MODELO PARA LA OPTIMIZACIÓN DE LAS CAPAS DE TRANSPORTE EN LA ARQUITECTURA WAP 2.0

ANEXOS

Trabajo de grado presentado como requisito para optar el título de Ingeniero en Electrónica y Telecomunicaciones

Guillermo Esteban Guerrero Rosero
Alejandro Cardozo Montes

Director: Ing. Javier Hurtado

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Popayán, 2005

TABLA DE CONTENIDO

ANEXO 1

1 EXPERIMENTOS ADICIONALES DE LOS ESCENARIOS DE SIMULACIÓN

1.1 Escenario 1: análisis del incremento del tamaño de la ventana inicial de transmisión de TCP.

Estas pruebas de la modificación de la ventana inicial son modeladas utilizando NS-2. Los códigos de estas simulaciones se encuentran en el Capitulo Anexo 1 Códigos de las Simulaciones.

Algunas de las pruebas que se hacen a continuación son las de la modificación del mecanismo de *Handshake* que se hace al inicio del establecimiento de la comunicación, en la cual se envían paquetes SYN.

1.1.1 Experimento 1: Conexión FTP/TCP FULL con ventana inicial de 2 segmentos

Características Principales del Enlace	
Nodos	2
Protocolo	TCP FULL
Aplicación	FTP
Ventana Inicial	2
Tiempo de Simulación	15 segundos

Tabla 1. Características del Experimento 2

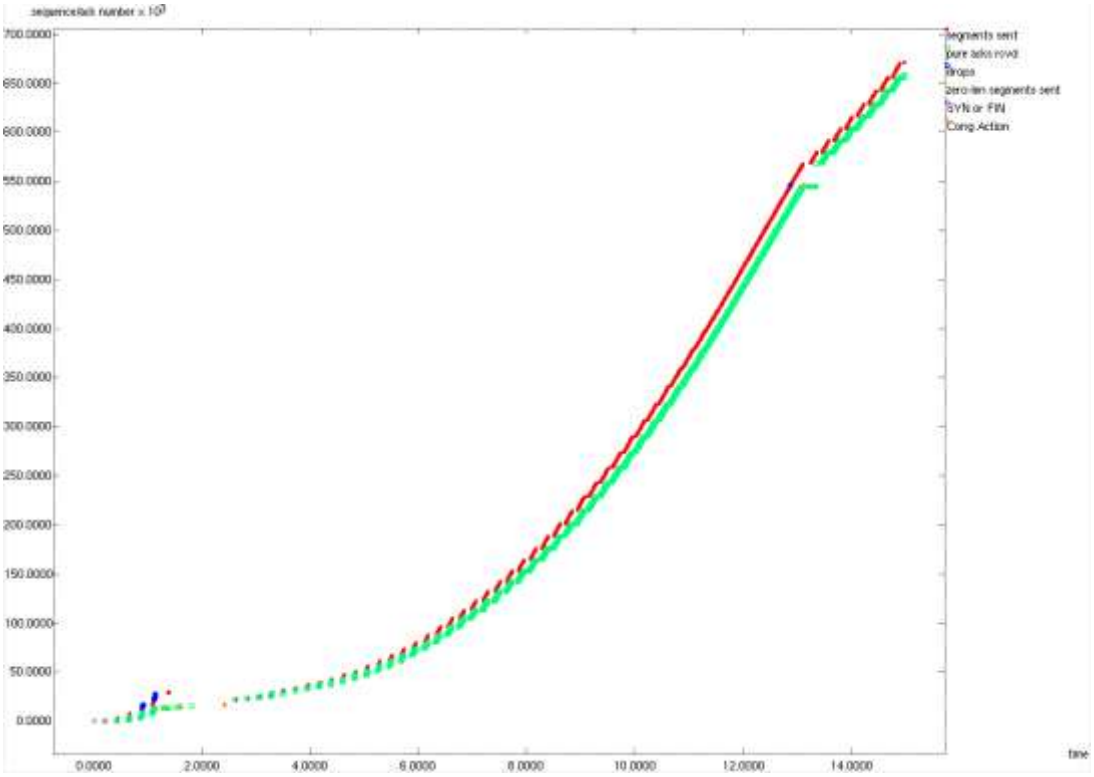

Figura 1. Experimento 1

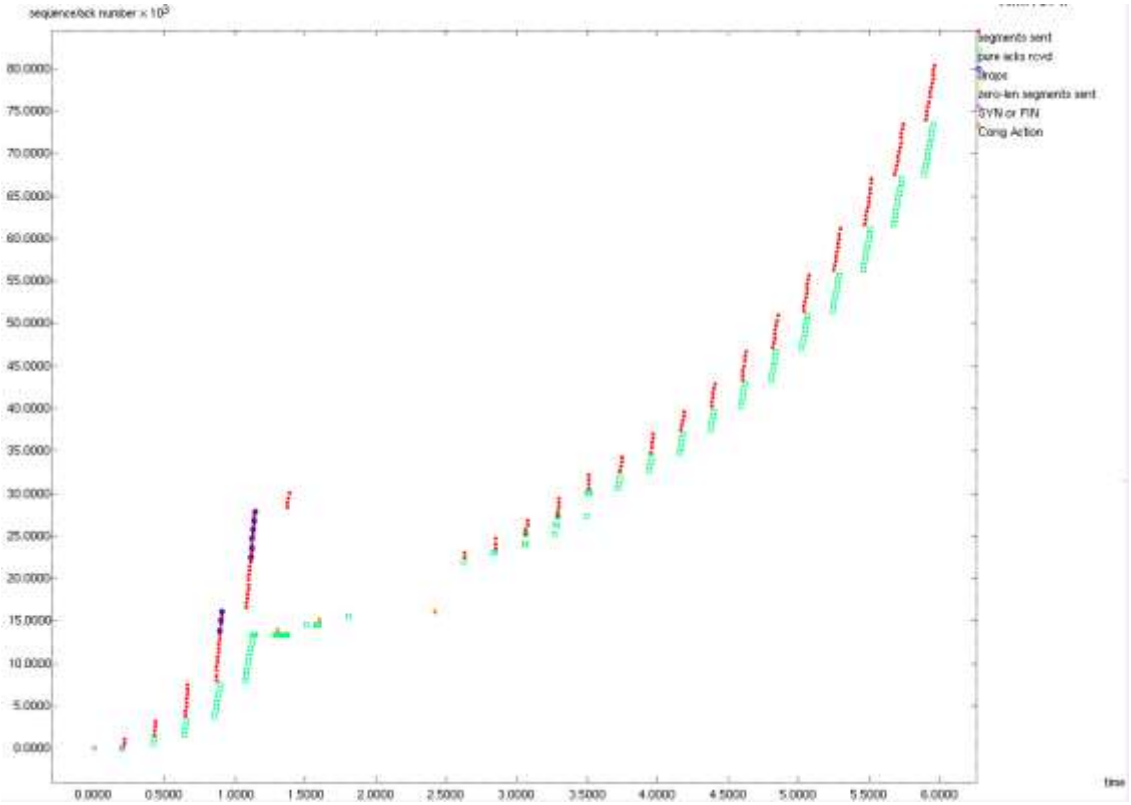

Figura 2. Detalle de los primeros 6 segundos. Experimento 1

En las dos anteriores figuras se aprecia una de las innovaciones que se le hizo al algoritmo TCP Tahoe y es la de aumentar el tamaño de la ventana inicial en 2 segmentos. En realidad como se puede apreciar, el comportamiento del protocolo con un cambio de un segmento en el inicio de transmisión no afecta mucho la transmisión de datos. En la simulación se puede ver que el comportamiento estable de la transmisión (aumento gradual no exponencial de la ventana de transmisión) con una ventana de 1 segmento inicia a los 2.7500 segundos en cambio con una ventana de inicio de 2 segmentos esta etapa “estable” inicia a los 2.6300 segundos.

1.1.2 Experimento 2: Conexión FTP/TCP FULL con ventana inicial de 8 segmentos

Características Principales del Enlace	
Nodos	2
Protocolo	TCP FULL
Aplicación	FTP
Ventana Inicial	8
Tiempo de Simulación	15 segundos

Tabla 2. Características del Experimento 2

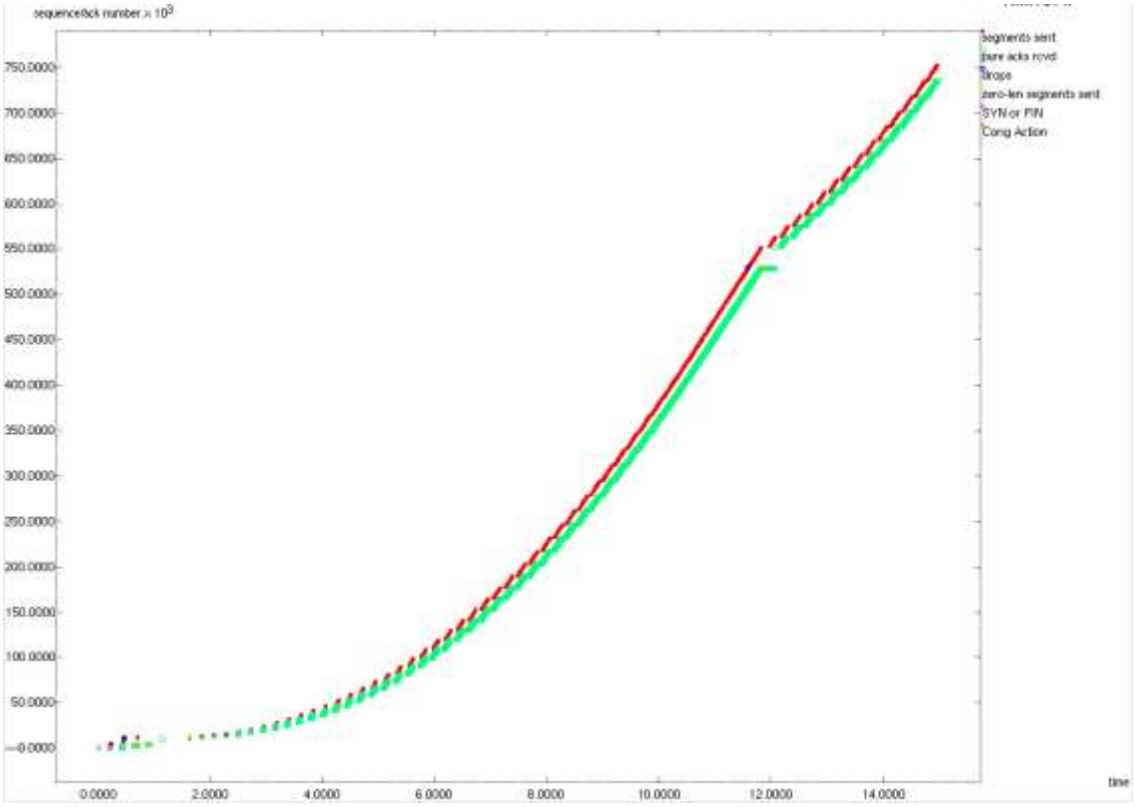

Figura 3. Experimento 2.

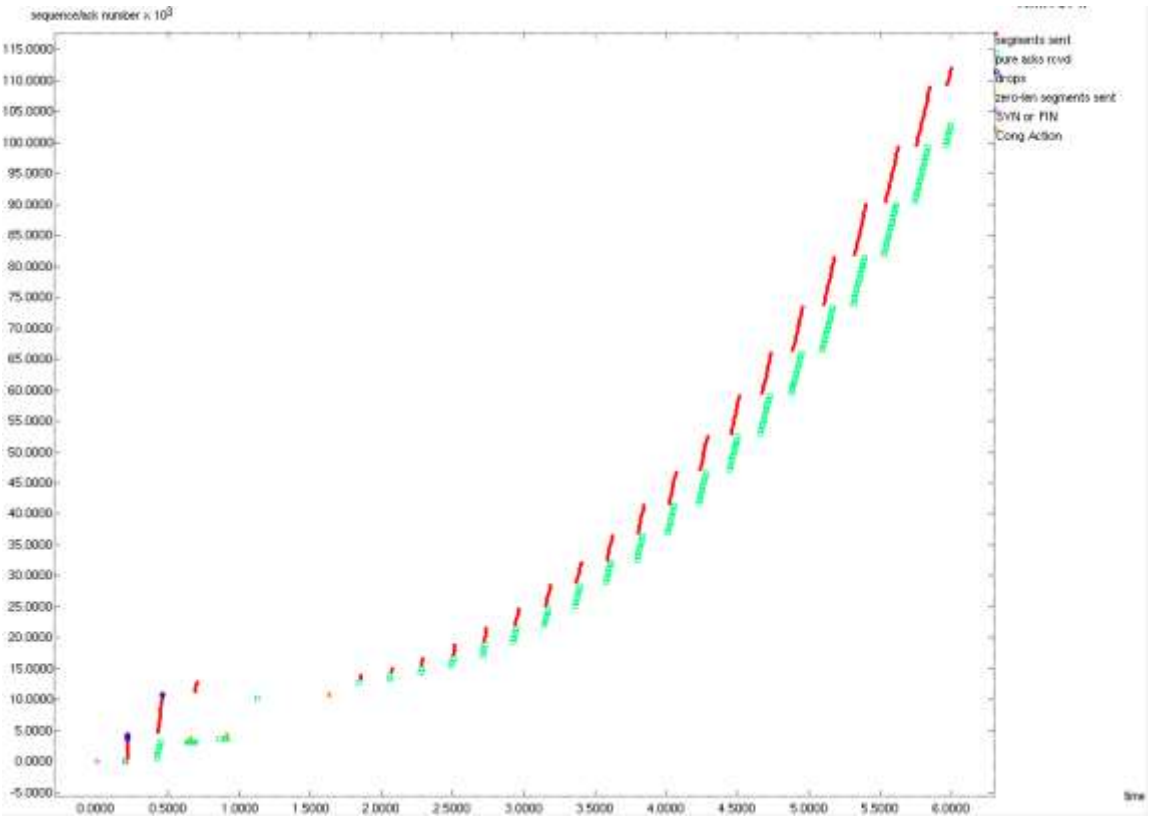

Figura 4. Detalle de los primeros 6 segundos. Experimento 2.

1.1.3 Experimento 3: Conexión FTP/TCP FULL con ventana inicial de 8 segmentos

Características Principales del Enlace	
Nodos	2
Protocolo	TCP FULL
Aplicación	FTP
Ventana Inicial	64
Tiempo de Simulación	15 segundos

Tabla 3. Características del Experimento 3

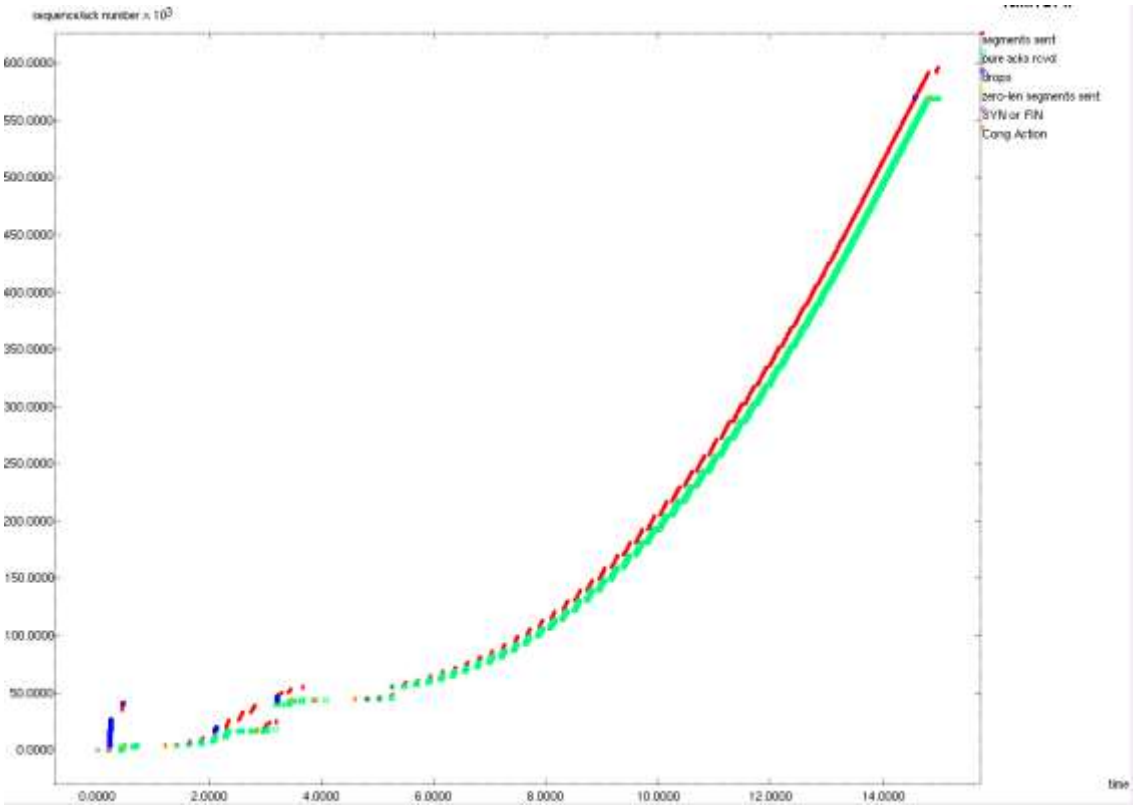

Figura 5. Conexión FTP/TCP FULL con ventana inicial de 64 segmentos

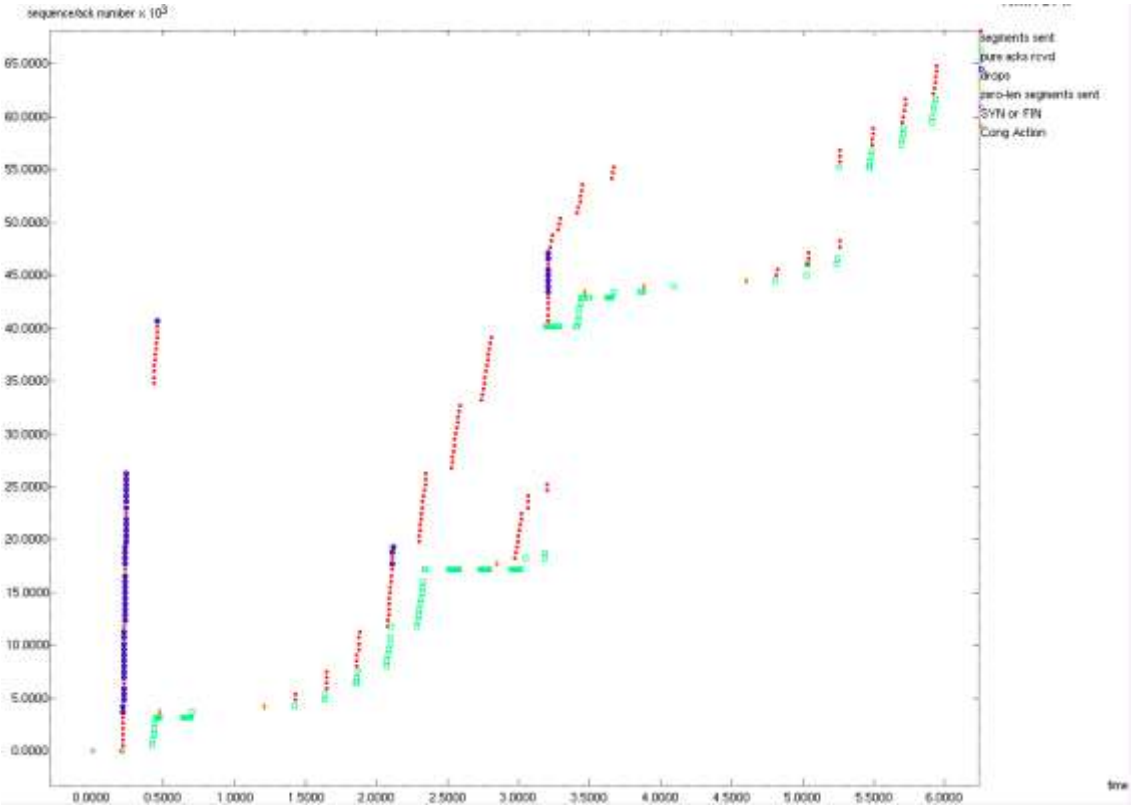

Figura 6. Detalle de los primeros 6 segundos de una conexión FTP/TCP FULL con una ventana inicial de 64 segmentos.

1.2 Escenario 2: análisis de la modificación del umbral del número de DUPACKS (duplicate ACKs) en la respuesta a la congestión de TCP.

1.2.1 Experimento 1: Conexión Tráfico FTP/TCP Tahoe utilizando un valor umbral de 4 para DUPACKS.

Características Principales del Enlace	
Nodos	2
Protocolo	TCP Tahoe
Aplicación	Trafico FTP

Ventana Inicial	2
Ventana de transmisión	50
Umbral DUPACKS	4
Tiempo de Simulación	15 segundos

Tabla 4. Características del Experimento 1

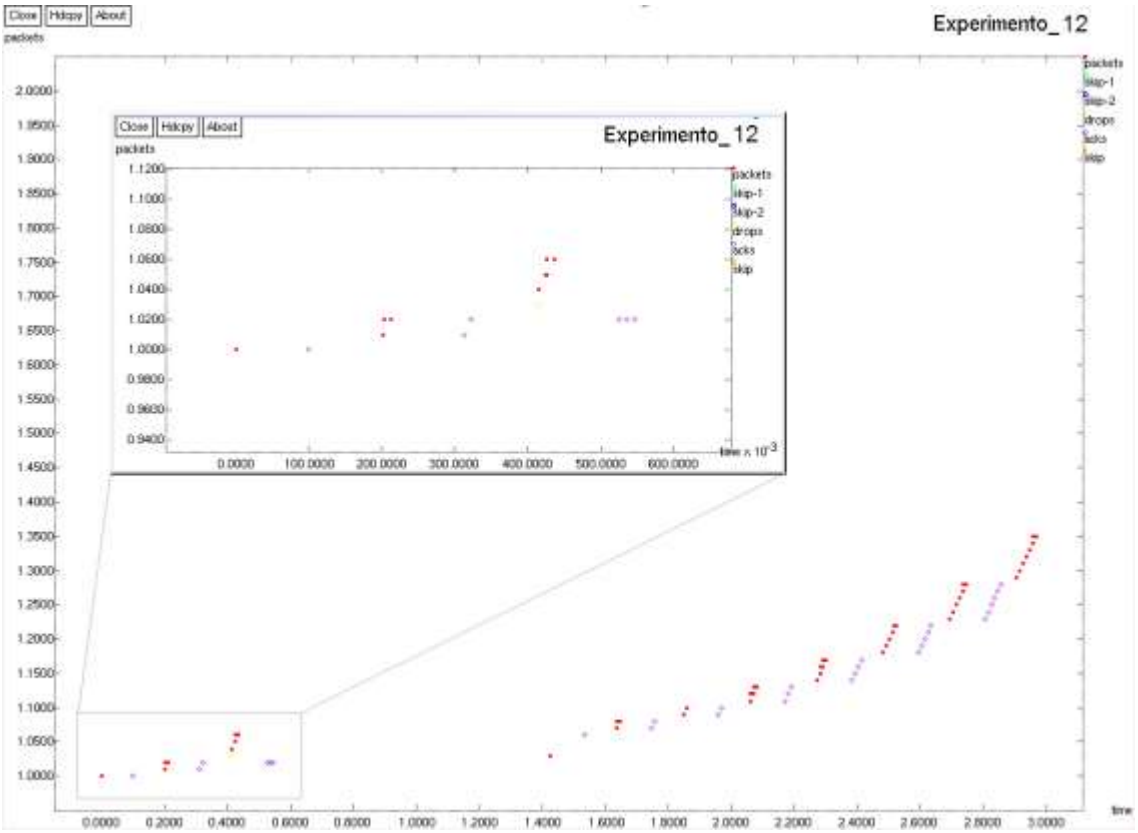

Figura 7. Conexión Tráfico FTP/TCP Tahoe utilizando un valor umbral de 4 para DUPACKS.

1.3 Escenario 3: análisis de la utilización del mecanismo FACK en un enlace TCP entre dos nodos.

Con este escenario se analiza la utilización del algoritmo FACK y comparar su comportamiento con la opción *Reno+SACK* en la recuperación de datos perdidos.

Esta opción sumada al análisis del número de DUPACKs necesarios para activar la retransmisión hacen un buen punto de inicio para implementar un protocolo TCP mejorado y optimizado para redes inalámbricas..

1.3.1 Experimento 1 transmisión de trafico FTP entre dos nodos, utilizando la opción Reno + SACK y FACK cuando se presentan tres paquetes perdidos.

Características Principales del Enlace (figura 33)	
Nodos	2
Protocolo	TCP Reno
Ventana	16 segmentos
Opción SACK	activada
Aplicación	Trafico FTP
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb

Características Principales del Enlace (figura 34)	
Nodos	2
Protocolo	TCP Reno
Ventana	16 segmentos
Mecanismo FACK	activado
Aplicación	Trafico FTP
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb

Tabla 5. Características de los Experimento 17

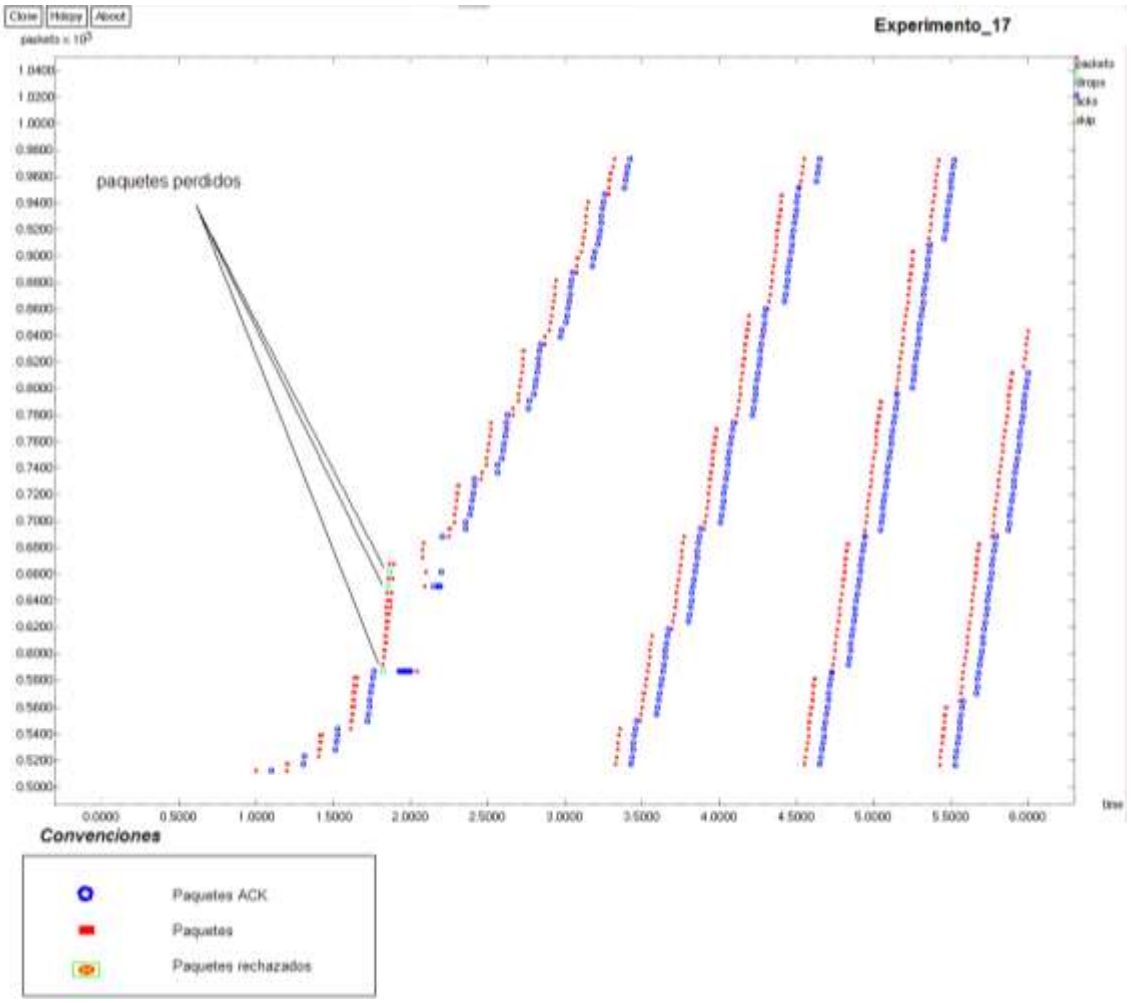

Figura 8. Transmisión de trafico FTP entre dos nodos, utilizando la opción Reno + SACK cuando se presentan tres paquetes perdidos.

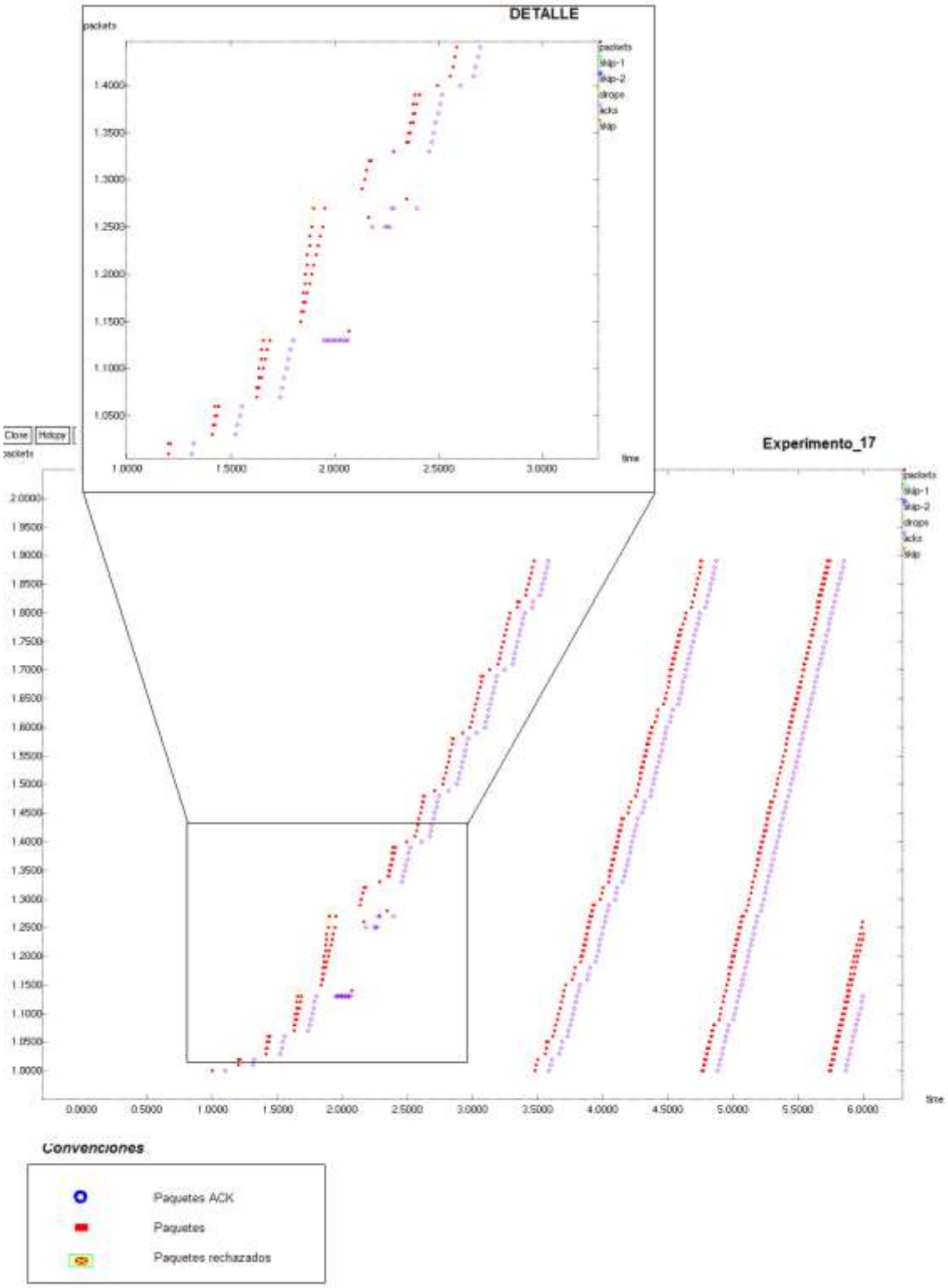

Figura 9. Transmisión de trafico FTP entre dos nodos, utilizando la opción FACK cuando se presentan tres paquetes perdidos.

En la figura 8, se puede apreciar el tráfico entre dos nodos TCP Reno en el cual se presentan pérdidas de paquetes. La opción SACK esta activada en la simulación de la grafica 8, por lo que podemos ver que con la llegada de 3 DUPACKs se inicia la retransmisión de los paquetes faltantes (igual al escenario 4). En la figura 9 se analiza el efecto del mecanismo FACK el cual fue explicado en el capitulo III y su relación con la opción SACK. En este experimento nos podemos dar cuenta que el reinicio de paquetes se hace a los 2.1320 segundos (con la opción FACK) y que con la opción SACK (figura 8) el inicio de la transmisión estable de paquetes se realiza a los 2.0750 segundos; este retraso de unos milisegundos es debido al tratamiento que se hace en el transmisor, ya que el algoritmo FACK realiza mas operaciones que el SACK. Aunque a simple vista el mecanismo FACK no optimiza la transmisión de datos por el retraso existente entre la perdida del paquete y la reacción del mecanismo, se puede utilizar en una implementación TCP inalámbrica debido a que en la mayoría de implementaciones FACK es mas agresiva que TCP Reno o que TCP Reno+SACK, porque puede enviar la repetición de los bytes faltantes (perdidos) con un solo DUPACK.

La implementación de este mecanismo tiene que ser analizado cuidadosamente porque se podría caer en una retransmisión innecesaria de los datos ya que debido a la latencia de los paquetes en redes inalámbricas puede confundir al algoritmo una congestión y un retraso de paquetes debido a latencia o perdida.

ANEXO 2

2 SIMULADOR DE RED NS-2

2.1 Descripción Interna NS

Network Simulator es un simulador discreto de eventos creado por la Universidad de Berkeley para modelar redes de tipo IP. En la simulación se toma en cuenta lo que es la estructura (topología) de la red y el tráfico de paquetes que posee la misma, con el fin de crear una especie de diagnostico que nos muestre el comportamiento que se obtiene al tener una red con ciertas características. Trae implementaciones de protocolos tales como TCP y UDP, que es posible hacerlos comportar como un tráfico FTP (*File Transport Protocol*), Telnet, Web, CBR (*Constant Bit Rate*) y VBR (*Variable Bit Rate*). Maneja diversos mecanismos de colas que se generan en los routers, tales como DropTail, RED, CQB y el algoritmo de *Dijkstra*.

Actualmente, el proyecto NS es parte de VINT (*Virtual InterNetwork Testbed*) proyecto que desarrolla herramientas para visualizar los resultados de una simulación (por ejemplo, interfaces gráficas).
La versión con que fueron efectuadas las pruebas de la presente investigación es la NS versión 2 escrita en los lenguajes de programación C++ y OTcl[1]. Para comenzar se muestra una vista bastante simplificada de lo que es NS. Figura 1.

[1] *Object Tcl (Tool Command Language)* Lenguaje Scripting orientado a objetos, desarrollado por MIT

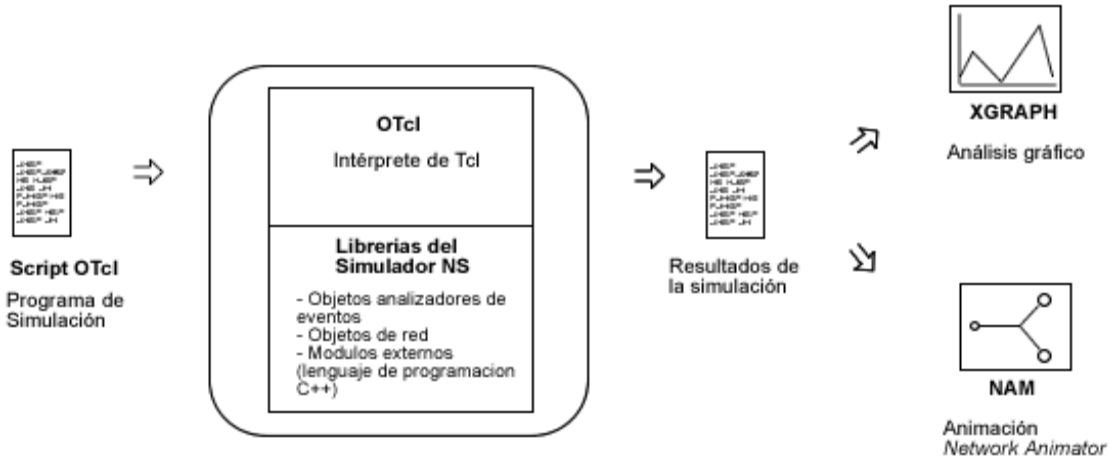

Figura 1. Vista simplificada del proceso de simulación en NS-2.

Como se puede observar, la simulación inicia con un *script* en lenguaje OTcl que viene a hacer lo que el usuario codifica para simular. El script es la única entrada de datos que el usuario ingresa al programa, el resto es el procesamiento interno de NS. La simulación queda en un archivo que es bastante incómodo de leer o analizar para el usuario, sin embargo, utilizando una aplicación especial se puede mostrar mediante una interfaz grafica; los programas que se utilizaron para graficar los datos resultado de la simulación fueron XGRAPH y NAM (*Network Animator*). Estos dos programas vienen con el instalador de NS-2. XGRAPH es una aplicación para el diseño de esquemas en X-Windows (LINUX). NAM es una herramienta de animación basada en Tcl/Tk para la visualización de simulaciones y análisis de paquetes. Soporta el diseño de topologías, animación a nivel de paquetes y varias herramientas para la inspección de datos.

El script es un archivo escrito en Tcl orientado a objetos, es decir, OTcl, que tiene diversos componentes internos que se muestran en el cuadro del medio de la figura 1. En estos componentes se configura la topología de la red, eventos de la red o de los nodos, carga las funciones necesarias para la simulación, planifica cuando iniciar o terminar el tráfico de un determinado paquete, entre otras cosas.

2.1.1 Objeto Planificador de Eventos (*Event Scheduler Object)*

Este evento en NS, es un paquete único con un programa dado en la codificación. Internamente se identifica con un puntero al objeto que maneja este evento. En la figura 2 se muestra la forma de programar los eventos.

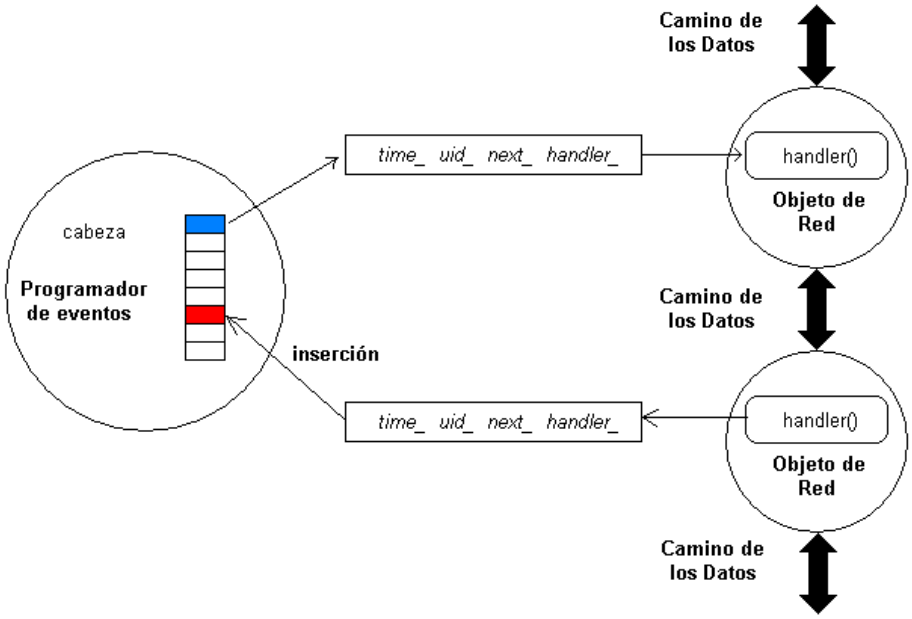

Figura 2. Planificador de Eventos

Los usuarios principales del planificador de eventos es el *Network Component* que se verá a continuación. Esto porque la transmisión de paquetes requiere de ciertos tiempos o retardos necesarios para la simulación. Por ejemplo, al declarar un enlace con un ancho de banda muy bajo, el planificador de eventos deberá realizar retardos más prolongados en ese enlace para simular que la transmisión es lenta.

Por otro lado, cada objeto de red utiliza un planificador de eventos, que es quien maneja el evento por el tiempo planificado. Importante es hacer notar que la trayectoria de datos entre los objetos de la red es diferente de la trayectoria del evento.

2.1.2 Objeto Componente de Red *(Network Component object)*

Se encarga de hacer consistente la comunicación que hay entre distintos componentes de red, por donde pasarán los paquetes. Los componentes de red pueden ser; el ancho de banda de un link, un link unidireccional o bidireccional, retardos de paquetes, etc. En el caso de los retardos también actúa el programador de eventos.

En la figura 3 se muestra el componente de red que permite unir dos nodos, es decir, un enlace simple.

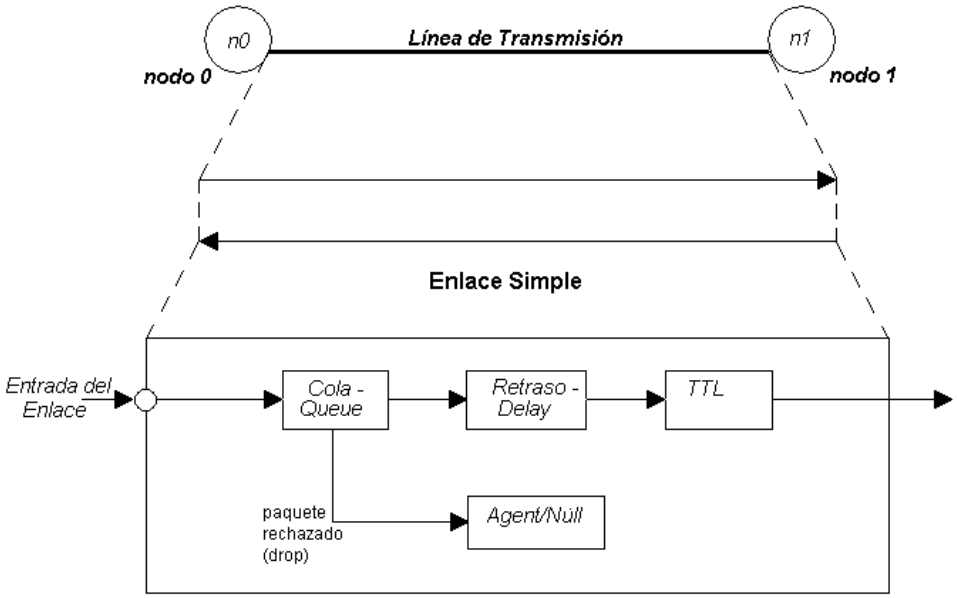

Figura 3. Componentes de red en un enlace con dos nodos.

En esta figura se representa un enlace simple unidireccional. En el caso de requerir uno bidireccional, simplemente se crea otro objeto con la misma estructura para el lado contrario. En la entrada al enlace el paquete deberá quedar en la cola. Se realizan una serie de procesamientos dependiendo del tipo de cola que tenga ese enlace Considerando esto, se tomará la decisión si el paquete es descartado, en cuyo caso pasará a Drop y a un agente NULO. De lo contrario, se realiza un retardo simulado (Delay). Finalmente se recalcula y actualiza el TTL (*time to live*, tiempo de vida) del paquete para llegar al nodo destino Para finalizar, veamos la figura 4 que representa el flujo de paquetes entre los nodos.

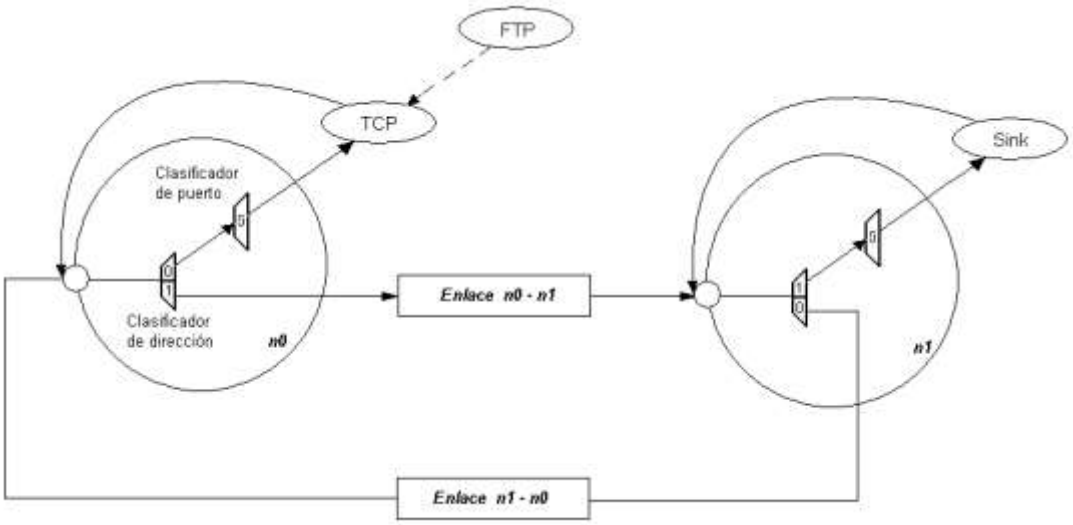

Figura 4: Flujo de paquetes entre nodos

En la figura 4. se quiere representar una comunicación entre 2 nodos mediante el protocolo TCP. Este protocolo requiere de una respuesta de confirmación cuando el receptor reciba el paquete. La idea es la siguiente: la red consiste en 2 nodos (n0 y n1). En el nodo n0, cuando genera el paquete, este sigue el camino por el puerto 0 (*Clasificador de puerto*) para añadir al paquete la información que es de tipo TCP. Luego, siguiendo el camino, vuelve a entrar al nodo n0 y ahora pasa por el puerto 1 para salir por el link n0 ! n1 y llegar al nodo n1. De la misma manera que en el nodo n0, en n1 pasa por el puerto 0 para generar el Sink de respuesta y vuelve a entrar a n1 para salir por el link (puerto 0 de n1) n1 ! n0. Al llegar a n0 entra por el puerto 0 y se genera la confirmación. Luego de esto se genera otro paquete y así se repite para cada transmisión.

2.1.3 Modulo de ayuda para la configuración de la red *(Network Setup Helping Module)*

Por último, los *Network Setup Helping Modules* indicarán las bibliotecas necesarias para realizar la simulación. Esto es necesario ya que los 2 primeros componentes descritos anteriormente, están escritos y compilados en C++ y están disponibles para el intérprete

OTcl a través de un *linkage*[2]. La razón tiene que ver con el tiempo de procesamiento (no de simulación). Se puede hacer la analogía entre C con C++ y tcl con OTcl.
En la siguiente figura se logra mostrar la forma en que se comunican las bibliotecas compiladas de C++ y OTcl. Más bien, es OTcl que llama a estas bibliotecas.

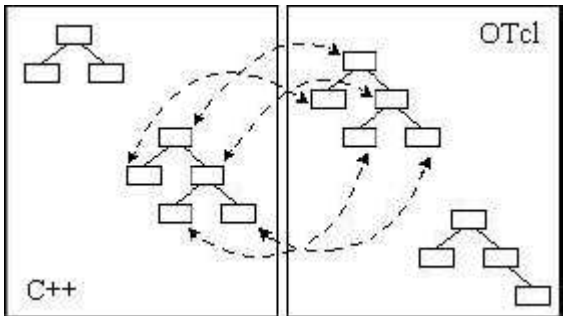

Figura 5: Enlace entre bibliotecas C++ y OTcl

2.2 Ejecución de un script

Para ejecutar la aplicación, Network Simulator toma como entrada (*INPUT*) a un script en OTcl. En este script se define físicamente la configuración de la red (nodos, conexiones entre nodos), los protocolos que serán utilizados en las conexiones y definiciones especificas de aplicaciones usando tipos de conexiones. El script es un archivo con extensión *.tcl*. Para hacerlo correr se debe ejecutar:

```
$ns ejemplo1.tcl
```

desde la línea de comandos (LINUX) y esto crea un archivo que contiene la salida (*OUTPUT*) del análisis, un archivo de extensión .nam (o el similar .tr que más adelante se analiza). Este archivo es una completa descripción de la simulación, donde cada línea describe los paquetes recibidos, enviados, encolados, sacados de la cola, etc. La visualización se realiza mediante el programa nam y se ejecuta simplemente con el comando:

[2] Para enlazar a OTcl las bibliotecas de C++

```
$nam ejemplo1.nam.
```

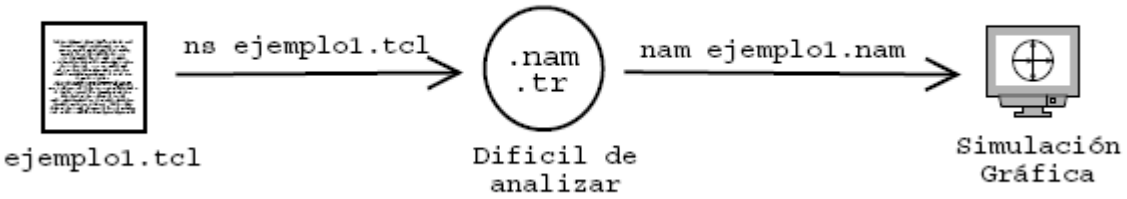

Figura 6. Proceso de generación de los archivos en la simulación.

En la Figura 6 se muestra lo anteriormente explicado:

Aparte de generar el archivo .nam, también se puede generar otro archivo .q (opcional) que contiene información acerca de una cola de un nodo en particular durante la simulación.

2.3 Análisis de trazado

Al ejecutar el comando:

```
$ ns ejemplo1.tcl
```

Se genera un archivo .nam y/o .tr (tienen la misma información pero en distinto formato).

Al ver este archivo, se distinguen todos los eventos realizados durante la simulación línea por línea. Por lo tanto, es poco lo que uno puede concluir. Es por ello que se usa el programa *nam* que tiene por objeto interpretar estos valores y simularlos en una interfaz gráfica.

La diferencia entre el archivo .nam y .tr es que *.nam* es el formato que debe tener para la lectura del programa nam y .tr es un formato más amigable para nosotros si se requiere un análisis. Por lo tanto, desde el punto de vista de contenido son exactamente iguales, pero difieren sólo en el formato. Por lo tanto, el análisis lo concentraremos en el archivo .tr. Es importante saber leer este archivo .tr ya que puede ser de mucha utilidad

a la hora de requerir filtrar ciertos eventos. Eso se puede lograr con un buen manejo en la línea de comandos en Linux, específicamente con el comando `egrep`.

En la Figura 7 se muestra el formato que tiene cada línea.

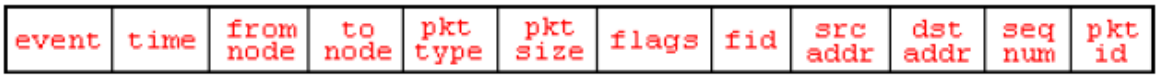

Figura 7. Formato del archivo .nam de la simulación.

Una parte del archivo .tr generado en una simulación es el siguiente:

```
r 0.494 2 3 cbr 1000 ------- 2 1.0 3.1 44 44
r 0.498 1 2 cbr 1000 ------- 2 1.0 3.1 48 48
+ 0.498 2 3 cbr 1000 ------- 2 1.0 3.1 48 48
- 0.498 2 3 cbr 1000 ------- 2 1.0 3.1 48 48
+ 0.5 0 2 tcp 40 ------- 1 0.0 3.0 0 50
- 0.5 0 2 tcp 40 ------- 1 0.0 3.0 0 50
+ 0.5 1 2 cbr 1000 ------- 2 1.0 3.1 50 51
- 0.5 1 2 cbr 1000 ------- 2 1.0 3.1 50 51
r 0.502 2 3 cbr 1000 ------- 2 1.0 3.1 45 45
r 0.506 1 2 cbr 1000 ------- 2 1.0 3.1 49 49
```

Análisis de cada campo:

- r: recibido, +: encola, -: sale cola, d: descartado.
- Tiempo. Fijarse en que a los 5 segundos comienza el tr_a_co TCP, tal como se había propuesto en el ejemplo.
- Nodo fuente.
- Nodo Destino.
- Tipo de paquete.
- Tamaño del paquete.
- *Flags* varios.