

MANUAL DE USUARIO

1. Introducción

NS-2 o *Network Simulator version 2* es un simulador de tiempo discreto cuya elaboración se inició en 1989 con el desarrollo de *REAL Network Simulator*. Probablemente una de las principales razones que explican su éxito es el hecho de que la distribución posee licencia GPL, condición que impulsa el desarrollo libre del mismo. Inicialmente, *NS-2* fue creado para redes fijas, sin embargo, el grupo *Monarch* de *CMU* desarrolló una ampliación para el análisis de redes inalámbricas donde se incluyen las principales propuestas de redes *ad hoc*, y también con redes WLAN (*Wireless Local Area Networks*).

El *Network Simulator* se apoya en dos lenguajes de programación para su correcto funcionamiento. Por un lado, el usuario introduce las especificaciones del escenario que desea analizar (topología) a través del lenguaje interprete *OTcl*, versión orientada a objetos. Por otro lado, la implementación de los protocolos se encuentra en lenguaje compilador C++. Como resultado de la simulación, se pueden obtener datos matemáticos para un estudio posterior, o bien, trazas específicas para visualizarlas por medio de la herramienta *NAM* y *XGRAPH* del *NS-2*.

2. Implementación de Protocolos en NS-2

En el momento de trabajar en la implementación de un protocolo en el *NS-2*, es necesario seguir los siguientes pasos:

- Implementación del protocolo a analizar mediante la incorporación de código en C++ y *Otc* dentro del núcleo de *NS-2*.
- Descripción de la simulación mediante *OTcl*. En esta sección, el usuario detalla el escenario a simular, es decir, crea la topología que necesita el protocolo. Para ello, debe manejar su conocimiento en medios de propagación, en protocolos a distintos niveles, en limitaciones de cableados, etc.
- Ejecución de la simulación. La ejecución de una simulación sencilla en *NS-2* se limita a hacer funcionar el comando *ns* seguido del fichero donde se encuentra la descripción de la simulación. Si el objetivo es analizar el impacto de la configuración de ciertos parámetros dentro de los protocolos, el usuario debe desarrollar un *script* para automatizar la simulación de diversos escenarios variando los parámetros a analizar.
- Análisis de Resultados. El *Network Simulator* cuenta con una herramienta denominada *NAM* que permite la visualización del comportamiento de los terminales de la red. Con ello, el usuario

puede apreciar como el paquete de datos se va encaminando a través de los distintos dispositivos, como se van perdiendo los paquetes en redes cableadas por un exceso de tráfico, etc. También es posible extraer métricas cuantitativas como resultados de las simulaciones y manejar la herramienta de *Xgraph* que permite trazar estos resultados en una forma más fácil de analizar.

3. Procedimiento para hacer modificaciones en *NS-2*.

Al momento de hacer modificaciones en el simulador de redes *NS-2* se deben tener en cuenta los siguientes pasos:

- ❖ Encontrar el respectivo archivo o fichero a modificar dentro de la carpeta del simulador. Se debe tener en cuenta que estos son ficheros en lenguaje C++, y es la parte interna del simulador. En caso de querer modificar la parte externa del simulador, es decir, cambiar la topología o los parámetros de simulación, las modificaciones se harían en los *scripts* de *Tcl*. En caso de modificar la estructura interna del *NS-2*, los ficheros se encuentran en */usuario/ns-allinone-2.31/ns-2.31/*. Al modificar estos archivos es necesario crear nuevas carpetas con los nuevos cambios, para así no cambiar la estructura del *NS-2* como tal, sino agregar módulos.
- ❖ Si se agregan nuevos ficheros al *ns*, es indispensable incluirlos en el fichero de *makefile.in*, para poder incluir esos nuevos programas en la compilación general del simulador. Esta carpeta se encuentra en el fichero de *ns-allinone-2.31*.
- ❖ Después de crear los nuevos ficheros e incluirlos en la carpeta de *makefile.in*, se debe compilar todos los programas del simulador para así comprobar si las nuevas modificaciones no tienen ningún error. Para realizar esta compilación correctamente se debe digitar los siguientes comandos en un terminal:
 - *./configure*: este comando nos sirve para reconfigurar los ficheros recién incluidos en el *makefile.in*.
 - *make depend*: Este comando añade los nuevos cambios hechos en los archivos del simulador, para luego realizar el comando *make*.
 - *make*: este comando compila todos los ficheros existentes dentro del simulador, y muestra si hay errores en alguno de ellos, mostrando el fichero y la línea, para una mayor facilidad al corregir los errores.

Con la ayuda de los anteriores pasos, es más fácil modificar los archivos que trae el *NS-2* por defecto, y crear nuestros propios programas en base a ellos. Este simulador se caracteriza en que el usuario puede

crear sus propias herramientas o mejorar las existentes solo con el aprendizaje de los lenguajes necesarios (C++ y *Otcl*)

4. Manual de Instalación del Simulador NS-2

Para instalar el *Network Simulator* bajo el sistema operativo de *Linux* se deben seguir los siguientes pasos:

- ✓ Primero que todo se descarga el paquete “todo en uno”, el cual trae todos los paquetes del simulador en uno. Para descargarlo se puede acceder a esta página: <http://www.isi.edu/nsnam/dist/> y escoger la versión más conveniente para el usuario. Se debe colocar el archivo descargado en */home/nombre-usuario/*, donde nombre-usuario es la cuenta personal en el computador.
- ✓ Para descomprimir el archivo se habilita un terminal y se aplica el comando *gunzip ns-allinone-2.31.tar.gz*, dependiendo de la versión que se haya descargado.
- ✓ Se reconstruye el sistema de archivos, aplicando el comando *tar -xvf ns-allinone-2.31.tar*.
- ✓ Pasar a modo superusuario, aplicando el comando *su* y el *password* correspondiente.
- ✓ Modificar el *PATH*, lo que permite encontrar los archivos necesarios. Para ello se edita el archivo */home/nombre-usuario/.bashrc/* insertando las siguientes líneas (respetar mayúsculas-minúsculas):

```
PATH=$PATH:/usr/X11R6/bin:/usr/X11R6/lib:/usr/X11R6/lib/X11:/usr/X11R6/include/X11:/home/nombre-usuario/ns-allinone-2.30/bin:/home/nombreusuario/ns-allinone-2.30/tcl8.4.13/unix:/home/nombre-usuario/ns-allinone-2.30/tk8.4.13/unix:/home/nombre-usuario/ns-allinone-2.30/xgraph-12.1:/home/nombre-usuario/ns-allinone-2.30/nam-1.12
export LD_LIBRARY_PATH=/home/nombre-usuario/ns-allinone-2.30/otcl-1.12:/home/nombre-usuario/ns-allinone-2.30/lib
export TCL_LIBRARY=/home/nombre-usuario/ns-allinone-2.30/tcl8.4.13/library
```

- ✓ Activar el *PATH*, aplicando el comando *source .bashrc*.
- ✓ Ir al directorio de instalación del NS-2 con el siguiente comando, *cd ns-allinone-2.31*.
- ✓ Instalar NS-2, aplicando el comando *./install*

- ✓ Una vez finalizada la instalación, ir al directorio `/home/nombre-usuario/ns-allinone-2.31/ns-2.31/`, y correr el programa de validación, aplicando el comando `./validate`.
- ✓ Seguir y ejecutar los ejemplos indicados en el tutorial “ns by example” (<http://nile.wpi.edu/NS/>).

5. Instalación del Planificador WFQ

Para realizar la instalación del Planificador WFQ se requiere seguir los siguientes pasos:

- Se descarga el instalador de Internet (<http://www.cc.jyu.fi/~sayenko/src/wfq-1.2.4.tar.gz>).
- Se extraen los archivos en la carpeta deseada.
- Se abre un terminal y se accede a la carpeta donde se encuentran los archivos extraídos y se digita el siguiente comando:

```
./patch.sh /home/nombre-usuario/ns-allinone-2.31/ns-2.31
```

La ruta anterior varía dependiendo de donde se hayan extraído los archivos de la herramienta.

- Finalmente se digitan los siguientes comandos:

```
./configure  
make depend  
make
```

Al realizar los anteriores comandos en el terminal, se obtiene un error que tiene la herramienta por defecto, el cual es necesario corregirlo para poder utilizar la herramienta correctamente. En el terminal se obtiene el archivo donde se encuentra el error y la línea de código:

```
wfqclassifier.h:31  
  
31 wfqClass: : wfqClass();
```

La parte resaltada en negrita se borra de la línea y se vuelve a compilar el simulador con los 3 comandos mencionados anteriormente.

De esta forma queda instalada correctamente la herramienta del Planificador WFQ.

6. Instalación de la Herramienta Creada

El proceso de instalación es bastante sencillo. A continuación se describen cada uno de estos pasos:

- La herramienta tiene en total 11 nuevos archivos, los cuales deben ser ubicados en cada una de sus respectivas carpetas del simulador *NS-2*. Es por esto que se crea la Tabla 1, la cual muestra cada uno de los archivos, la ubicación donde deben ser añadidos y la respectiva descripción de cada uno.

Número	Nombre del Archivo	Ruta	Descripción del Archivo
1	classifier-IntServ.h	home/usuario/ns-allinone-2.31/ns-2.31/classifier	Declaración de variables y métodos en C++ del Clasificador IntServ
2	classifier-IntServ.cc	home/usuario/ns-allinone-2.31/ns-2.31/classifier	Programa principal en C++ del Clasificador IntServ
3	classifier-IntServ6.h	home/usuario/ns-allinone-2.31/ns-2.31/classifier	Declaración de variables y métodos en C++ del Clasificador IntServ6
4	classifier-IntServ6.cc	home/usuario/ns-allinone-2.31/ns-2.31/classifier	Programa principal en C++ del Clasificador IntServ6
5	IntServ6.h	home/usuario/ns-allinone-2.31/ns-2.31/classifier	Declaración de variables y métodos del módulo de Host Origen en IntServ6
6	IntServ6.cc	home/usuario/ns-allinone-2.31/ns-2.31/classifier	Programa Principal del cálculo del Número Hash en el Host Origen (IntServ6)
7	ip.h	home/usuario/ns-allinone-2.31/ns-2.31/common	Configuración de la cabecera IP del paquete
8	ip.cc	home/usuario/ns-allinone-2.31/ns-2.31/common	Configuración de la cabecera IP del paquete
9	config.h	home/usuario/ns-allinone-2.31/ns-2.31	Configuración de estructuras utilizadas en los dos clasificadores
10	makefile.in	home/usuario/ns-allinone-2.31/ns-2.31	Adición de los nuevos objetos creados
11	ns-default.tcl	home/usuario/ns-allinone-2.31/ns-2.31/tcl/lib	Adición de nuevas variables de los Clasificadores creados.

Tabla 1. Archivos necesarios para la instalación de la Herramienta

- El segundo paso al finalizar la ubicación de los archivos, es la activación de los nuevos objetos ingresados al *makefile.in*. Cabe resaltar que si el usuario no ingresa los nuevos archivos a este archivo, no puede hacer uso de la nueva herramienta.

La activación se realiza mediante los siguientes comandos:

- a) El usuario debe abrir un terminal de *linux* y ubicarse en la carpeta donde se encuentra instalado el simulador *NS-2*. Esto se realiza mediante el siguiente comando:

```
cd nombre-usuario/ns-allinone-2.31/ns-2.31
```

- b) Al estar ubicado en la carpeta principal del simulador, se activa el *makefile* con el siguiente comando:

```
./configure
```

- c) Y finalmente se digitan los comandos de compilación del simulador. El primer comando para adjuntar los nuevos cambios a cualquiera de los archivos del simulador, y el segundo comando para compilar todos los archivos del simulador. Los comandos son los siguientes:

```
make depend  
make
```

De esta forma ya queda instalada la nueva herramienta para luego crear el *script* de *Tcl* con el que se desea simular. El *script* de *Tcl* puede ser diseñado como lo desee el usuario, pero más adelante se hace un breve ejemplo de cómo deben añadirse los nuevos Clasificadores a la topología de la simulación.

7. Manejo de la Herramienta

Teniendo la herramienta instalada en el simulador *NS-2*, el usuario debe proceder a crear los *scripts* de *Tcl* ingresando los nuevos módulos de Clasificadores en los nodos de la topología. Para esto se debe crear un programa principal, donde se encuentren configurados los parámetros de la simulación en general (creación de archivos para resultados, generadores de tráfico, configuración del Planificador *WFQ*, tiempos de inicio y finalización de la simulación, entre otros) y un subprograma que es llamado por el programa principal y que configura todo lo necesario en la topología. Cabe resaltar que para cualquiera de los dos Clasificadores se puede crear un solo programa principal, y dos topologías diferentes, lo que equivale a dos subprogramas de topología con la configuración de cada uno de sus clasificadores.

7.1. Creación de la Topología para *IntServ*

Para una mayor comprensión se crea una topología ejemplo en la cual se describen cada uno de los módulos ingresados. (Ver Figura 1)

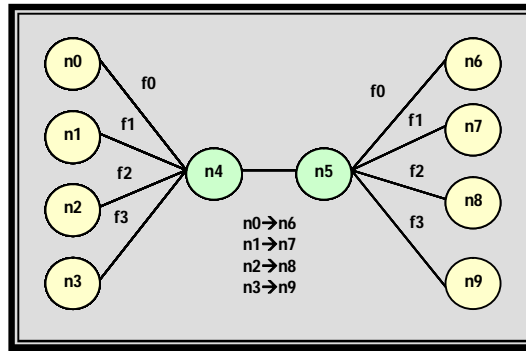


Figura 1. Topología Ejemplo para IntServ

La Programación en C++ de los nuevos objetos creados en el Simulador NS-2, se realiza dependiendo del tipo de topología que se desea usar. Es por esto que los nuevos módulos solo se deben utilizar con este mismo tipo de topología. En caso de querer cambiarla, se deben modificar los algoritmos de C++ se encuentran ubicados en la Tabla 9 del presente Proyecto.

De la anterior topología el nodo *n4* equivale al Nuevo Clasificador *IntServ* ingresado para esta simulación. El resto de nodos son los que trae el simulador por defecto sin ninguna modificación. El enlace entre los nodos *n4* y *n5* (*r* y *d*) equivale al Planificador *WFQ*.

El subprograma *topologia.tcl* invocado por el programa principal *cbr.tcl*, es utilizado para crear la topología de la simulación y configurar todos sus parámetros. El proceso comienza con la declaración del nombre del proceso y las variables globales utilizadas en el subprograma. A continuación aparece cada uno de los comandos utilizados para esto:

```

proc crear_topo {} {
  global ns n r d cl a b c

```

El segundo paso consiste en la inicialización de las variables globales utilizadas en el subprograma:

```

set a 1
set b 1
set c 0

```

Los valores de las anteriores variables dependen del Número de Flujos utilizados en la simulación, y el número de nodos de la topología.

La topología utilizada en esta simulación presenta dos nodos intermedios que siempre son constantes cuando se varían la cantidad de nodos y flujos del programa. Estos nodos fueron llamados *r* y *d*, donde *r* representa el nodo *IntServ*, y *d* representa un nodo normal del simulador. Estos dos nodos están enlazados por el Planificador *WFQ*, que es creado más adelante. A continuación se muestran las instrucciones que

se utilizan para crear estos dos nodos intermedios y los nodos de origen y destino de la simulación:

```
// Creación de nodo r para IntServ y d como nodo IP normal
set r [$ns node]
set d [$ns node]
Creación de nodos de Origen y Destino
for {set i 0} {$i <= $a} {incr i} {
    set n($i) [$ns node]
}
```

El siguiente paso es el más importante en la configuración de la topología, pues es el acople del modulo de *IntServ* creado con el nodo *r*. Primero que todo se inicializa la variable *flujo_*, necesaria en el código de C++ (*ns-allinone-2.31/ns-2.31/tcl/lib/ns-default.tcl*). Luego se crea el nuevo Clasificador *IntServ* y se inserta al nodo *r* mediante el comando *insert-entry*. Cabe resaltar que este nuevo Clasificador hace parte del modulo de enrutamiento de tipo Base del Simulador *NS-2* y es ingresado en el slot 1. Finalmente se ingresa el comando de Control de Admisión del Clasificador *IntServ* con sus valores iniciales para llenar la Tabla Hash y la Tabla de Colisiones. A continuación se presenta el código del anterior procedimiento:

```
Classifier/IntServ set flujo_ $b // Fija Parámetro de Flujo
set cls [new Classifier/IntServ] // Creación del Clasificador
IntServ
$r insert-entry RtModule/Base $cls "1" // Inserta Clasificador
IntServ en el nodo r
$cls add-num 0 $b // Ejecuta Control de Admisión e inicia Tabla
Hash y Tabla de Colisiones
```

Luego de configurar los nodos, se crean los enlaces con sus respectivos parámetros de retardo, ancho de banda y tipo de cola. En esta parte se hace uso de los bucles *for*, debido a que son demasiados y se optimizan más con estos ciclos. A continuación se muestra el código:

```
// Crea enlaces entre los nodos Origen y el nodo r
for {set i 0} {$i <= $c} {incr i} {
    $ns duplex-link $n($i) $r 5Mb 1ms DropTail
}
// Crea enlaces entre el nodo d y los nodos de Destino
for {set i $b} {$i <= $a} {incr i} {
    $ns duplex-link $n($i) $d 5Mb 1ms DropTail
}
```

Estos enlaces son creados entre los Host Origen y el nodo *r*, y los Host Destino y el nodo *d*. Además, con las siguientes instrucciones también se crea el enlace *WFQ* entre los nodos *r* y *d*, configurando también sus respectivos parámetros:

```
$ns simplex-link $r $d 2Mb 2ms WFQ
```



```
$ns simplex-link $d $r 2Mb 2ms DropTail
```

La configuración del enlace *WFQ* se realiza mediante dos enlaces simplex, debido a que las reservas solo se hacen en un solo sentido.

Por último se realiza la creación de un objeto tipo Clasificador que necesita el Planificador *WFQ* para clasificar los paquetes; este es instalado en el enlace entre *r* y *d*. A continuación se muestran las últimas líneas utilizadas para la configuración de la topología:

```
set cl [new WFQAggregClassifier]
$ns wfqclassifier-install $r $d $cl
}
```

7.2. Creación de la Topología para *IntServ6*

Para una mayor comprensión se crea una topología ejemplo en la cual se describen cada uno de los módulos ingresados. (Ver Figura 2)

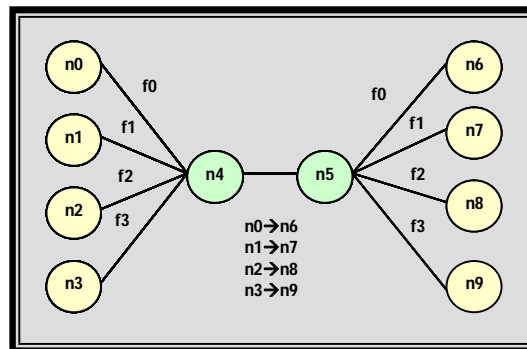


Figura 2. Topología Ejemplo para *IntServ6*

De la anterior topología el nodo *n4* equivale al Nuevo Clasificador *IntServ6* ingresado para esta simulación. Los nodos *n0* al nodo *n3*, equivalen a los módulos que calculan el Número Hash del Flujo y lo ingresan a sus paquetes; el nombre utilizado para este modulo es *HostOrigen* y se desde *Tcl* se invoca como un *Classifier/Start*. El resto de nodos son los que trae el simulador por defecto sin ninguna modificación. El enlace entre los nodos *n4* y *n5* (*r* y *d*) equivale al Planificador *WFQ*.

El subprograma *topologia6.tcl* tiene como objetivo crear la topología para una red *IntServ6* y configurar los parámetros de la misma. El primer paso de este *script* es la declaración del proceso que es llamado en el programa principal y de las variables globales utilizadas tanto en el programa principal como en el subprograma. Las líneas utilizadas son las siguientes:

```
proc crear_topo6 {} {
global ns n r d cl a b c
```

En las siguientes líneas se hace la inicialización de algunas variables utilizadas en este proceso, como se muestra a continuación:

```
set    a    1
set    b    1
set    c    0
```

Cabe resaltar que la inicialización de estas variables depende del Número de Flujos utilizados y del Número de nodos de la topología, por lo que estos valores son los mismos para *IntServ* cuando solo hay un flujo.

A continuación se crean los nodos intermedios de la topología llamados *r* y *d*, los cuales son constantes en el proceso de aumento de flujos y nodos para el análisis del comportamiento de la red. Cabe resaltar que el enlace entre estos dos nodos intermedios es donde se encuentra ubicado el Planificador *WFQ*. Los nodos de origen y los nodos de destino de la red son creados mediante un bucle *for*, debido a que estos varían dependiendo del número de flujos para los que se ejecute la simulación. Las líneas de código utilizadas para crear estos nodos son las siguientes:

```
//Creación de nodos r y d
set r [$ns node]
set d [$ns node]
// Creación de nodos de origen y destino
for {set i 0} {$i <= $a} {incr i} {
    set n($i) [$ns node]
}
```

El siguiente paso a seguir es la configuración de cada uno de los nodos. Los primeros nodos en configurarse son los Host Origen, los cuales deben calcular el Número Hash de cada flujo e ingresarlo a la cabecera de cada paquete. Para esto se creó un nuevo módulo *HostOrigen*, el cual es invocado como un clasificador de tipo *Start*. Este nuevo clasificador es ingresado a todos los nodos Origen mediante el comando *insert-entry* en el slot 1. Las siguientes líneas describen este proceso:

```
for {set i 0} {$i <= $a} {incr i} {
    set cls($i) [new Classifier/Start]
    $n($i) insert-entry RtModule/Base $cls($i) "1"
}
```

Después de configurar los nodos origen, se accede a crear un Clasificador de tipo *IntServ6* y acoplarlo al nodo *r*, que hace las veces de Router *IntServ6*. Además se inicializan algunas variables necesarias dentro del Clasificador *IntServ6* y se realiza el Control de Admisión y establecimiento de Tablas de Reservas antes de la transmisión de los paquetes. A continuación se describen las instrucciones que realizan este procedimiento:

```

Classifier/IntServ6 set flujoint_ $b //Fija el Número de Flujos del
Clasificador
Classifier/IntServ6 set destinoint_ $b // Fija el nodo de destino
inicial (nodo n6 de la Figura x)
set cl [new Classifier/IntServ6] // Se crea el Clasificador con
nombre cl
$r insert-entry RtModule/Base $cl "2" // Se inserta el Clasificador
en el nodo r
$cl adc-num 0 $b //Ejecuta el Control de Admisión y establece las
Tablas de Reservas

```

Luego de configurar los nodos, se crean los enlaces con sus respectivos parámetros de retardo, ancho de banda y tipo de cola. En esta parte se hace uso de los bucles *for*, debido a que son demasiados y se optimiza el código con estos ciclos. A continuación se muestra el código:

```

// Creación de enlaces entre el nodo r y los nodos de Origen
for {set i 0} {$i <= $c} {incr i} {
    $ns duplex-link $n($i) $r 5Mb 1ms DropTail
} // Creación de enlaces entre el nodo d y los nodos de Destino
for {set i $b} {$i <= $a} {incr i} {
    $ns duplex-link $n($i) $d 5Mb 1ms DropTail
}

```

Estos enlaces son creados entre los Host Origen y el nodo *r*, y los Host Destino y el nodo *d*. Además también se crea el enlace *WFQ* entre los nodos *r* y *d*, configurando también sus respectivos parámetros:

```

//Creación de enlace WFQ en sentido r→d
$ns simplex-link $r $d 2Mb 2ms WFQ
// Creación enlace DropTail en sentido d→r
$ns simplex-link $d $r 2Mb 2ms DropTail

```

Por ultimo se realiza la creación de un objeto tipo Clasificador que necesita el Planificador *WFQ* para clasificar los paquetes y es instalado en el enlace creado en el anterior procedimiento. A continuación se muestran las últimas líneas utilizadas para la configuración de la topología:

```

set cl [new WFQAggregClassifier]
$ns wfqclassifier-install $r $d $cl
}

```

7.3. Creación del Programa Principal

A continuación se realiza una explicación minuciosa de cómo se debe crear el *script* principal:

El *script* de *Tcl* puede ser creado en cualquier editor de texto, dándole el nombre deseado por el usuario y guardándolo con extensión *.tcl*. En este caso se llama *cbr.tcl* para *IntServ* y *cbr6* para *IntServ6*.

Primero, debe declararse el subprograma que crea la topología y configura los parámetros de la misma. Esta instrucción varía según el subprograma de topología que desea utilizar el usuario. Esto se hace con la siguiente instrucción:

```
source topologia.tcl           // source topologia6.tcl
```

En este caso para nuestra simulación, el subprograma que crea la topología fue llamado *topologia.tcl*; y el comando *source* declara que este archivo puede ser llamado en cualquier momento del programa para crear la topología y configurar cada uno de los routers y enlaces de la misma. Además mediante este subprograma se fijan otros parámetros de la topología como son: tiempos, retardos, anchos de banda, tipos de colas, etc

A continuación se declaran las variables globales a utilizar en el programa y en los subprogramas. Esto se hace mediante el comando *global* de la siguiente manera:

```
global ns n r d cl a b c reg delaywfq
```

El siguiente paso es fijar los valores iniciales a las variables utilizadas dentro del programa principal mediante el comando *set*. De esta forma era más fácil el cambio al iniciar el código en caso de querer cambiar los parámetros de simulación. El código utilizado es el siguiente:

```
set a 1
set b 1
set c 0
```

En las anteriores líneas, la variable *a* representa el número de nodos necesarios en la topología. Para simular esta red bajo diferentes parámetros fue necesario cambiar esta variable aumentándola exponencialmente. Esta variable también representa el último nodo de destino de la topología.

La variable *b* representa el número de flujos que tiene la red, y el primer nodo de destino. Y por último la variable *c* equivale al último nodo de origen.

Las siguientes líneas son las utilizadas para borrar todas las cabeceras que tiene el simulador por defecto, y activar sólo las necesarias para nuestra simulación. En este caso sólo se añaden las cabeceras *IP* y *TCP*. A continuación se muestra las instrucciones necesarias para este proceso:

```
remove-all-packet-headers
add-packet-header IP TCP
```

Al finalizar estos pasos, es necesario crear un Objeto Simulador, mediante el siguiente comando:

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

Luego, se abre un archivo de escritura que se utiliza más adelante para trazar los datos en el *.nam*. Esto se realiza mediante las siguientes instrucciones:

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

La primera línea abre el archivo de escritura "*out.nam*", y se le da el nombre de "*nam*" con el comando *set*. Y mediante la segunda línea se le dice al Objeto Simulador creado anteriormente que almacene todos los datos de la simulación para luego ser utilizados en la herramienta *nam*.

Las siguientes líneas son utilizadas para crear un archivo de escritura utilizado para monitorear el ancho de banda del enlace *WFQ*. Este proceso se realiza mediante la siguiente instrucción:

```
set reg [ open reg.tr w]
```

A continuación se crean los archivos de salida donde se almacenan los datos de la simulación, para más adelante ser graficados mediante la herramienta *Xgraph* en forma individual. En estos archivos se guarda el ancho de banda de cada uno de los flujos y se crean mediante las siguientes instrucciones:

```
for {set i 0} {$i <= $b} {incr i} {
    set f($i) [open fl($i).tr w]
    $ns trace-all $f($i)
}
```

En las anteriores instrucciones se utiliza un bucle *for* con la finalidad de optimizar el código y ahorrar memoria de simulación. Esto se realiza debido a que fue necesario simular la red para un número grande de flujos, y por lo tanto representaba un número grande de líneas de código. La primera línea crea los archivos de escritura y la segunda se encarga de trazar las gráficas.

El siguiente paso es crear un proceso *record* con el objetivo de leer el número de bytes recibidos en los sumideros de tráfico, calcular los anchos de banda de cada flujo y almacenarlos en los archivos de salida junto con el tiempo actual, para más adelante ser graficados. Luego se resetean los valores de *bytes_* de cada sumidero de tráfico. Cabe resaltar que un sumidero (*sink*) es un objeto el cual se encuentra ubicado en cada uno de los nodos de destino y se encarga de recibir el tráfico de paquetes y almacenar los anchos de banda de cada flujo para graficarlos mediante la

herramienta de *Xgraph*. A continuación se detallan las líneas de código de este proceso:

```
proc record {} {
    global sink ns f nam c b a reg
    set time 0.5
    for {set i 0} {$i <= $c} {incr i} {
        set bw($i) [$sink([expr $i+$b]) set bytes_ ]
    }
    set now [$ns now]
    for {set i 0} {$i <= $c} {incr i} {
        puts $f($i) "$now [expr $bw($i)/$time*8]"
    }
    for {set i 0} {$i <= $c} {incr i} {
        if {$now == 4.5} {
            puts $reg "[expr $bw($i)/$time*8]"
        } else {}
    }
    for {set i $b} {$i <= $a} {incr i} {
        $sink($i) set bytes_ 0
    }
    $ns at [expr $now+$time] "record"
}
```

Las dos primeras líneas del anterior proceso son la declaración de las variables globales utilizadas dentro del mismo y la fijación de la variable *time* en 0.5 que es necesaria para calcular los anchos de banda y almacenar los resultados de la simulación en los archivos creados para esta tarea. Las siguientes líneas se realizan mediante unos bucles *for*, que ayudan a optimizar el código y ahorrar memoria. El primer bucle *for*, se utiliza para analizar cuantos *bytes* han sido recibidos por los sumideros de tráfico en cada uno de los nodos de destino. En el segundo bucle se calcula el ancho de banda y se almacenan en los archivos creados al iniciar el programa. El tercer bucle representa las instrucciones que se utilizan para monitorear cada 4.5 segundos el ancho de banda en el enlace *WFQ*. Y finalmente el cuarto bucle es necesario para resetear los bytes en los sumideros de tráfico. Y por último se presenta una línea de código que sirve para re-planificar el proceso que se acaba de describir.

El siguiente proceso en el *script* es el *finish*, en el cual se cierran los archivos anteriormente creados y llenados, para finalmente ejecutar el *Xgraph*. Además se configura el tamaño de la grafica resultante y se realizan algunas sumatorias de resultados almacenados, indispensables más adelante para graficar los retardos de los paquetes. A continuación se muestran las líneas de código de este proceso:

```
proc finish {} {
    global ns f nam c fl reg delaywfq
    $ns flush-trace
    for {set i 0} {$i <= $c} {incr i} {
```

```

        close $f($i)
    }
    close $nam
    close $reg

    puts stdout "Retardo promedio de la cola wfq: [$delaywfq mean]"

    exec cat reg.tr | awk {{sum+=$1;print sum> "sbw1.tr"}}
    exec cat clintserv.tr | awk {{sum+=$i;print sum>
    "timeintserv1.tr"}}

    for {set i 0} {$i <= $c} {incr i} {
        exec xgraph fl($i).tr -geometry 800x400 -x Tiempo &
    }
    exec nam out.nam &
    exit 0
}

```

En las anteriores líneas resaltadas en negrita se encuentran tres comandos de *linux* llamados *exec*, *cat* y *awk*. El comando *awk* es un lenguaje de programación diseñado para procesar datos basados en texto, ya sean ficheros o flujos de datos. El comando *cat* sirve para mostrar un archivo y por último el comando *exec* realiza la ejecución de la línea. Para una mayor comprensión de estas líneas se hace una explicación detallada de cada una de ellas:

En la primera línea resaltada se crea un archivo llamado *reg.tr*, en el cual se guarda el ancho de banda de todos los flujos en un tiempo determinado, en este caso a los 4,5 segundos. Cabe resaltar que este proceso se realiza desde el "record". El comando *awk* crea una variable que se llama *sum*, en la cual se va guardando y sumando los datos del campo 1 (\$1), que esta en el archivo *reg.tr*. Finalmente se imprime (*print*) esta sumatoria en el archivo *sbw1.tr*, con el objetivo de que al final de la columna de este archivo se encuentre el ancho de banda total del enlace.

En la segunda línea resaltada, el comando *awk* crea una variable que se llama *sum*, en la cual se va guardando y sumando los datos del campo 1 (\$1), que esta en el archivo *clintserv.tr* (en este archivo se va grabando el retardo de cada paquete que pasa por el clasificador *IntServ6*) y después lo imprime (*print*) en el archivo *timeintserv1.tr*, con el fin de que al final de la columna de este archivo, este guardado la suma de todos los retardos de los paquetes. El archivo *clintserv.tr* se puede encontrar declarado en las líneas de código de C++ de los archivos *classifier-IntServ6.cc* y *classifier-IntServ6.h* ubicados en las rutas descritas en la Tabla 9.

El paso siguiente del programa principal es la creación de la topología, para esto se llama al proceso llamado *crear_topo* o *crear_topo6*, según el tipo de topología que el usuario desea implementar (*IntServ* o *IntServ6*), el cual se invoca simplemente digitando su nombre y automáticamente se llama al subprograma *topologia.tcl*. Este código se describe más adelante.

```
crear_topo      // crear_topo6
```

Las siguientes líneas son las encargadas de configurar los parámetros del Planificador *WFQ*. El comando *setqueue* le asigna una determinada cola a un flujo representado por su Identificación de Flujo. El comando *setweight* le asigna un peso a cada cola. Este peso puede oscilar desde 0.001 a 1, y la suma de todos los pesos de las colas deben sumar 1 que equivale al 100% del Ancho de Banda. Y por último el comando *setlength* le asigna el tamaño del paquete dentro del Planificador *WFQ*. A continuación se presenta la configuración de esta herramienta:

```
for {set i 0} {$i <= $c} {incr i} {  
    $cl setqueue $i [expr $i+$b]  
}  
for {set i 0} {$i <= $c} {incr i} {  
    $cl setweight [expr $i+$b] 0.01  
    $cl setlength [expr $i+$b] 5
```

Cabe resaltar que el objeto *cl* que corresponde al Planificador *WFQ*, es declarado en el subprograma *topologia.tcl*.

Luego se necesita configurar un Objeto para monitorear el comportamiento y retardos de las colas del Planificador *WFQ*. Para esto se necesita decidir que tipo de análisis se le quiere hacer a las colas del Planificador y asignarle un Objeto de muestreo para este monitoreo. Todas estas configuraciones se realizan mediante las siguientes instrucciones:

```
set monitorwfq [$ns monitor-queue $r $d stdout]  
set gpi [$monitorwfq get-pkts-integrator]  
set sam [new Samples]  
$monitorwfq set-delay-samples $sam  
set delaywfq [$monitorwfq get-delay-samples]
```

Esta herramienta es de fácil manejo de una gran utilidad para monitorear un conjunto de paquetes, bytes, paquetes entrantes, paquetes salientes y paquetes perdidos. Este también incluye un soporte para agregar estadísticas tales como tamaño promedio de cola, retardo promedio de cola, etc. Para mayor información acerca de esta herramienta se puede remitir al manual del simulador *NS-2*.

El siguiente paso es la creación de un agente *UDP* el cual es acoplado a cada uno de los Host origen y es el encargado de transportar los paquetes a través de la red, y luego se crea un agente generador de tráfico que es acoplado a su vez al agente *UDP*. El agente *CBR* tiene una transmisión de paquetes a una rata constante. Cada flujo tiene una identificación de flujo que es utilizada en el Planificador *WFQ* para entregarle a cada paquete su reserva pactada en el Control de Admisión del Clasificador *IntServ*.

```
for {set i 0} {$i <= $c} {incr i} {
```



```

        set udp($i) [new Agent/UDP]
        $ns attach-agent $n($i) $udp($i)
        $udp($i) set fid_ $i
    }
    for {set i 0} {$i <= $c} {incr i} {
        set cbr($i) [new Application/Traffic/CBR]
        $cbr($i) attach-agent $udp($i)
    }

```

Al finalizar la creación de los agentes de Tráfico, se crean los sumideros de tráfico acoplados a los nodos de destino con la finalidad de monitorear los paquetes que llegan y guardar los resultados para luego graficarlos. La creación de estos sumideros se encuentra a continuación:

```

    for {set i $b} {$i <= $a} {incr i} {
        set sink($i) [new Agent/LossMonitor]
        $ns attach-agent $n($i) $sink($i)
    }
    for {set i 0} {$i <= $c} {incr i} {
        $ns connect $udp($i) $sink([expr $i+$b])
    }

```

Finalmente se fijan los tiempos de inicio y fin de los procesos *record*, *finish* y los generadores de Tráfico:

```

$ns at 0.0 "record"

for {set i 0} {$i <= $c} {incr i} {
    $ns at 0.0 "cbr($i) start"
}

$ns at 5.0 "finish"

```

Por ultimo se comienza la simulación con el siguiente comando:

```

$ns run

```

Finalmente se corre la simulación con el comando *ns script.tcl*, siendo *script* el nombre del *script* principal que se desea ejecutar.