CONSTRUCCIÓN DE UNA ARQUITECTURA PARA LA INTEROPERABILIDAD DE LA TECNOLOGÍA BLUETOOTH CON REDES IP CABLEADAS PARA TRANSPORTE DE VOZ

ANEXO A

JUAN PAULO GUZMÁN FLÓREZ
DARYAN FRANCISCO REINOSO ROJAS

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Popayán
2007

Anexo A. Instalación y Descripción del Funcionamiento de la herramienta de simulación Network Simulator 2.

1. Instalación

En esta sección se describirá paso a paso el proceso de instalación de la herramienta de simulación Network Simulator 2 bajo el sistema operativo Linux OS, específicamente Ubuntu 7.07 i386.

1.1. Descarga del paquete de instalación

NS-2 puede ser instalado de dos maneras; en la primera se instalan uno o uno los paquetes necesarios para su funcionamiento (Tcl/Tk, otcl, etc.), y la segunda consiste en la instalación de un solo paquete todo en uno (all-in-one).

Nota: El paquete todo en uno solo funciona bajo sistemas Unix/Linux.

Se debe descargar el paquete desde la página oficial (http://www.isi.edu/nsnam/ns/).

```
$ wget http://nchc.dl.sourceforge.net/sourceforge/nsnam/ns-allinone-
2.29.3.tar.gz
$ tar -xzvf ns-allinone-2.29.3.tar.gz
$ cd ns-allinone-2.29
```

Una vez descargado dicho paquete se deben instalar los requerimientos necesarios para su correcta compilación. Estos son:

PAQUETE	DESCRIPCIÓN
build-essential	Este paquete contiene una lista informativa de paquetes que se consideran esenciales para compilar paquetes Debian.
libxt6	Librería intrínseca de herramientas X11. Provee un conjunto de herramientas de X11.
libxt-dev	Librería intrínseca de herramientas X11 (cabeceras de desarrollo).
libsm6	Librería de gestión de sesión X11. Este paquete provee la interfaz principal para gestión de interfaz X11.
libsm-dev	Librería de gestión de sesión X11 (Librerías de desarrollo).
libice6	Librería de intercambio Inter-Cliente. Este paquete provee la interfaz principal para sesiones de gestión X11.
libice-dev	Librería de intercambio Inter-Cliente (Librerías de desarrollo).
libxmu-dev	Librería de utilidad miscelánea X11 (Cabeceras de desarrollo). Provee un conjunto de utilidades misceláneas con funciones de conveniencia para librerías X11 a usar.
autoconf	Constructor de scripts de configuración automática.
automake	Herramienta para generación de Makefile GNU estándares.

Tabla A1. Paquetes necesarios para la instalación de NS-2

```
$ sudo apt-get install build-essential libxt-dev libxt6 libsm-dev libsm6
libice-dev libice6 libxmu-dev autoconf automake libxmu-dev
```

1.2. Ejecución de la sentencia de instalación.

Al instalar estos paquetes se procede a la instalación de NS-2.

```
$ ./install
```

Cuando la instalación se realice de manera exitosa, se debe poder observar un mensaje instalación correcta, tal como la Figura A1.

Figura A1. Instalación exitosa de la herramienta de simulación NS-2

1.3. Cambio de las variables de ambiente

Luego se deben configurar las variables de ambiente para que se reconozcan los comandos propios de los de la herramienta.

```
$ gedit ~/.bashrc
```

Se deben agregar las siguientes líneas al final del archivo. Se entiende que se debe reemplazar /your/path por la ruta donde se instaló la herramienta.

```
# LD_LIBRARY_PATH

OTCL_LIB=/your/path/ns-allinone-2.31/otcl-1.13
NS2_LIB=/your/path/ns-allinone-2.31/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$X11_LIB:$USR_LOCAL_L
IB

# TCL_LIBRARY
TCL_LIB=/your/path/ns-allinone-2.31/tcl8.4.14/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/your/path/ns-allinone-2.31/bin:/your/path/ns-allinone-
2.31/tcl8.4.14/unix:/your/path/ns-allinone-2.31/tk8.4.14/unix
NS=/your/path/ns-allinone-2.31/ns-2.31/
NAM=/your/path/ns-allinone-2.31/nam-1.13/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Se debe ejecutar el siguiente comando para que los cambio de las variables de ambiente tengan efecto inmediato.

```
$ source ~/.bashrc
```

Por último se puede verificar la instalación con el comando:

```
$ ns
```

entonces un signo "%" aparecerá en la pantalla. Se escribe *exit* para volver al terminal ("$")

1.4. Validación

Después de estos pasos, se puede ejecutar la suite de validación de ns de la siguiente forma:

```
$ cd ns-2.29
$ ./validate
```

A continuación una descripción detallada de NS-2 y se explicará su funcionamiento a profundidad. Este simulador funciona bajo plataforma Linux como bajo plataforma Windows. Por lo tanto, al hablar de “Línea de Comandos" se hará referencia a las consolas de Linux.

2. 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 diagnóstico 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, Telnet, Web, CBR y VBR. Maneja diversos mecanismos de colas que se generan en los routers, tales como DropTail, RED, CQB, algoritmo de Dijkstra, etc.

Actualmente, el proyecto NS-2 es parte de VINT Project, desarrolla herramientas para visualizar los resultados de una simulación (por ejemplo, una interfaz gráfica). Las últimas versiones de NS-2 están escritas en los lenguajes de programación C++ y OTcl[1].

El funcionamiento de Network Simulator se explicará poco a poco mostrando las partes más generales a las más particulares. Para comenzar se mostrará una vista bastante simplificada de lo que se realiza en NS-2.

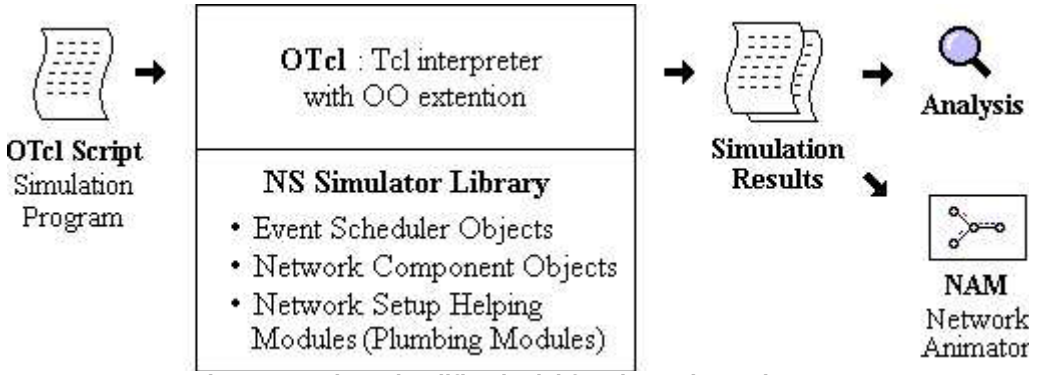


Figura A2: Vista simplificada del funcionamiento de NS-2

Como se puede observar, se comienza con un script en OTcl que viene a hacer lo que el usuario codifica para simular. Es la única entrada que da el usuario al programa. El resto es el procesamiento interno de NS-2. La simulación queda en un archivo que puede ser bastante incómodo de leer o analizar para el usuario, sin embargo, usando una aplicación especial (NAM – Network AniMator) se puede mostrar mediante una interfaz gráfica.

El script es un archivo escrito en Tcl [1] orientado a objetos, es decir, OTcl [2], que tiene diversos componentes internos que se muestran en el cuadro del medio de la Figura A2. En estos componentes se configura la topología de la red, "calendariza" los eventos, carga las funciones necesarias para la simulación, planifica cuando iniciar o terminar el tráfico de un determinado paquete, entre otras cosas.

A continuación se entrará a especificar un poco más como funciona cada componente.

[1] Lenguaje Scripting orientado a objetos, desarrollado por MIT.

2.1. Event Scheduler Object

Este evento en NS-2 es un paquete único con un planificador de eventos o programa dado por el programador en la codificación. Internamente se identificará con un puntero al objeto que maneja este evento. En la Figura A3 se muestra la forma de calendarizar los eventos.

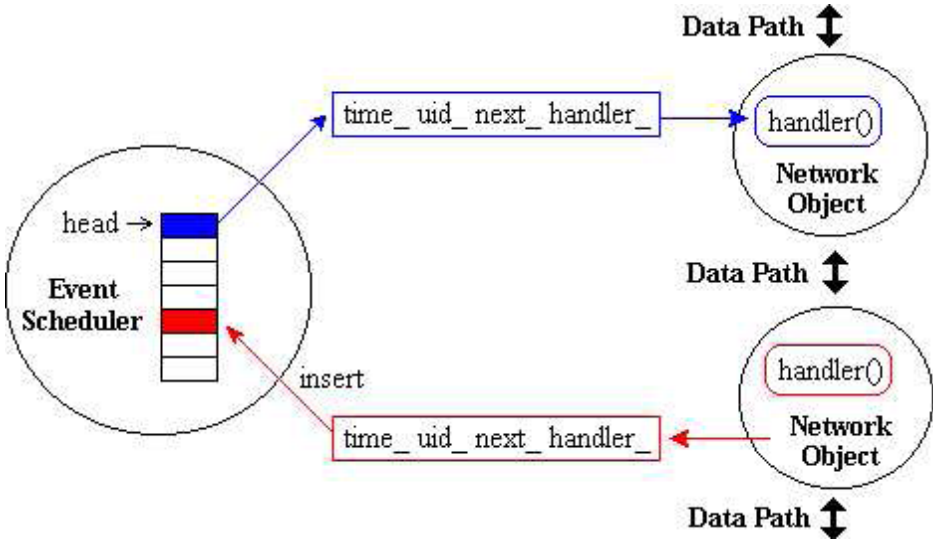


Figura A3: Planificador de eventos.

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 link 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 usa 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 [3].

2.2. 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 event scheduler.

A modo de ejemplo, en la Figura A4 se muestra el componente de red que permite unir dos nodos, es decir, un *link* o enlace.

En esta Figura se representa un *link* 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 *link* el paquete deberá quedar en la cola. Acá se realizarán una

serie de procesamientos dependiendo del tipo de cola que tenga ese *link*, tales como, si el tamaño del paquete supera el tamaño de la cola, o si la cola simplemente está llena, etc. 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 realizará un retardo simulado (*Delay*) del que se hablaba anteriormente. Finalmente se recalculará y actualizará el TTL (*time to live*, tiempo de vida) del paquete para llegar al nodo destino.

Para finalizar, veamos la Figura A5 que representa el flujo de paquetes entre los nodos.

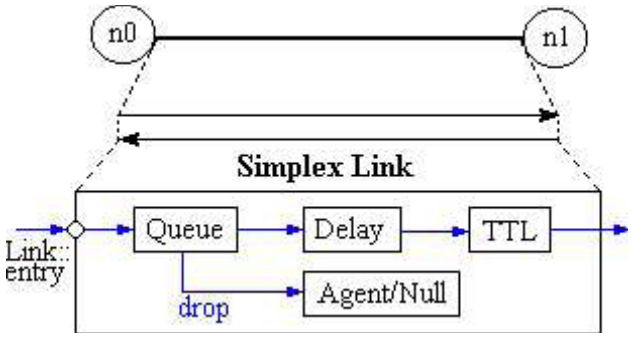


Figura A4: Componente de red – Link

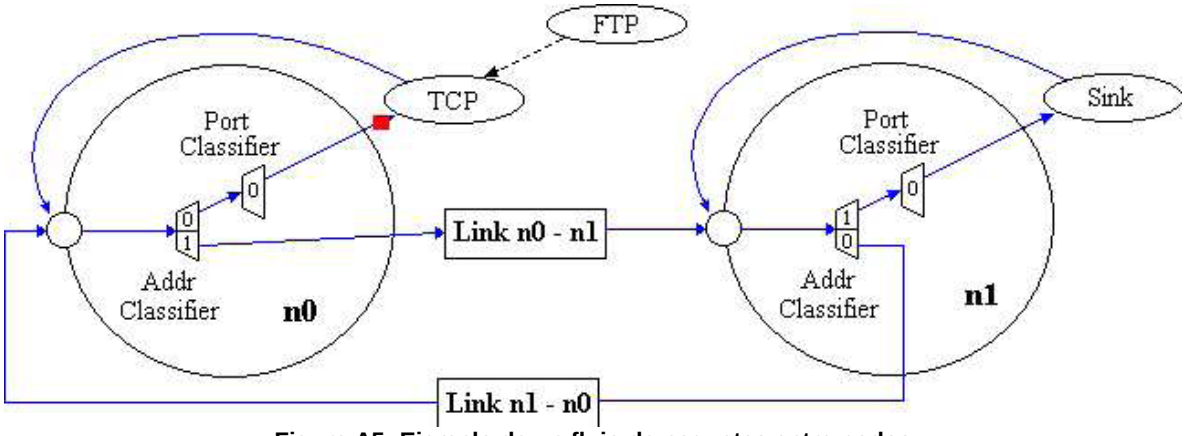


Figura A5: Ejemplo de un flujo de paquetes entre nodos

Acá 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 se genera el paquete, este sigue el camino por el puerto 0 (*port classifier*) 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 [3].

2.3. Network Setup Helping Module

Por último, el *Network Setup Helping Module* indicará las bibliotecas necesarias para realizar la simulación. Esto es necesario ya que los dos primeros componentes, descritos en las secciones 2.1 y 2.2, están escritos y compilados en C++ y están disponibles para el intérprete OTcl a través de un *linkage*[2]. La razón no es muy clara, pero 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 Figura A6 se logra mostrar la forma en que se comunican las bibliotecas compiladas de C++ y OTcl [3].

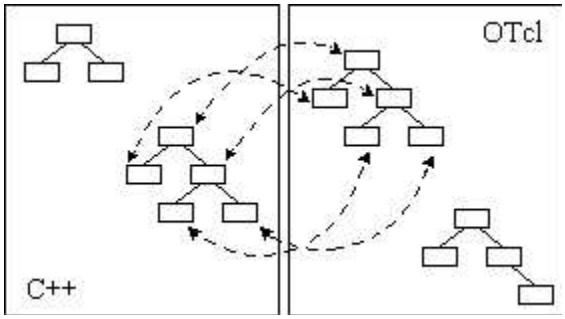


Figura A6: linkage entre bibliotecas C++ y OTcl

3. Ejecutar un script

Para ejecutar la aplicación NS-2 toma como entrada un *script* en OTcl. En este *script* se define físicamente la configuración de la red (nodos, conexiones entre nodos), los protocolos que serán usados en las conexiones y definiciones específicas de aplicaciones usando tipos de conexiones.

El *script* es un archivo con extensión .tcl. Para hacerlo correr se debe ejecutar `ns archivo.tcl` desde la línea de comandos y esto creará un archivo que contendrá el *OUTPUT* del análisis, un archivo de extensión .nam (o el similar .tr que más adelante se analizará la pequeña diferencia). Este archivo es una completa descripción de la simulación, donde cada línea describe los paquetes recibidos, enviados, encolados, sacados de la cola, etc. Sin embargo, por mucho que se mire este archivo, será muy difícil obtener una gran fotografía de lo que sucede en la simulación. Es por ello que la visualización se realiza mediante el programa nam[3] y se ejecuta simplemente con el comando `nam archivo.nam`.

En la Figura A7 se muestra lo recién explicado: Aparte de generar el archivo .nam, también puede generar otro archivo .q (si se específica) que contiene información acerca de una cola de un nodo en particular durante la simulación. Esta puede ser graficada u otros fines [4].

---

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

[3] Nam es Network AniMator, que es una interfaz gráfica para ver la simulación.

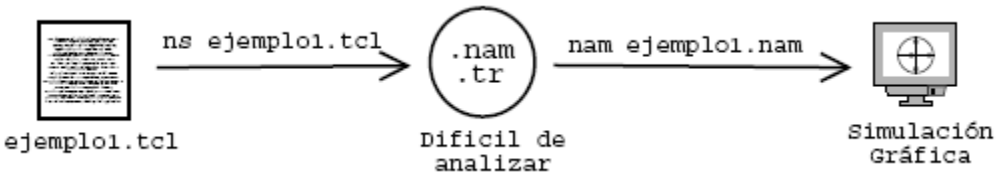


Figura A7. Pasos para realizar la simulación

4. Análisis de trazado

Como se explicó en sección 3, al ejecutar el comando `ns archivo.tcl`, se generará 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 bastante amigable.

La diferencia entre el archivo .nam y .tr radica en que .nam es el formato que debe tener para la lectura del programa nam y .tr es un formato más amigable para el usuario si se requiere un análisis. Por lo tanto, del punto de vista de contenido son exactamente iguales, pero difieren sólo en el formato; así, el análisis lo concentraremos en el archivo .tr. Es importante saber leer e interpretar el archivo .tr ya que puede ser de muchísima 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 A8 se muestra el formato que tiene cada línea [4].

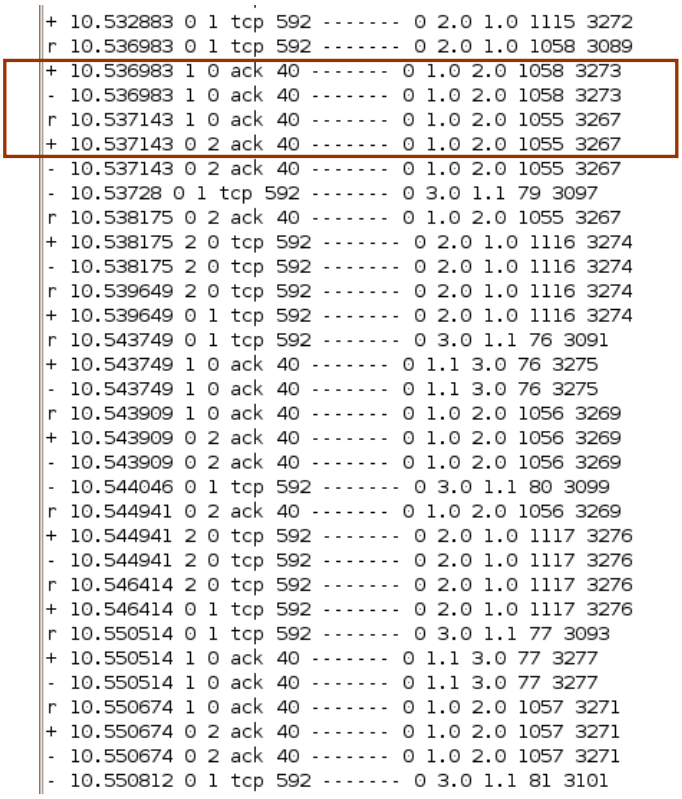
event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

Figura A8: Formato de la estructura en archivo .nam

Análisis de cada campo.

- Evento, r: recibido, +: encola, -: salecola, d: descartado.
- Marca de tiempo.
- Nodo fuente
- Nodo Destino
- Tipo de paquete
- Tamaño del paquete
- Banderas varias

En la Figura A9 se aprecia el registro de los paquetes procesados por el simulador cuando se corre un script ejemplo.



```
+ 10.532883 0 1 tcp 592 ------- 0 2.0 1.0 1115 3272
r 10.536983 0 1 tcp 592 ------- 0 2.0 1.0 1058 3089
+ 10.536983 1 0 ack 40 ------- 0 1.0 2.0 1058 3273
- 10.536983 1 0 ack 40 ------- 0 1.0 2.0 1058 3273
r 10.537143 1 0 ack 40 ------- 0 1.0 2.0 1055 3267
+ 10.537143 0 2 ack 40 ------- 0 1.0 2.0 1055 3267
- 10.537143 0 2 ack 40 ------- 0 1.0 2.0 1055 3267
- 10.53728 0 1 tcp 592 ------- 0 3.0 1.1 79 3097
r 10.538175 0 2 ack 40 ------- 0 1.0 2.0 1055 3267
+ 10.538175 2 0 tcp 592 ------- 0 2.0 1.0 1116 3274
- 10.538175 2 0 tcp 592 ------- 0 2.0 1.0 1116 3274
r 10.539649 2 0 tcp 592 ------- 0 2.0 1.0 1116 3274
+ 10.539649 0 1 tcp 592 ------- 0 2.0 1.0 1116 3274
r 10.543749 0 1 tcp 592 ------- 0 3.0 1.1 76 3091
+ 10.543749 1 0 ack 40 ------- 0 1.1 3.0 76 3275
- 10.543749 1 0 ack 40 ------- 0 1.1 3.0 76 3275
r 10.543909 1 0 ack 40 ------- 0 1.0 2.0 1056 3269
+ 10.543909 0 2 ack 40 ------- 0 1.0 2.0 1056 3269
- 10.543909 0 2 ack 40 ------- 0 1.0 2.0 1056 3269
- 10.544046 0 1 tcp 592 ------- 0 3.0 1.1 80 3099
r 10.544941 0 2 ack 40 ------- 0 1.0 2.0 1056 3269
+ 10.544941 2 0 tcp 592 ------- 0 2.0 1.0 1117 3276
- 10.544941 2 0 tcp 592 ------- 0 2.0 1.0 1117 3276
r 10.546414 2 0 tcp 592 ------- 0 2.0 1.0 1117 3276
+ 10.546414 0 1 tcp 592 ------- 0 2.0 1.0 1117 3276
r 10.550514 0 1 tcp 592 ------- 0 3.0 1.1 77 3093
+ 10.550514 1 0 ack 40 ------- 0 1.1 3.0 77 3277
- 10.550514 1 0 ack 40 ------- 0 1.1 3.0 77 3277
r 10.550674 1 0 ack 40 ------- 0 1.0 2.0 1057 3271
+ 10.550674 0 2 ack 40 ------- 0 1.0 2.0 1057 3271
- 10.550674 0 2 ack 40 ------- 0 1.0 2.0 1057 3271
- 10.550812 0 1 tcp 592 ------- 0 3.0 1.1 81 3101
```

Figura A9: Archivo de salida tr resultado de simulación

En este archivo se puede apreciar el tipo de evento por paquete, la marca de tiempo, la transmisión del nodo 1 al 0, y la transmisión del nodo 0 al 2, así como paquetes del protocolo TCP con su correspondiente ACK y su respectivo tamaño de paquete [4].

Referencias

[1] http://netmedia.gist.ac.kr/~dulee/tclns.html

[2] http://otcl-tclcl.sourceforge.net/otcl/

[3] http://www.isi.edu/nsnam/ns/ns-documentation.html

[4] http://www.isi.edu/nsnam/ns/tutorial/