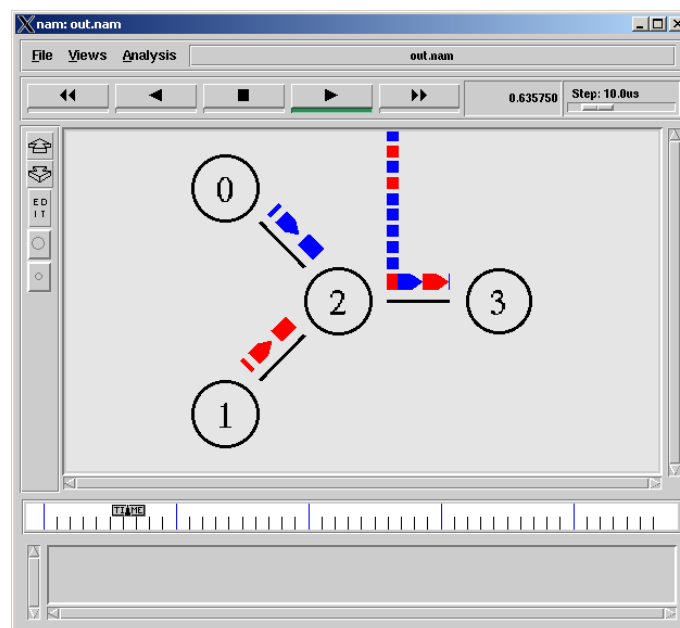


Figura 3: Cuando la cola del *router* no está llena envía bien los paquetes

Sin embargo en la realidad la cola tiene un límite que si se excede irremediablemente hay que desechar paquetes, lo cual provoca un error en la comunicación (Figura 4)

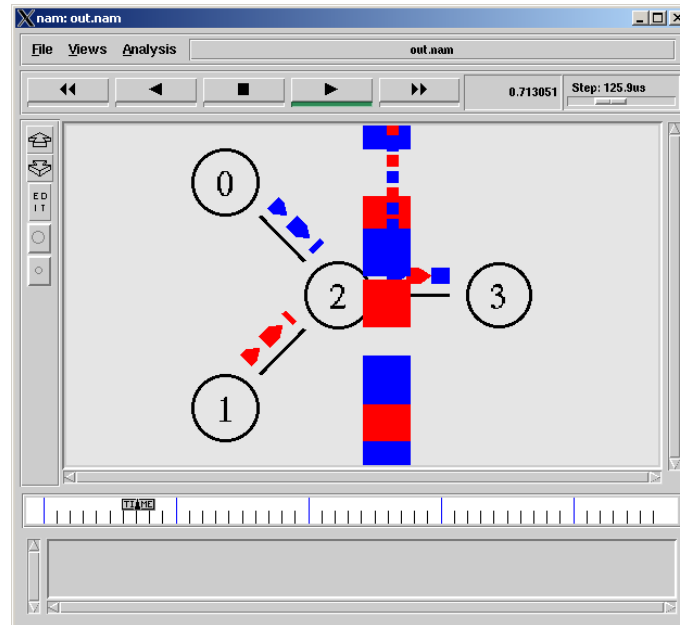


Figura 4: Cuando la cola del *router* se llena, se pierden paquetes

En este ejemplo sí se da la circunstancia de que finaliza el tiempo de simulación en el instante $t = 5s$, pero aún no ha terminado la transmisión de los paquetes. Esto ocurre porque si bien los nodos fuente no están emitiendo, la cola del router aún no se ha vaciado, y tarda más de 0,5s en hacerlo.

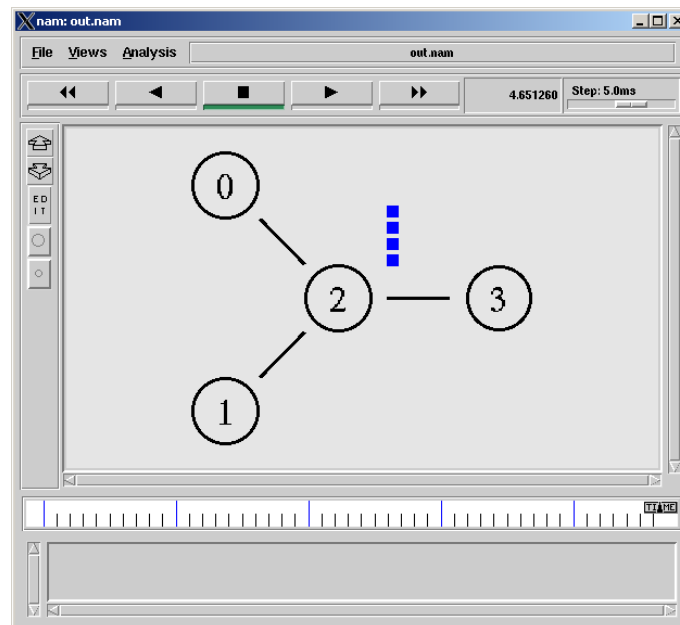


Figura 5: Al finalizar la simulación aún no se ha vaciado la cola

3. Ejemplo 3

3.1. Descripción del programa. Código fuente

En este último ejemplo crearemos una red *dinámica*, donde el encaminamiento se ajusta a la caída de un enlace. En concreto crearemos una red de topología *circular*, con siete nodos y seis enlaces, en la que cada nodo va conectado con siguiente, y el último con el primero. Aprovecharemos el potencial de un lenguaje de programación interpretado como TCL que nos permitirá crear un vector de variables mediante el uso de un bucle.

Nota: El nodo origen es el número 0 y el destino es el número 3.

```
# Creamos el objeto simulador:
set ns [new Simulator]

# Habilitamos el "encaminamiento dinámico":
$ns rtproto DV

# Abrimos el fichero donde volcaremos el resultado para luego visualizarlo con el nam.
Definimos también el procedimiento "finish" para volcar el resultado de la simulación y
ejecutar el nam:
set nf [open out.nam w]
$ns namtrace-all $nf

proc finish {} {
    global ns nf f0
    $ns flush-trace
    close $nf
    exec nam out.nam &
    exit 0
}

# Necesitamos crear 7 nodos. Como TCL es un lenguaje interpretado podemos utilizar
un bucle para definir un vector de nodos:
for {set i 0} {$i<7} {incr i} {
    set n($i) [$ns node]
}

# Creamos los enlaces. Cada nodo irá conectado al siguiente, y el último con el primero,
es lo que se conoce como "topología circular":
for {set i 0} {$i<7} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}

# Creamos un Agente UDP y lo asociamos al nodo "n0":
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# Creamos la fuente de tráfico de tipo CBR:
set cbr0 [new Application/Traffic/CBR]
```

```

# Especificamos el tamaño tamaño (en bytes) de cada paquete:
$cbr0 set packetSize_500

# Una frecuencia de 200 paquetes por segundo equivale a un intervalo de 0.005 segundos:
$cbr0 set interval_0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

# Conectamos la fuente de tráfico con el "traffic sink":
$ns connect $udp0 $null0

# Planificación. Veremos que pasa cuando uno de los enlaces se rompe. Los paquetes
deberán utilizar una ruta alternativa:
$ns at 0.5 "$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"

# Llamamos al procedimiento finish, una vez terminada la simulación:
$ns at 5.0 "finish"

# Ejecución de la simulación:
$ns run

```

3.2. Funcionamiento

Al comienzo de la simulación, los nodos se envían entre sí unos paquetes de información de la red, con los que determinan cuál es la ruta más corta para transmitir desde el nodo origen (0) hasta el nodo destino (3). De manera que cuando comienza la emisión de paquetes de datos, éstos siguen la ruta que se ve en la Figura 6.

Sin embargo, cuando en el instante $t = 1s$ se cae el enlace que une los nodos 1 y 2, se ocasiona una pérdida de paquetes, así que se vuelve a calcular la ruta óptima. En este caso sólo existe una posibilidad (Figura 7). A partir de entonces se envían los paquetes por esta nueva ruta.

A pesar de haber podido solucionar el problema ante la caída de un enlace, la red sigue comprobando en todo momento el estado de todos los enlaces, para optimizar las rutas. Así pues, cuando en el instante $t = 2s$ se restablece el nodo que se había caído, los paquetes vuelven a enviarse a través del camino original. En la Figura 8 se observa un momento del recorrido en el que se restablece la ruta original, pero aún quedan paquetes de datos viajando por la ruta antigua.

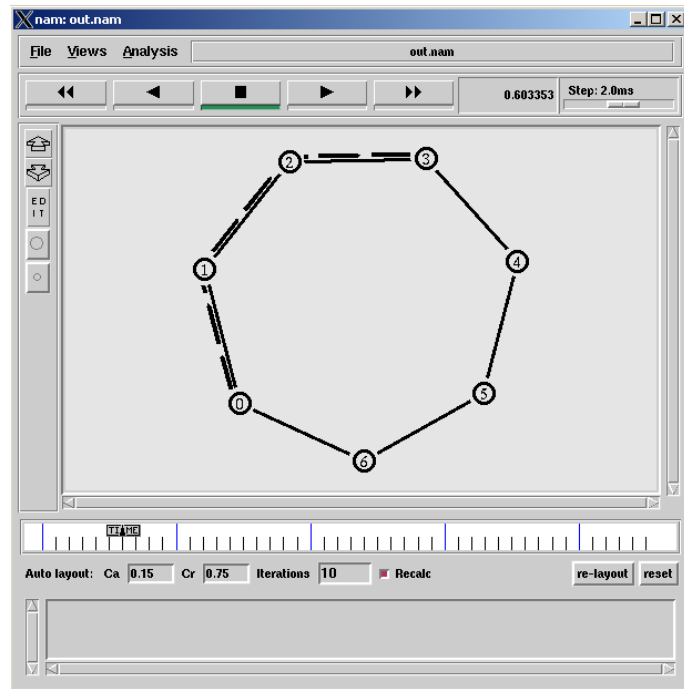


Figura 6: Al finalizar la simulación aún no se ha vaciado la cola

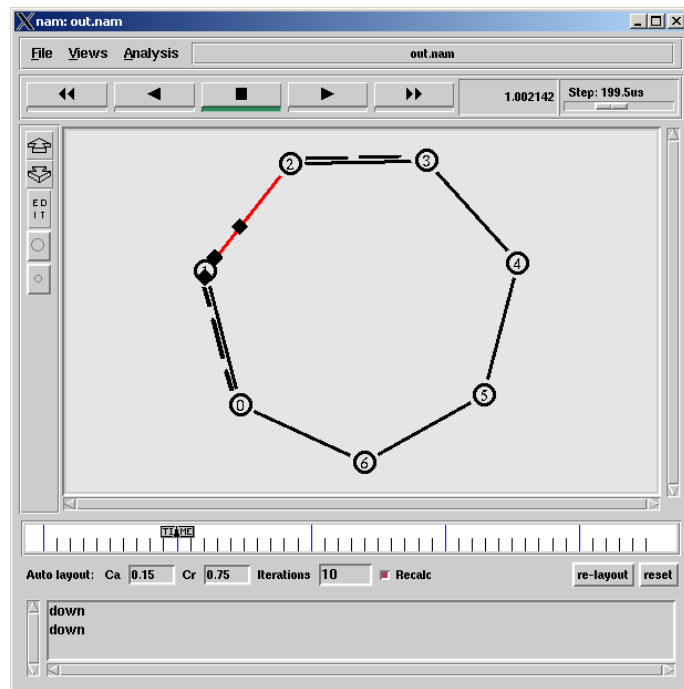


Figura 7: Al finalizar la simulación aún no se ha vaciado la cola

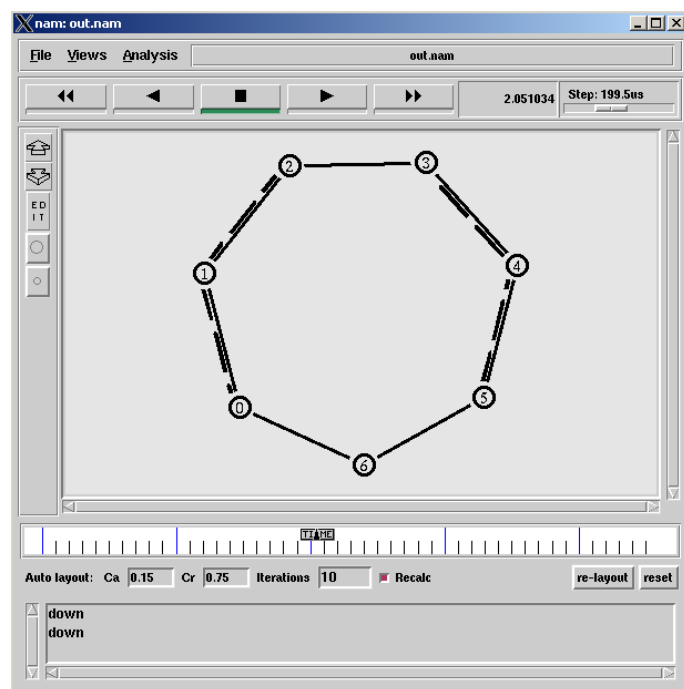


Figura 8: Al finalizar la simulación aún no se ha vaciado la cola