Análisis de Protocolos de Transporte en Redes Híbridas

Alumno: Alberto Lluch Lafuente
Director: José Miguel Alonso

Proyecto Fin de Carrera, Junio 1999

El acceso a datos a través de redes celulares de telefonía móvil resulta cada vez más común. Dichas redes están basadas en tecnologías inalámbricas, ignoradas cuando fueron diseñados los protocolos de comunicación clásicos. El protocolo de transporte TCP demuestra una especial falta de eficiencia en el uso de los enlaces inalámbricos. En este proyecto se estudiado este tipo de entornos híbridos enfocando el caso concreto de acceso a Internet a través de GSM, y proponiendo alternativas al uso de conexiones TCP de extremo a extremo, basadas en el uso del modelo indirecto de comunicación y la sustitución de TCP por otro protocolo. Para validar estas ideas se han realizado pruebas con el simulador de redes ns: UCB/LBNL Network Simulator, que ha debido ser extendido pues no contaba con la capacidad para modelar los entornos objeto de estudio. Los resultados de las simulaciones han permitido concluir que el uso de conexiones indirectas con un protocolo específico como sustituto de TCP en la parte inalámbrica de la conexión resulta en un aumento considerable de la eficiencia en las comunicaciones. Palabras clave: acceso a datos a través de redes celulares, comunicaciones indirectas, sustitución de TCP, simulaciones.

Gero eta gehiago datuak lortzeko telefonia mobilaren erabilketa sare zelularren bidez egiten da. Sare hauek teknologia inalambrikoetan daude esarrita, teknologia hauek lehendabizi ignoratutak izan ziren, beraien diseinua komunikazio protokolo klasikoekin egin ziren. Garraio protokoloa TCP oso eragite txikia daukate lotura inalambrikoekin. Proyekto onetan estudiatu diren inguru xinitre hauen erabilera estudiatu da, bereiziki Internet-eko sarrerari buruz GSM-ren bidez, eta alternatiba berriak proposatuz TCP ordez extremo batetik bestera esarritak modelo indirektoaren erabilkerarekin eta TCP-aren ordez.Ideia hauek ontzat hartzeko sare simulagaiuan frogak egin dira ns: UCB/LBNL Network Simulator, haundituta behar izan dela ez zuelako kapazitate nahikoa estudiatzen ziren inguruak oreztatzeko. Simulazioen emaitzak argi utzi dute konexio indirektoen erabilketa protokolo espezifiko baten bidez TCP-aren ordez alde inalambrikoarekin komunikazioen obekuntza nabarmena dela eragikortasunaren aldetik. Hitz Esangurutzuak: datuekiko sarrea sare zelularren bidez, komunikazio indirektoak, TCP-aren aldaketa, simulazioak.

Contents

Chapter 1

Memoria

1.1 Introducción

El presente capítulo constituye la memoria del proyecto **Análisis de protocolos de transporte en redes híbridas**, realizado como PFC (Proyecto Fin de Carrera) de la titulación Ingeniero en Informática por Alberto Lluch Lafuente, bajo el tutorazgo de José Miguel Alonso, quien además ideó el proyecto. Este proyecto se integra en una línea de investigación del departamento de A.T.C. de la UPV/EHU en la que participa esta misma persona.

Se trata realizar experimentos con una herramienta de simulación redes ns: UCB/LBNL Network Simulator, que avalen un conjunto de alternativas al actual sistema de acceso a datos a través de GSM.

Uno de los motivos de la elección de este proyecto ha sido la curiosidad por conocer una herramienta de simulación de redes. Se ha preferido también realizar el proyecto para un profesor de la F.I.S.S. frente a hacerlo para un profesor del I.F.F. (Institut Für Informatik) a pesar haber estado disfrutando de una beca SOCRATES/ERASMUS en la Albert Ludwigs Universität (universidad a la que pertenece el I.F.F.), pues se intuía que la realización de un proyecto en el instituto de acogida se limitaría a realizar el trabajo impuesto por un profesor, sin dejar lugar a la creatividad que todo alumno necesita desarrollar. Además el alto riesgo del proyecto ha supuesto un aliciente adicional.

Este riesgo que tiene el proyecto se debe a que no existía la seguridad de que el proyecto fuera realizable, pues las situaciones que se deseaba simular no eran nada comunes, y la probabilidad de que la herramienta no estuviera preparada para ello era bastante alta. De hecho ya antes de comenzar el proyecto se sospechaba que la herramienta de simulación de redes debería ser modificada. La acotación del proyecto, pues no estuvo muy bien definida al comienzo y se fue refinando según éste iba siendo realizando. Inicialmente se pensó en realizar un estudio de la herramienta, un estudio de entornos híbridos de intercomunicación de datos y un conjunto de experimentos para avalar este último estudio. No estaba, sin embargo, muy clara la acotación de cada una de estas cosas, que finalmente quedaron como fases del proyecto, cada una compuesta de varias tareas para la obtención de los entregables del proyecto. Algunas cosas que quisieran haberse realizado (como un estudio basado en modelos analíticos), no pudieron realizarse por falta de tiempo. Así pues, la acotación del proyecto ha

estado marcada por el tiempo disponible más que por cualquier otra cosa.

Para tener una idea del riesgo del proyecto, téngase en cuenta que el tiempo dedicado a éste a sido superior a 500 horas, por encima de las 300 horas que indica el plan de estudios.

1.2 Análisis de Antecedentes

Este proyecto no es revolucionario; las ideas que en el se proponen no son nuevas y la idea de avalarlas mediante simulaciones tampoco. Existen varios antecedentes cuyas ideas han sido recogidas y analizadas en el capítulo 6.

1.3 Análisis de Factibilidad

Al comienzo del proyecto, como ya se ha dicho, la factibilidad del proyecto era algo dudoso. Fue sólo durante los primeros pasos cuando, haciendo uso de literatura científica sobre estudios semejantes al que se estaba realizando, comenzó a haber más seguridad en que el proyecto fuera algo factible.

Se presenta continuación un resumen de lo que otros grupos de investigación han planteado basándose en las mismas ideas que este proyecto, y que hicieron más clara la factibilidad del mismo. En el capítulo 6 pueden encontrarse además ideas alternativas a las propuestas, que fueron recogidas de otros estudios.

Del estudio presentado en [Rivanedeyra et al., 1998a] y [Rivanedeyra et al., 1998b] han sido tomadas la mayor parte de las ideas relacionadas con el acceso a Internet a través de GSM. Básicamente éstas son la ineficiencia que existe en estos sistemas por el uso de TCP sobre enlaces inalámbricos, y la conveniencia de utilizar modelos indirectos de comunicación aislando los problemas en una de las partes de la comunicación, sustituyendo TCP en la parte problemática por otro protocolo diseñado ad-hoc. Las simulaciones que se llevarán a cabo como parte del proyecto servirán de aval a estas ideas.

En [Wang et al., 1998] se recoge un estudio que se asemeja bastante al del proyecto. Estudia entornos distintos de acceso a datos a través de redes inalámbricas (concretamente acceso desde una red local inalámbrica WaveLAN a otra a través de una red WAN alámbrica) , pero de igual manera basa las soluciones en el uso de conexiones indirectas sustituyendo TCP por un nuevo protocolo que funcionará sobre las partes problemáticas (los enlaces inalámbricos). En este estudio se utilizó la misma herramienta de simulación [Network Research Group], aunque en diferente versión. Tuvieron que introducir elementos para implementar comunicaciones indirectas y para modelar los enlaces inalámbricos. El conocimiento de la existencia de este estudio dio grandes esperanzas sobre la factibilidad del proyecto.

[Balakrishnan et al., 1996] y [Balakrishnan et al., 1997] presentan comparaciones de resultados de experimentos basados en simulaciones de entornos de acceso a datos a través de redes inalámbricas utilizando diferentes sistemas. La herramienta de simulación utilizada ha sido la misma que la utilizada en el proyecto, y algunos de esos sistemas implican el uso de conexiones indirectas y la introducción de nuevos protocolos, ideas en las que también se basa el proyecto.

El entorno de intercomunicación de datos que es objeto de estudio de este proyecto, esto es, acceso a Internet a través de GSM, lo es también del es-

tudio recogido en [Kojo et al., 1997]. Se presenta una solución un poco más complicada, pero también basada en sistemas de comunicación indirectos y un tratamiento específico de las comunicaciones en la red GSM. Las pruebas que se realizaron para elaborar este estudio fueron pruebas de campo y no simulaciones.

Diversos estudios (especialmente [Nguyen, 1996]) han tratado el tema de modelar enlaces inalámbricos e implementarlos en la herramienta de simulación de redes ns. No obstante, en todos ellos se han modelado los enlaces con errores, mientras que en el proyecto se pretende modelarlos con aumento en los retardos de propagación causados como consecuencia de la corrección de errores provocada por el uso del modo no transparente en acceso a datos a través de GSM (ver capítulo 6). Este factor es el que supuso la gran duda sobre la factibilidad.

1.4 Documento de Objetivos del Proyecto

1.4.1 Objetivos del Proyecto

Determinar un conjunto viable de alternativas para acceso a datos desde GSM, avalado por unos resultados de simulación positivos como paso previo a pruebas de campo.

1.4.2 Motivación

Se prevé un aumento del uso de GSM para transmisión de datos, utilizando mayoritariamente la familia de protocolos TCP/IP. Puesto que fueron diseñados para redes fijas, se intuye que su funcionamiento en redes inalámbricas es ineficiente. Se quiere comprobar que esto es así y proponer alternativas que demuestren un comportamiento adecuado. Se ha optado por la simulación para avalar las propuestas, pues una implementación directa de las éstas, requeriría mucho esfuerzo, podría ser en vano, y por el bajo coste de simular. La herramienta de simulación elegida ha sido ns: UCB/LBNL Network Simulator [Network Research Group, 1998] por ser adecuada para su uso con redes IP y por ser extensible.

1.4.3 Actividades de cada Fase y Planificación

La realización del proyecto ha sido divida cinco fases: fase de estudio de la herramienta de simulación, fase de estudio de entornos híbridos, fase de preparación de las simulaciones, fase de realización de las herramientas y fase de documentación. Las dos primeras fases se han solapado en el tiempo, mientras que el resto han seguido un orden secuencial.

En principio se llevo a cabo una planificación y gestión basada en los conocimientos expuestos en [Duncan et al., 1996], pero después de una inicial planificación y varias replanificaciones causadas por duraciones excesivas de algunas tareas, finalmente se optó por trabajar tanto como se pudiera y confiar en que el trabajo se terminaría antes del plazo impuesto.

Fase de Estudio de la Herramienta de Simulación

Los objetivos de esta fase son que el equipo de trabajo adquiera los conocimientos suficientes sobre la herramienta de simulación ns: UCB/LBNL Network

Simulator para poder afrontar las fases de preparación y realización de las simulaciones, y que esos conocimientos sean recogidos en diferentes documentos que queden como legado a futuros usuarios de la herramienta. Se presenta a continuación una breve descripción de las actividades que forman parte de esta fase.

Actividad F.1 Instalar ns. Instalar exitósamente la herramienta de simulación de redes ns: UCB/LBNL Network Simulator.

Actividad F.2 Escribir el informe de instalación de ns. Obtener el entregable llamado **Informe de instalación de ns**.

Actividad F.3 Seguir el tutorial. Seguir el tutorial de ns [Greis, 1998] para adquirir un primer contacto con la herramienta de simulación.

Actividad F.4 Leer la documentación de ns. Leer los documentos *ns manual page* [Network Reseach Group, 1997] y *ns Notes and Documentation* [Fall et al., 1998] para profundizar más en los conocimientos de la herramienta.

Actividad F.5 Escribir el manual de funcionamiento externo de ns. Obtener el entregable llamado **Manual de funcionamiento externo de ns**.

Actividad F.6 Escribir el manual de funcionamiento interno de ns. Obtener el entregable llamado **Manual de funcionamiento interno de ns**.

Actividad F.7 Diseñar las simulaciones sencillas. Obtener el diseño de las simulaciones que se presentarán en el entregable llamado **Documento de simulaciones sencillas**.

Actividad F.8 Realizar las simulaciones sencillas. Implementar y ejecutar las simulaciones diseñadas en la actividad F.7.

Actividad F.9 Escribir el documento de simulaciones sencillas. Obtener el entregable llamado **Documento de simulaciones sencillas**.

Fase de Estudio de Entornos de Redes Híbridas

Esta fase tiene como objetivos estudiar la problemática de los entornos de comunicación que implican a redes fijas y móviles, prestando especial atención al caso de acceso a datos a través de GSM, y documentar ese estudio.

Actividad E.1 Leer fuentes documentales sobre entornos híbridos y TCP. Leer artículos y libros que tratan sobre entornos de acceso a datos en redes fijas a través de redes móviles y sobre TCP con el fin de prepararse para realizar la actividad E.2.

Actividad E.2 Escribir el informe de entornos híbridos. Obtener el entregable llamado **Informe de entornos híbridos**.

Fase de Preparación de las Simulaciones

El objetivo de esta fase es preparar la herramienta de simulación añadiendo y/o modificando componentes para que se puedan simular los sistemas descritos en el capítulo 6.

Actividad P.1 Caracterizar enlaces. Caracterizar de manera adecuada los enlaces que forman parte de los entornos de comunicación descritos en el capítulo 6.

Actividad P.2 Implementar enlaces. Implementar en ns los elementos caracterizados en la actividad P.1.

Actividad P.3 Caracterizar comunicaciones. Caracterizar un conjunto de comunicaciones para las simulaciones que se llevarán a cabo.

Actividad P.4 Implementar comunicaciones. Implementar los elementos caracterizados en la actividad P.3.

Actividad P.5 Diseñar Proxy. Diseñar este elemento (el Proxy o intermediario) que ya habrá sido presentado en el capítulo 6.

Actividad P.6 Implementar Proxy. Implementar el elemento diseñado en la actividad P.5.

Actividad P.7 Diseñar el protocolo STP. Diseñar este elemento (el protocolo de transporte STP) que ya habrá sido presentado en el capítulo 6.

Actividad P.8 Implementar el protocolo STP. Implementar el elemento diseñado en la actividad P.7.

Actividad P.9 Diseñar la recogida de resultados. Definir los parámetros de medida de las simulaciones y la forma en que se recogerán.

Actividad P.10 Implementar la recogida de resultados. Implementar los elementos diseñados en la actividad P.9.

Actividad P.11 Escribir el informe de elementos necesarios. Obtener el entregable Informe de elementos necesarios.

Actividad P.12 Escribir el documento de cambios en ns. Obtener el entregable Documento de cambios en ns.

Fase de Realización de las Simulaciones

El objetivo de esta fase es llevar a cabo las simulaciones necesarias para avalar el estudio presentado en el capítulo 6.

Actividad S.1 Diseñar simulaciones. Diseñar el conjunto de pruebas que se van a realizar.

Actividad S.2 Implementar simulaciones. Implementar el conjunto de simulaciones diseñado en la actividad S.1.

Actividad S.3 Realizar simulaciones. Ejecutar las simulaciones implementadas en la actividad S.2.

Actividad S.4 Escribir el informe de simulaciones. Recoger los resultados de las actividades S.1, S.2 y S.3 para obtener el entregable llamado **Informe de simulaciones**.

Fase de Documentación

El objetivo de esta fase preparar la documentación de todo el proyecto para que se ajuste a la normativa de proyectos de la Facultad de Informática de San Sebastián (F.I.S.S.).

Actividad D.1 Escribir el documento de mejoras. Obtener el entregable **Documento de mejoras en el proyecto**.

Actividad D.2 Adecuar documentos a la normativa. Ajustar todos los documentos escritos a la normativa de proyectos de la F.I.S.S..

Actividad D.3 Escribir la memoria. Escribir el presente documento, exigido por y descrito en la normativa de proyectos de la F.I.S.S.

1.4.4 Entregables del Proyecto

Los entregables del proyecto son varios documentos y un conjunto de ficheros de código fuente. Los documentos se entregan como capítulos del documento presente y el código en forma de diskette anexo.

La bibliografía utilizada durante todo el proyecto está unificada al final del presente documento. En ella están incluidos los documentos que forman parte del conjunto de entregables aunque el formato de las referencias a éstos difiere del resto, por claridad.

A continuación se presenta una descripción de cada uno de los entregables.

Informe instalación de ns. Es un informe en el que se indica dónde conseguir la herramienta de simulación ns: UCB/LBNL Network Simulator, qué requerimientos son necesarios para utilizarla, cómo instalarla, qué problemas pueden surgir durante la instalación y cómo encontrarles solución. Recogido en el capítulo 2.

Manual de funcionamiento externo de ns. Es un documento que adiestra en el uso de la herramienta de simulación ns: UCB/LBNL Network Simulator para construir simulaciones básicas, sin contemplar la posibilidad de modificación de la herramienta. Recogido en el capítulo 3.

Manual de funcionamiento interno de ns. Es un documento que da a conocer los conocimientos necesarios de la estructura y los mecanismos internos de la herramienta de simulación ns: UCB/LBNL Network Simulator, para poder modificar la herramienta, incorporando nuevos elementos o cambiando los existentes. Recogido en el capítulo 4.

Documento de simulaciones sencillas. Es un documento que presenta un conjunto de simulaciones sencillas que sirven como ejemplos completos de los conocimientos expuestos en los capítulos 3 y 4. Recogido en el capítulo 5.

Informe de entornos híbridos. Es un documento que trata sobre entornos de acceso a datos en redes fijas a través redes inalámbricas, llamando la atención sobre la problemática que surge por el uso de los sistemas de comunicación clásicos y proponiendo alternativas de mejora. Recogido en el capítulo 6.

Documento de elementos necesarios. Es un documento que recoge el diseño e implementación de los elementos necesarios para simular los sistemas de comunicación presentados en el capítulo 6. Recogido capítulo 7

Informe de simulaciones. Documento que presenta los resultados obtenidos mediante simulaciones de los sistemas presentados en el capítulo 6, que sirven de aval a las propuestas de dicho documento. Recogido capítulo 8

Documento de cambios en ns. Documento que recoge el código fuente que ha sido necesario incorporar y/o modificar para la realización del proyecto. Recogido capítulo 9

Documento de mejoras en el proyecto. Documento que recoge ideas de mejora en diferentes aspectos del proyecto, y que por falta de recursos (principalmente tiempo) no han podido llevarse a cabo. Recogido capítulo 10

Códigos fuente. Código fuente completo incorporado y/o modificado durante la realización del proyecto, y parte del cual se recoge en forma de texto en el capítulo 9.

1.5 Resultados Experimentales

Los resultados experimentales han sido recogidos en el capítulo 8.

1.6 Concordancia entre Resultados y Objetivos

Los objetivos del proyecto han sido plenamente satisfechos a la vista de los resultados obtenidos y recogidos en el capítulo 8.

1.7 Comparación con otras Alternativas

La mayoría de los estudios que han puesto a prueba propuestas similares a las del proyecto, han concluido que estas propuestas suponen efectivamente una mejora frente al sistema actual (se está hablando de acceso a datos a través de redes inalámbricas). Sólo en [Balakrishnan et al., 1996] se dice que el uso de conexiones indirectas TCP no mejora el throughput de las comunicaciones, pero hay que tener en cuenta que el proxy que utilizan está muy limitado en su capacidad lo que supone una gran desventaja en sistemas de comunicación indirectos.

Los resultados de otras alternativas no pueden ser comparados directamente. Esto quiere decir que los valores de throughput y goodput obtenidos no son comparables. La razón está en que las condiciones de prueba de los diferentes sistemas (simulación o prueba de campo) son diferentes en cada estudio.

Si se está interesado en una comparación más o menos justa de diferentes propuestas de sistemas de acceso a datos a través de redes inalámbricas, puede consultarse [Balakrishnan et al., 1996]. Este estudio no incluye, evidentemente, la propuesta del proyecto.

1.8 Conclusiones

A la vista del trabajo realizado se concluye lo siguiente.

La herramienta de simulación de redes ns: UCB/LBNL Network Simulator carece de una buena documentación; la existente está incompleta y, en algunos casos, obsoleta. Esto ha provocado una necesidad excesiva de horas de trabajo para entender su funcionamiento e implementar nuevos elementos en la herramienta. Se han escrito varios documentos (capítulos 2, 3, 4, 5) que pueden evitar que les suceda esto a futuros usuarios.

El comportamiento de los protocolos de transporte tradicionales sobre redes híbridas ha sido objeto de estudio de varios grupos de trabajo. En todos ellos se concluye que el uso de conexiones TCP de extremo a extremo resulta en problemas de ineficiencia cuando hay enlaces inalámbricos de por medio, y proponen nuevas alternativas. En el capítulo 6 han sido recogidas las ideas de estos estudios, profundizando en los problemas de ineficiencia y presentando propuestas de mejora.

Las propuestas de mejora propuestas han sido validadas mediante simulaciones cuyos resultados han sido recogidos en el capítulo 8. Para poder llevar a cabo las simulaciones, han debido ser diseñados e implementados los diferentes elementos que forman parte de ellas. Estos elementos han sido documentados en capítulo 7. Los resultados de las simulaciones han sido positivos, esto es, se ha comprobado la verdad de las ideas expuestas sobre el comportamiento de los protocolos de transporte tradicionales en redes híbridas, y de las nuevas propuestas basadas en el modelo indirecto de comunicación y la sustitución de TCP por otro protocolo específicamente diseñado.

1.9 Agradecimientos

A José Miguel Alonso, por todo el apoyo prestado (indicaciones, correcciones, aclaraciones, recursos, etc.), y también por haber aguantado mis quizá excesivamente largos mensajes de correo. A José María Rivanedeyra Sicilia, por sus datos sobre enlaces GSM, sus ideas sobre experimentos, y su atención. A los autores de las fuentes utilizadas, por su involuntaria ayuda. A Alois Heinz por los recursos facilitados. A Roberto Marrodán García, José Vicente Hernández e Iñigo Bocos Zárraga, por sus simpáticos mensajes de correo. A Haritz Paris Goenaga, por la traducción al euskera del resumen, y por sus gratificantes llamadas a mi lejana residencia en Freiburg, que hicieron más llevadero el trabajo. A Jon Etxebeste Aizpurua por el no menos gratificante intercambio de mensajes de correo, casi las únicas alegrías que encontré en la pantalla de mi puesto de trabajo. A Héctor De Paz Hernández por ser mi hermano en el camino al Valhalla. A mi familia por echarme de menos y apoyarme. A Chiara Marmugi, por hacer de los sueños realidad.

Chapter 2

Informe de Instalación de ns

2.1 Introducción

En el presente informe se pretende ofrecer información, no muy específica en algunos casos, que permita tener una idea de cómo instalar la herramienta de simulación de redes desarrollada por la University of California at Berkeley y llamada ns: UCB/LBNL Network Simulator [Network Research Group, 1998].

Concretamente se explica qué requerimientos son necesarios para instalar la herramienta, dónde conseguirla, cómo construirla una vez obtenida, qué problemas pueden surgir y cómo encontrarles solución.

Este informe está basado en diversas fuentes de información que se especifican más adelante y en la experiencia de haber instalado esta herramienta para la realización del proyecto de cuyo conjunto de entregables forma parte el presente documento.

2.2 Requerimientos

La herramienta ns, ha sido instalada en una estación de trabajo SUN SPARC-Station 4, que funciona con el sistema operativo SUN OS 5.5.

En principio, es posible instalar la herramienta ns en estaciones de trabajo que funcionen bajo Windows 95, Windows NT o UNIX (principalmente FreeBSD, SunOS y Linux). Presumiblemente también es posible instalarla en un ordenador que tenga Windows 98 por sistema operativo.

Es preferible, aunque no indispensable, disponer de un interfaz gráfico, pues algunas de la herramientas que complementan a ns lo requieren.

No se han encontrado datos sobre la capacidad de proceso y memoria requeridos o recomendados. A este respecto, sólo puede decirse que el funcionamiento de la herramienta en el sistema en el que se ha instalado y probado ha resultado suficientemente satisfactorio.

Más adelante se hablará de requisitos de espacio y de la necesidad de disponer de otras componentes software además del simulador.

2.3 Descarga

La fuente principal de la que obtener el simulador ns, es el lugar donde fue desarrollada, es decir, la University of California at Berkeley. A través de la página dedicada a ns[1], es posible descargar la herramienta vía http o ftp.

Además en está página puede encontrarse todo tipo de información relativa a ns, incluida, por supuesto, información más completa sobre la instalación de la herramienta. Pero no sólo eso, sino también documentación del uso y funcionamiento de la herramienta, referencia a un tutorial, demos, e informaciones de otros tipos, útiles para quien desee conocer y/o utilizar el simulador ns.

La descarga puede hacerse vía ftp[2] o http, siguiendo los enlaces de la página de descarga[3]. En está página se explica que la herramienta ns consta de varias piezas, algunas de las cuales son obligatorias (requeridas) y otras optativas (recomendadas). La descarga puede realizarse por piezas o por completo, aunque esta última opción solo está disponible para sistemas UNIX.

Se recomienda la descarga por piezas si ya se tienen algunas de éstas o si se tiene claro cuáles van a utilizarse y cuáles no. Por el contrario, si se va a instalar el simulador por primera vez y no se sabe qué piezas de entre las optativas se utilizarán, ni si se dispone ya de las piezas requeridas, es de recomendar la opción de descarga completa.

Las consecuencias de elegir una u otra opción son diferencias en la construcción de la herramienta y el espacio ocupado. La opción que se ha elegido para la realización del proyecto, ha sido la de descarga completa del simulador. Esto ha permitido comprobar que este tipo de instalación requiere un espacio disponible de 57 MB.

En concreto, los componentes que conforman la herramienta de simulación son las siguientes:

- Tcl release 8.0p2 (componente requerido)
- Tk release 8.0p2 (componente requerido)
- Otcl release 1.0a3 (componente requerido)
- TclCL release 1.0b7 (componente requerido)
- Ns release 2.1b4 (componente requerido) Nam release 1.0a6 (componente opcional). Permite visualizar simulaciones animadas)
- Xgraph versión 12 (componente opcional. Permite visualizar gráficas de parámetros de las simulaciones)
- CWeb versión 3.4g (componente opcional)
- SGB versión 1.0 (componente opcional)
- Gt-itm & sgb2ns 1.1 (componente opcional)

[1] `http://www-mash.CS.Berkeley.EDU/ns`
[2] `ftp://mash.cs.berkeley.edu/dist`
[3] `http://www-mash.CS.Berkeley.EDU/ns/ns-build.html`

(NOTA: cada componente puede constar de varias piezas.)

Aunque la descarga puede realizarse con una conexión directa, la experiencia ha demostrado que esto es tarea casi imposible debido a la lentitud de las comunicaciones entre Europa y América, y el gran tamaño del código fuente (unos 20 MB).

Tras infructuosos intentos de descarga, finalmente fue utilizada una pasarela situada en el Reino Unido[4]. A través de esta pasarela se realizó la descarga del simulador, y su rapidez resultó sorprendente. Una vez se ha accedido a esta página, es necesario escribir el nombre del servidor de ftp de la universidad de Berkeley (`mash.cs.berkeley.edu`). Establecido el contacto se ha de acceder al directorio `dist/vint` desde el cual es posible realizar la descarga completa del simulador eligiendo el fichero `ns-allinone-2.1b4.tar.gz`, que hasta el momento de realización de este informe contiene todo lo necesario para una completa instalación de la última versión de ns.

También es posible utilizar esta pasarela para descargar la herramienta por piezas.

2.4 Construcción

En la página de descarga, pueden encontrarse también instrucciones para la construcción de la herramienta a partir de los fuentes.

Los pasos a seguir en las estaciones UNIX son los siguientes: una vez obtenido el fuente, descomprimirlo (utilizando para ello las herramientas tar y gzip) en el directorio en el que se desea instalar la herramienta. Una vez descomprimidos deben ejecutarse (en este orden) los ficheros `./configure` y `make` desde el directorio `ns`. Tras ello se ha de hacer lo mismo desde el directorio `TclCL`.

Puede encontrarse además un fichero `install` cuya ejecución es equivalente a todo lo anterior.

Una vez hecho esto, es conveniente hacer una verificación de que la instalación ha sido correcta. Esto se hace ejecutando el fichero `ns/validate`. Si todo está en orden, no se presenta ningún mensaje de error, y la instalación puede darse por exitosa.

En el caso de plataformas Windows, el procedimiento es diferente. Como no ha sido instalada la herramienta sobre tales plataformas, no se hablará sobre ello (consúltese la página de descarga y construcción de ns en caso de interés).

2.5 Problemas

Durante la instalación pueden surgir varios problemas, y esto lo ha demostrado la experiencia propia y gran cantidad de mensajes encontrados en la lista de usuarios de ns. La información sobre como apuntarse a esta lista de correos está en el sitio web de ns. No es imprescindible apuntarse, ya que los mensajes de esta lista están puestos a disposición pública en páginas web de este mismo sitio. En este informe no se dirá mucho de los posibles errores de instalación, así que si se desea más información, véase la referencia correspondiente en la página de ns.

[4] `http://www.hensa.ac.uk/cftp/front.html`

Sólo se dirá que durante la instalación para este proyecto, se encontraron dificultades debido a errores en ficheros `configure` y `Makefile`, cuya corrección necesitó de unas tres horas.

Concretamente los fallos encontrados y sus soluciones fueron los siguientes:

1. En el fichero `install` fue necesario añadir el parámetro `-enable-gcc` allí donde se ordenaba la ejecución de ficheros `./configure`, para hacer que se utilizará el compilador gcc.

2. El fichero Imakefile de la componente Xgraph tuvo que ser modificado: se cambió `cc` por `gcc` en todas sus apariciones, pues el compilador cc no se encuentra instalado en la plataforma.

3. El fichero install carecía de espacios en blanco que separasen `./Makefile` de `]` y contenía saltos de línea donde no debía haberlos.

Chapter 3

Manual de Funcionamiento Externo de ns

3.1 Introducción

Existe documentación bastante completa sobre el simulador de redes ns [Network Research Group, 1998], pero quizá un poco inaccesible para quien quiera introducirse en su uso. Un buen ejemplo es el documento ns Notes and Documentation [Fall et al, 1998], muy completo, ahondando incluso en la implementación del simulador, pero difícil de seguir para usuarios noveles. Además la página de manual [McCanne, 1998] que debiera servir como guía de referencia está incompleta y obsoleta. Ninguno de estos documentos ofrece a un usuario novato facilidad para comenzar a utilizar la herramienta y además, en ellos sólo se habla del simulador, sin decir nada sobre herramientas complementarias que hacen más atractivo el uso de ns. Existe un tutorial [Greis, 1998] que sí lo hace. Puede servir guía en los primeros pasos, aunque no como guía de referencia; deja muchas cosas sin definir ni explicar, ya que su objetivo es proveer rápidamente el conocimiento necesario para construir pequeñas simulaciones.

La intención de este capítulo es servir tanto de guía para introducirse en el uso de ns, como de guía de referencia. Podría decirse que es como un manual de usuario introductorio. Se explica cómo definir una simulación y cómo interpretar sus resultados utilizando algunas de las herramientas complementarias al simulador.

NOTA: Se considera que la herramienta ha sido ya instalada. Por lo tanto no se darán indicaciones sobre la instalación. En caso de necesitarse consúltese el informe de instalación de ns (capítulo 2).

El corazón de la herramienta es el simulador ns, que se invoca a través del intérprete `ns` (un programa). Por medio de scripts se dan al simulador las instrucciones necesarias para definir la simulación.

Existen además utilidades complementarias a este simulador. Entre ellas cabe destacar NAM, una herramienta que permite visualizar las simulaciones en forma animaciones. También es interesante la utilización de XGraph para obtener gráficas de diferentes datos de la simulación.

NOTA: Habitualmente estos programas estarán almacenados cada uno en un subdirectorio dentro de un directorio común a todos los componentes de ns,

y serán accesibles desde cualquier lugar por estar incluidas las rutas de acceso a ellos en la variable de entorno correspondiente. Si esto último no es así, conviene por comodidad hacer lo necesario para que así sea. Los ficheros de datos que se utilizarán deberían estar en cualquier otro directorio de los correspondientes a las componentes de ns.

Este capítulo se ha organizado de la siguiente manera. En la sección 2 se habla sobre el simulador ns: como definir una simulación, hacerla funcionar y extraer trazas de su ejecución. En la siguiente sección se presentan dos herramientas complementarias a ns que sirven para visualizar resultados: NAM y XGraph.

3.2 Simulador

Como anteriormente se ha dicho, el núcleo de la herramienta de simulación, es el simulador ns. Es éste un simulador de redes dirigido por eventos. El interfaz que permite definir la simulación es el lenguaje OTcl (Object Tool command Language). En principio, no es necesario conocer este lenguaje de programación para usar la herramienta; basta con conocer los comandos que se presentan en este capítulo. No obstante se recomienda estudiar un poco este lenguaje de programación, pues el esfuerzo será pequeño (sobre todo si se tienen claros los conceptos de programación orientada a objetos) y la recompensa considerable: mejor comprensión de lo que aquí se presenta y mayor capacidad a la hora de desarrollar cosas nuevas. Para la elaboración de este capítulo se ha usado un tutorial de programación en Otcl [Wetheral, 1995] y un manual de referencia de Tcl [Welch, 1995] que es la versión no orientada a objetos de Otcl.

Estos comandos se escribirán habitualmente en un script formando instrucciones que definirán la simulación. Este script será más tarde utilizado por el intérprete para ejecutar la simulación. Para hacer esto se debe invocar desde el shell `ns script` donde `script` es el nombre de del fichero que contiene el script.

NOTA: La llamada a ns es en general tal que así: `ns [file [arg arg...]]`, pero no se ha encontrado información sobre cuáles son los argumentos posibles.

El simulador es, además, extensible y, por supuesto, modificable. Quiere esto decir que es posible añadir nuevas funcionalidades al simulador o cambiar las ya existentes

Son diferentes los aspectos que definen una simulación: la topología, los eventos y la recogida de estadísticas.

La topología de la red la definen los nodos y enlaces, objetos estáticos de la simulación, y los agentes, objetos dinámicos que dirigen la simulación. Además la red tiene otra característica: el encaminamiento (o routing). En un siguiente punto se habla sobre qué son estos objetos, y cómo crearlos y definir sus características.

La definición de eventos permite planificar la simulación. Esto comprende desde indicar el comienzo y final de la actividad de los agentes, hasta reconfigurar la red. Se trata esto con mayor profundidad más adelante.

Para conocer los resultados de la simulación existen mecanismos que permiten recoger estadísticas en ficheros de trazas para su posterior procesamiento. En un siguiente punto se habla de esto.

Todos los objetos que existen en una simulación forman parte de una jerarquía de clases. Se dan por conocidos los conceptos de orientación a objetos;

no se explicarán, pues, términos como métodos, instancias, herencia, etc. En [McCanne, 1998] aparece una jerarquía de clases de ns, pero desgraciadamente incompleta y obsoleta. En [Fall et all, 1998] la jerarquía más o menos definida pero diseminada a lo largo de todo el capítulo.

3.2.1 Inicio

El primer paso a dar para definir una simulación es obtener una instancia del simulador. Esto se hace mediante incluyendo en el script una línea con la instrucción siguiente:

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

NOTA: No se pretende en este capítulo dar a conocer el lenguaje OTcl, pero se incluirán explicaciones cuando ello sea conveniente para comprender lo que se está haciendo.

El método `set` sirve para asignar valores a variables. En este caso, a la variable con nombre `ns` se le asigna como valor una instancia de la clase Simulator. `new` y `delete` son los métodos constructor y destructor de objetos en OTcl.

3.2.2 La Topología

Tres son (básicamente) los conjuntos de objetos que forman parte de la topología de la red y que deben ser creados y configurados: nodos, enlaces y agentes.

Los nodos representan los routers y hosts de la red, tanto su existencia física como parte de los procesos asociados a protocolos de niveles inferiores al nivel de red, éste incluido.

Los enlaces definen accesibilidad directa o unión entre nodos, esto es, representan enlaces directos entre nodos. Junto con los nodos, definen lo que clásicamente (no en este capítulo) se conoce como topología, esto es, la forma de la red.

Los agentes son los objetos que dirigen activamente la simulación: entidades de nivel de transporte, aplicaciones, módulos de routing, etc.

El routing (encaminamiento) forma parte también de la simulación de una red.

Nodos

Los nodos representan en ns a los hosts y routers de la simulación. A ellos se asociarán los agentes que representen aplicaciones o protocolos ejecutándose en dichos nodos, pero esto es tema de otro apartado.

Los nodos son creados mediante el método `node` del simulador, siéndole asignado al nodo creado una dirección única en la red. Las referencias a los nodos creados son almacenadas en la variable `Node_` del simulador, un array que se indexa con identificadores de nodos. Más adelante se indica como obtener el identificador de un nodo.

Para instrucción típica para crear un nodo será la siguiente:

```
set mi_nodo [$ns node]
```

`mi_nodo` es el nombre de la variable que representará al nodo y `ns` es la instancia del simulador (supuesto que éste es el nombre elegido para una previa instanciación del simulador). En OTcl para hacer referencia al valor de una variable debe escribirse el nombre de la variable precedido por el carácter '$'.

A veces puede resultar conveniente almacenar los nodos en un array, en lugar de tener una variable por nodo. Para ilustrar como hacer se presenta el siguiente patrón:

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

Se suponen conocimientos de programación estructurada suficientes para entender la estructura del bucle (inicialización, condición, incremento y cuerpo).

El número de nodos está limitado en principio a 256, pero puede ser expandido a 2^22 mediante la siguiente instrucción,:

```
Node expandaddr
```

Por defecto los nodos son creados para simulaciones con routing unicast (punto a punto). Es posible, sin embargo, crear nodos multicast (punto a multipunto). De esto se habla en el apartado dedicado al routing.

Los objetos nodo poseen varios métodos, de los cuales no se definirá aquí más que un par de ellos, dejando el resto para apartados en los que su presentación es más adecuada.

El método `id` permite obtener el identificador de un nodo, mientras que el método `neighbors` permite obtener la lista de nodos vecinos de un nodo. Los siguientes ejemplos ilustran el uso de estos métodos:

```
set mi_nodo_id [$mi_nodo id]
set vecinos_de_mi_nodo [$mi_nodo id]
```

NOTA: Para saber cómo gestionar una lista, consúltese [Welch, 1995]

Enlaces (y Colas)

Los enlaces son objetos que permiten unir nodos y definir así la conectividad de la red. Están caracterizados por los nodos que unen, el ancho de banda, el retardo y una cola.

Hay que decir que existen dos tipos de enlace básicos en ns: el enlace punto a punto unidireccional y el bidireccional. Existen además otros tipos de enlace (broadcast, múltiple access LAN, etc.), pero no se hablará sobre ellos en este capítulo. Si se desea saber sobre ellos se deberá recurrir a [Fall et al, 1998]

El método que permite crear un enlace unidireccional (respectivamente bidireccional) se llama `simplex-link` (respectivamente `duplex-link`). Es un método del simulador que toma como parámetros el dos nodos, el ancho de banda, el retardo y el tipo de cola. La siguientes dos líneas son un ejemplo de creación de enlaces.

```
$ns simplex_link $nodo1 $nodo2 $mi_bw $mi_delay $mi_tipo_cola
$ns duplex_link $nodo1 $nodo2 $mi_bw $mi_delay $mi_tipo_cola
```

Las variables `ns`, `nodo1`, `nodo2`, `mi_bw`, `mi_delay` y `mi_tipo_cola` se suponen previamente definidas. Variables no definidas en sucesivos ejemplos deben igualmente considerarse como definidas.

NOTA: El retardo(delay) se refiere, naturalmente, al retardo de propagación. Se recuerda, que el retardo total en una transmisión será (tamaño / ancho_de_banda) + retardo_de_propagación.

En principio, ancho de banda y retardo pueden ser omitidos, y automáticamente se les asigna valores por defecto (1'5 Mbps y 100 ms respectivamente). No ocurre así con el caso de los nodos a unir, qué además deben haber sido previamente creados, ni en el caso del tipo de cola a usar.

Para obtener una referencia a un enlace creado, se utiliza el método `link` del simulador como puede verse en el siguiente ejemplo:

```
set mi_enlace [$ns link $nodo1 $nodo2]
```

Debe tenerse en cuenta que en ns, los enlaces se implementan como enlaces unidireccionales. Así la siguiente línea no es equivalente a la anterior:

```
set mi_enlace [$ns link $nodo2 $nodo1]
```

El significado de ancho de banda y retardo se considera falto de necesidad de explicación. Sólo se dirá que, en principio, esto es, sin modificar la implementación del simulador, sus valores son constantes, no queriendo esto decir que una vez determinados sus valores éstos no puedan ser cambiados, sino que no es posible definir valores que se ajusten a una cierta distribución, aproximándose así al mundo real. No obstante, es posible cambiar los valores de ancho de banda y retardo de un objeto enlace *en cualquier momento*. El significado de estas palabras se entenderá cuando se hable de eventos.

De momento sépase que todo objeto cuenta con un método `set` que permite asignar valores a sus miembros variable u obtener sus valores. Así, para cambiar el retardo de un enlace, podría escribirse:

```
$mi_enlace set delay_ 10ms
```

Las variables de los objetos terminan habitualmente con el carácter '_'. El retardo puede también ser definido mediante el método delay del simulador como en la siguiente línea:

```
$ns delay $nodo1 $nodo2 $nuevo_retardo
```

Los valores de retardo y ancho de banda pueden expresarse como enteros o reales. De ser así, se interpretan como bits por segundo en el caso de valores de ancho de banda y como segundos en el caso de valores de retardo. El añadido de sufijos puede cambiar las unidades.

Los caracteres `k`, `K`, `m`, y `M` tras un valor numérico permiten expresar kilos y megas, mientras que los caracteres b y B escritos al final, expresan respectivamente bits y bytes. Los siguientes ejemplos asignan el mismo valor de ancho de banda a cada variable:

```
set ancho_de_banda1 1500000
set ancho_de_banda2 1500k
set ancho_de_banda3 1.5Mb
set ancho_de_banda4 187.5kB
```

En el caso de valores de retardo (o más en general, de tiempo), los sufijos `m`, `n` y `p` permiten expresar respectivamente milisegundos, nanosegundos y picosegundos. Las siguientes cuatro líneas tienen el mismo efecto:

```
set retardo 1.5
set retardo 1500m
set retardo 1.5e9n
set retardo 1500e9p
```

El carácter s puede seguir a cualquier valor de ancho de banda y retardo, haciendo que ns ignore cualquier cosa que no sea una especificación numérica válida.

Colas Las colas representan lugares en los que los paquetes son producidos, consumidos o descartados. Las características distintivas de los tipos de cola son su política de planificación de paquetes (cuáles servir y cuáles descartar) y la estrategia de regulación de su ocupación.

Los tipos de cola disponibles en ns son DropTail (simples colas FIFO), FQ (Fair Queueing, colas que implementan una política más justa que FIFO), SFQ (Stochastich Fair Queueing, implementan una política más justa que FIFO basada en datos estocásticos.), DRR (Deficit Round Robin, colas con una política de planificación Round Robin), RED (Random Early-Detection), CBQ (Class-based Queueing) y CBQ/WRR (Classed Based Queueing/Weighted Round Robin).

Algunas de estos tipos de colas poseen métodos y parámetros específicos. Para saber de ellos y obtener más amplia explicación de ellas, consúltese [Fall et al, 1998]. Aquí únicamente se definirán un par de características comunes a todas la colas.

La capacidad de una cola puede ser limitada con el método queue-limit del simulador o modificando la variable limit de una cola. Los dos siguientes instrucciones son equivalente:

```
$ns queue-limit $nodo1 $nodo2 $nuevo_tamaño
$mi_cola cola set limit_ $nuevo_tamaño
```

Para obtener una referencia a una cola se ha de utilizar el método queue del enlace correspondiente. Así para entender la equivalencia del ejemplo anterior la variable cola debería haber sido establecida de la siguiente manera:

```
set mi_enlace $nodo1 $nodo2
set cola [$mi_enlace queue]
```

Existen además un par de miembros variable de objetos de tipo cola que pueden ser de interés. La variable `bloqued_` indica si una cola está bloqueada y `unblock_on_resume` indica si una cola debe desbloquearse cuando ha sido enviado (y no recibido) el último paquete. El tipo de estas variables es booleano. Valores verdaderos se expresan con `T`, `t` (a lo que puede seguir cualquier cosa, que será ignorada) o cualquier valor entero distinto de cero, mientras que valores falsos se expresan con`F`, `f` o el entero cero. Las siguientes instrucciones cambian el estado (bloqueado, desbloqueado) de una cola.

```
if ([$mi_enlace set blocked_]) {
$mi_enlace set blocked_ false
```

```
} else {
$mi_enlace set blocked_ true
}
```

Es posible además cambiar el estado de los enlaces en lo que respecta a su actividad o inactividad, simulando así cuando un enlace funciona o está caído. Los métodos `up` y `down` de un enlace sirven respectivamente para activar y desactivar un enlace. El método `up?` permite saber si un enlace está activo o no. Las siguientes líneas cambian el estado de activación de un enlace.

```
if (![$mi_enlace up?]) {
$mi_enlace up
} else {
$mi_enlace down
}
```

También ofrece ns un mecanismo para hacer que el comportamiento de un enlace (o de un nodo) en lo que respecta a su funcionamiento o no, siga una cierta distribución aleatoria.

El método `rtmodel` del simulador permite que el estado de un enlace (o de un nodo) siga una distribución o modelo aleatorio. Como parámetros deben aparecer la distribución a utilizar, los parámetros de la distribución y por último el par de nodos que definen el enlace o el nodo. Además la llamada devuelve un handle para el modelo del enlace (o nodo).

El método `rtmodel-delete` del simulador, toma un handle de los mencionados y elimina el modelo del enlace (o nodo) que corresponda, haciendo que a partir del momento de la llamada, el objeto permanezca en un estado constante. La siguiente línea sirve de ejemplo para lo dicho:

```
$ns rtmodel-delete $mi_handle
```

Las distribuciones (o modelos) posibles son Deterministic, Exponential, Manual y Trace.

El modelo Determinista (Deterministic) tiene cuatro parámetros.. El primer y último parámetros son opcionales e indican el comienzo y final de la dinámica del estado del objeto. Por defecto el valor de comienzo es 0'5. No conviene establecer valores de comienzo menores a 0'5, pues los protocolos de routing necesitan un cierto tiempo de preprocesado. Si no se expresa valor de final, el estado sigue la distribución hasta el final de la simulación. Los parámetros segundo y tercero determinan las duraciones de los intervalos durante los cuales el objeto estará activo o no. El valor por defecto de ambos es de 2'0 s y 1'0 s respectivamente. La siguiente línea ilustra lo dicho:

```
$ns rtmodel Deterministic 1.0 800ms 200ms $nodo1 $nodo2
```

El modelo Exponencial (Exponential) tiene dos parámetros, que son las medias de las distribuciones exponenciales que se seguirán los periodos de actividad e inactividad. Los valores por defecto son 10 s y 1 s respectivamente. La siguiente línea instala un modelo exponencial en un nodo y almacena el handle.

```
set mi_handle [ns rtmodel Exponential 0.2s 0.1s $mi_nodo]
```

La distribución manual consiste simplemente en decidir cuándo sucederán las caídas y reactivaciones. Es posible hacerlo mediante eventos (más adelante se explican) y los métodos `up` y `down`, o con el método `rtmodel-at` del simulador. Toma como primer argumento el instante de tiempo en que ocurrirá el cambio de estado, tras lo cual siguen el estado al que pasar (up o down) y el enlace (definido por los dos nodos que lo forman) o el nodo. Es realmente una forma de definir eventos especiales. Una instrucción típica podría ser la siguiente:

```
$ns rtmodel-at $tiempo_caida down $nodo1
```

Usando el modelo `Trace` (traza) los eventos se leen de un fichero. El único parámetro de este modelo es el handle del fichero de traza que tiene la información de la dinámica del enlace o nodo. En el apartado dedicado a las trazas se hallarán más explicaciones sobre ficheros de traza. Las siguientes líneas ilustran esto, incluyendo instrucciones de apertura de ficheros:

```
set mi_traza [open mi_fichero_de_trazas.tr r]
$ns rtmodel Trace $mi_traza $nodo1 $nodo2
```

Agentes

Los agentes son las entidades que dirigen activamente la simulación. Pueden verse como procesos ejecutándose en un nodo y realizando funciones de protocolos, de aplicaciones generadoras y/o consumidoras de tráfico o módulos de routing dinámico. De esto último no se hablará aquí, sino más adelante. Trata esta sección de agentes en general, profundizando luego en dos tipos: agentes de transporte y agentes de aplicación.

Primero de todo se dirá que en general, los agentes se crean instanciando una subclase de la clase Agent. Cuando un agente es asociado a un nodo recibe automáticamente un número de puerto único en ese nodo. Haya gentes no generan tráfico ellos mismo y otros que lo generan, necesitando ser asociados a un agente de transporte u otro tipo o directamente a un nodo.

Para asociar un agente a un nodo se utiliza el método `attach-agent` del simulador. La siguiente línea es una instrucción típica de asociación de un agente a un nodo.

```
$ns attach-agent $mi_nodo $mi_agente
```

El método del simulador `detach-agent` permite eliminar la asociación de agentes a nodos. Los parámetros de este método son el nodo y el agente.

Los agentes deben ser conectados entre sí de manera explícita, antes de comiencen a emitir o recibir tráfico. El método `connect` del simulador toma dos agentes como parámetros y establece una conexión entre ellos. También es posible hacerlo mediante el método `connect` de un agente, pasando como parámetros la dirección del nodo y el puerto de destino. Más adelante se explica como obtener el puerto de un nodo. La siguiente línea sirve de ejemplo de conexión entre agentes:

```
$ns connect $mi_agente $tu_agente
```

Asociados a un agente existen además los siguientes métodos: `reset` inicializa un agente, los métodos`dst` y `port` devuelven la dirección y el puerto que identifica al agente en el nodo en el que está, mientras que `dst-addr` y `dst-port` devuelven

la dirección y el puerto de destino. Las direcciones corresponden a direcciones de agentes formadas por ocho bits inferiores que indican el nodo y otros ocho bits más. En caso de haber expandido el número máximo de nodos a utilizar, los bits correspondientes al nodo se elevan hasta 22.

En siguientes puntos se profundiza en el uso de agentes distinguiendo entre agentes de nivel de aplicación y agentes de nivel de transporte. Cabe decir que para la interacción entre ambos tipos de agentes, hay definido un interfaz que se debe conocer si se desea implementar nuevos agentes de transporte o aplicación. Como esto es algo que pertenece a los mecanismos internos de ns, no se cuenta aquí sino en el capítulo de funcionamiento interno de ns.

Agentes de transporte En ns hay varios protocolos de transporte implementados. De ellos se tratarán aquí solamente los protocolos de las familias UDP y TCP.

Los agentes UDP son una implementación del protocolo de transporte no orientado a conexión UDP. Este protocolo se utilizará normalmente para ofrecer servicio de transporte a agentes de aplicación que no requieran servicio de conexión. En ns tales aplicaciones se conocen como aplicaciones generadoras de tráfico y están definidas bajo la clase Application/Traffic. Los agentes UDP pertenecen todos a la clase Agent/UDP. Crear un agente UDP supone escribir una línea como la siguiente:

```
set mi_transporte [new Agent/UDP]
```

Un agente de transporte UDP que soporte un generador de tráfico deberá asociarse al mismo nodo en el que se encuentre el agente de transporte y conectarse con otro agente de transporte o a un agente especial Null (sumidero universal) si no hay aplicación que consuma el tráfico generado. Las siguientes líneas ilustran parte de lo dicho.

```
set mi_transporte_sumidero [new Agent/Null]
set mi_transporte_emisor [new Agent/UDP]
$ns attach-agente $mi_nodo_emisor $mi_transporte_emisor
$ns connect $mi_transporte_emisor $mi_transporte_sumidero
```

Los agentes UDP cuentan con un miembro variable `packtSize_` modificable que indica el tamaño máximo de los paquetes.

Existen diferentes implementaciones del protocolo de transporte TCP en ns. Pueden distinguirse dos grandes grupos: agentes TCP unidireccionales y agentes TCP bidireccionales.

Los unidireccionales se dividen a su vez en emisores y receptores. Los agentes emisores pueden ser TCP (Tahoe TCP), TCP/Reno (Reno TCP), TCP/NewReno (Reno TCP con una modificación), TCP/Sack1 (TCP con ACKs selectivos siguiendo la norma RFC2018), TCP/Vegas (TCP Vegas) y TCP/Fack(TCP Reno con *forward acknowledgement*).

Los agentes receptores soportados son TCP/Sink (sumidero de TCP que envía un ack por paquete), TCPSink/DelAck (sumidero de TCP con retardo de ACK configurable), TCPSink/Sack1 (sumidero de TCP con ACKs selectivos según la norma RFC2018) y TCPSink/Sack1/DelAck (combinación de los dos anteriores).

El protocolo TCP se usará, en general, para ofrecer servicio de transporte orientado a conexión a agentes de nivel de aplicación que mantengan comunicaciones unidireccionales, como aquellos de las clases Application/Telnet y Application/FTP. La aplicación emisora se asociará a un agente TCP emisor y la aplicación receptora (si la hubiera), se asociará a un agente TCP sumidero. Los agentes de transporte deberán ser asociados cada uno en el nodo que corresponda y conectados entre sí mediante el método `connect`. En las siguientes líneas se presenta un ejemplo.

```
set mi_transporte_emisor [new Agent/TCP]
set mi_transporte_receptor [new Agent/TCPSink]
$ns attach-agent $mi_nodo_emisor $mi_transporte_emisor
$ns attach-agent $mi_nodo_receptor $mi_transporte_receptor
$ns connect $mi_transporte_emisor $mi_transporte_receptor
```

El único agente TCP bidireccional implementado es TCP/FullTCP (Tahoe TCP). Este protocolo ofrece servicio de transporte orientado a conexión a aplicaciones con comunicación unidireccional o bidireccional. Los agentes de transporte creados y asociados a las aplicaciones son ahora de la misma clase, pero uno de ellos, el que vaya a recibir la petición de apertura de conexión debe ponerse en un estado de escucha mediante el método `listen`. A continuación se presenta un conjunto de instrucciones típicas.

```
set mi_transporte_emisor [new Agent/TCP/FullTcp]
set mi_transporte_receptor [new Agent/TCP/FullTcp]
$ns attach-agent $mi_nodo_emisor $mi_transporte_emisor
$ns attach-agent $mi_nodo_receptor $mi_transporte_receptor
$ns connect $mi_transporte_emisor $mi_transporte_receptor
$mi_tranporte_receptor listen
```

Cabe decir que este tipo de agentes de transporte bidireccionales es incompatible con el anterior.

Queda fuera del ámbito de este capítulo, explicar el funcionamiento de las distintas implementaciones de TCP. Se presentarán únicamente los parámetros de los agentes TCP utilizables para configurar su comportamiento.

Objetos de la clase TCP, y por consiguiente todas sus subclases, tienen los siguientes parámetros:

window_: cota superior de la ventana de emisión. El tamaño de la ventana (o tamaño del crédito) permanece constante y se mide en segmentos.

maxcwnd: cota superior de la ventana de congestión. Por defecto es cero(se ignora).

windowInit_: tamaño inicial de la ventana de congestión cuando se usa el algoritmo slow-start.

windowOption_: algoritmo de gestión de la congestión (1=estándar).

windowTresh_: constante de ganancia exponencial usada para computar la ventana de congestión.

overhead_: rango de una variable aleatoria uniforme usada para retrasar el envío de cada paquete.

ecn_: determina si se usa notificación explícita de error o no.

packetSize_: tamaño de los paquetes.

tcpTick_: granularidad del reloj TCP para medir el tiempo de round-trip.

bugFix_: determina si eliminan errores al usar la técnica de retransmisiones rápidas.

maxburst_: determina el número máximo de paquetes que la fuente puede enviar para responder a un simple ack. Si es cero, se ignora.

slow_start_restart_: indica si se dene activar el algoritmo slow-start tras cada periodo de inactividad.

Además cada objeto cuenta con las siguientes variables de estado:

dupacks_: número de acks recogidos, desde que un nuevo paquete fue aceptado.

seqno_: mayor valor de secuencia recogido.

t_seqno_: número de secuencia actual.

ack_: mayor ack recibido por el receptor.

cwnd_: valor actual de la ventana de congestión.

awnd_: valor actual de la versión low-pass filtered de la ventana de congestión.

ssthresh_: valor actual del *treshold* del algoritmo slow-start.

rtt_: tiempo de round-trip estimado.

srtt: estimación del tiempo fluido de round-trip.

rttvar_: desviación media del tiempo de Round-trip.

backoff_: constante de backoff exponencial del tiempo de round-trip.

Algunas de las subclases tienen parámetros específicos. Para más información consúltese [Fall et al, 1998].

Agentes de aplicación Las aplicaciones se sirven de los agentes de transporte para mantener comunicaciones. Las comunicaciones en ns no implican intercambio real de datos. Se simula el intercambio de datos utilizando cantidades expresadas en bytes. Existe, sin embargo, la posibilidad de transmitir datos reales a nivel de aplicación, utilizando mecanismos distintos de los que aquí se explicarán.

Las aplicaciones, por tanto, se limitan a enviar y/o recibir cantidades de bytes. Puede distinguirse entre aplicaciones simuladas (FTP y Telnet) y aplicaciones generadoras de tráfico, pues unas usarán agentes de transporte TCP y las otras, agentes de transporte UDP. En general, durante el ciclo de vida de las aplicaciones, éstas se crean, se asocian al agente de transporte correspondiente (mediante el método `attach-agent`), se lanza (mediante el método `start`) y,

por último, se hacen finalizar (mediante el método `stop`). A continuación se dan detalles de algunas de las clases de aplicación provistas por ns.

La clase Application/FTP implementa la aplicación FTP. Dicha aplicación cuenta con los métodos `start` (provoca envío continuo de datos), `produce` (establece la cuenta de paquetes a enviar con el valor de su único argumento), `producemore` (incrementa la cuenta de paquetes a enviar con el valor de su único argumento), `send` (similar al anterior, pero el argumento se expresa en bytes). Las siguientes instrucciones son típicas:

```
set mi_aplicación [new Application/FTP]
$mi_aplicación attach-agent $mi_transporte_emisor
$mi_aplicación start
```

La clase Application/Telnet implementa una aplicación que genera paquetes individuales con un intervalo que sigue una cierta distribución, simulando el comportamiento de una aplicación Telnet real. Un objeto de la clase Application/Telnet cuenta con los métodos start y stop para lanzar y detener la ejecución, y con un parámetro llamado `interval_`. Dicho intervalo determina la distribución que siguen los tiempos de intervalo entre envío de mensajes. Si el valor del miembro variable `interval_` es cero, el intervalo sigue una distribución llamada tcplib, de lo contrario sigue una distribución exponencial con el valor asignado.

Bajo la clase Application/Traffic se agrupan varias clases que implementan distintos generadores de tráfico. Dichos generadores de tráfico están caracterizados por distribuciones aleatorias. Lo normal es que aplicaciones de este tipo utilicen agentes de transporte UDP, aunque no esto no es obligatorio.

La clase Application/Traffic/Exponential implementa agentes que genera tráfico de acuerdo a una distribución exponencial ON/OFF. Paquetes de tamaño constante son enviados durante periodos ON, mientras que durante periodos OFF no se envía nada. Las duraciones de cada periodo siguen distribuciones aleatorias. Los métodos `start` y `stop` sirven para lanzar y detener el agente. La clase cuenta además con los siguientes miembros variable configurables:

packet_size_: tamaño de los paquetes a enviar.

burst_time_: media de la distribución exponencial que siguen los periodos ON.

idle_time_: media de la distribución exponencial que siguen los periodos OFF.

rate_: índice de transmisión.

Las siguientes líneas pueden resultar aclaradoras:

```
set mi_generador [new Application/Traffic/Exponential]
$mi_generador set packet_size_ 1KB
$mi_generador set burst_time_ 1s
$mi_generador set idle_time_ 2s
$mi_generador set rate_ 30Kb
```

La clase Application/Traffic/Pareto implementa agentes que un comportamiento similar a los agentes de la clase anteriormente definida. La diferencia está en

que en esta nueva clase, las duraciones de los periodos siguen distribuciones de Pareto, en lugar de distribuciones exponenciales. La clase cuenta igualmente con los métodos `start`, `stop`, `packet_size_`, `burst_time_`, `idle_time_`, y `rate_`. Incluye además el método variable `shape_` que determina el valor del correspondiente parámetro de las distribuciones de Pareto.

La clase Application/Traffic/CBR implementa agentes generadores de tráfico constante. Además de los métodos `start` y `stop`, cuenta con otros miembros variable:

interval_: tiempo entre envío de paquetes.

packet_size_: tamaño de los paquetes a enviar.

maxpkts_: máximo número de paquetes a enviar.

random_: si vale 0 no se inserta ruido, y si vale 1, se inserta ruido (con un intervalo entre 0'5×`interval_` y 1'5×`interval_`).

La clase Application/Traffic/Trace implementa un tipo de agentes que generan tráfico obtenido de un fichero de trazas. Dicho fichero debe ser una instancia de la clase Tracefile. Una vez obtenido un objeto de esta clase, mediante el método `filename` se especifica el nombre del fichero de trazas. Para asociar el fichero al generador de tráfico se debe usar el método `attach-tracefile`. Para aclarar un poco todo esto, se presenta el siguiente ejemplo:

```
set mi_traza [new Tracefile]
$mi_traza filename mi_traza.tr
set mi_generador [new Application/Traffic/Trace]
$mi_generador attach-agent $mi_transporte_udp
$mi_generador attach-tracefile $mi_traza
$mi_generador start
```

Routing

Los mecanismos de routing o encaminamiento implementados en ns deciden el camino que toman los paquetes para llegar a su destino desde el nodo que los originó.

Es posible utilizar dos tipos de routing en ns: unicast (punto a punto) y multicast (punto a multipunto). Por defecto se utiliza un método de routing unicast estático (se considera que la red no va a cambiar su topología durante el transcurso de la simulación). Sobre cómo cambiar la topología (cambiar el estado de nodos y enlaces) durante la ejecución de la simulación ya se ha hablado cuando se trataban los enlaces. Aquí se hablará de cómo configurar la política de encaminamiento de la red, distinguiendo entre los dos tipos, unicast y multicast.

Routing unicast Tres son los protocolos de encaminamiento unicast que pueden ser utilizados en ns: Static (por defecto), Session y Dynamic. No se pueden combinar Session y Static con DV.

Para implementar un protocolo de encaminamiento se utiliza el método `rtproto` del simulador. Dicho método toma como primer parámetro el tipo de routing a utilizar y como segundo parámetro el conjunto de nodos (en forma

de lista) sobre los que regirá el protocolo especificado. En caso de no ser especificado el segundo parámetro, el protocolo de encaminamiento se instala sobre la entera red. En la línea siguiente puede verse un ejemplo típico.

```
$ns rtproto DV
```

Los protocolos Static y Session son muy parecidos. Ambos son una implementación del algoritmo de Djikstra all-pairs SPF. La diferencia está en que Static computa los caminos más cortos entre pares de nodos una sola vez (al principio), pues está pensado para topologías estáticas. Session, por contra, computa las rutas cada vez que un enlace de la red cambia de estado. El método del simulador llamado `compute-routes` permite hacer esto mismo, haciendo que su combinación con el protocolo Static sea , en cierto modo, equivalente al protocolo Session.

Por defecto el coste de todos los nodos (que es tenido en cuenta por los algoritmos) es el mismo: la unidad. Es posible. sin embargo, cambiar esto con el método `cost` de los objetos enlace. El método `cost?` permite consultar el coste de un enlace. La instrucción de la siguiente línea incrementa el coste de un enlace. Puede verse además la forma en la que las expresiones aritméticas son representadas en Tcl.

```
$mi_enlace cost [expr [$mi_enlace cost?] +1]
```

El routing dinámico es una implementación del algoritmo Distance Vector (También conocido como Distributed Bellman-Ford).

Para hacer posible que un nodo soporte el routing multicast, es necesario que la variable `multipath_` de la clase Node tenga valor 1 antes de que se cree el nodo. Por ello, si se va a utilizar routing multicast, la siguiente línea debería preceder a la creación de los nodos de la red:

```
Node set multipath_ 1
```

Cuando conviven varios protocolos de routing en una red y los conjuntos de sobre los que actúan no son disjuntos, puede ser interesante dar preferencia a un protocolo sobre otro. Para ello, hay definido miembro variable en la clase Agent/rtProto (y, en consecuencia, en todas sus subclases). La variable se llama preference.

Es posible obtener diferente información sobre el routing en la red, como, por ejemplo, las tablas de encaminamiento de un router. El método del simulador `get-routelogic` permite obtener un objeto que representa al routing global de la red y cuyos métodos se utilizan para obtener la información de encaminamiento. El método `lookup`, por ejemplo, toma como parámetros dos nodos y devuelve el siguiente nodo en la ruta que parte del primer nodo y finaliza en el segundo.

Cada nodo sobre el que actúe uno o más protocolos de routing unicast dinámico tiene asociado un objeto rtObject. Este objeto actúa como coordinador entre los diferentes protocolos del nodo. Cuenta con métodos interesantes como `dump-routes`, que dado un fichero escribe en él las tablas de routing del nodo, `rtProto?`, que dado un protocolo devuelve un valor booleano que indica si el protocolo está o no instalado en el nodo, `nextHop?`, que dado un nodo destino, devuelve el enlace por el que se encaminan la información que debe alcanzar dicho nodo, y otros métodos que pueden encontrarse en [Fall et al, 1998]. El objeto `rtObject` de un nodo se obtiene mediante el método rtObject del nodo.

Routing Multicast El utilizar protocolos de routing multicast permite la comunicación punto-multipunto en la red. Para que esto sea posible en ns es necesario establecer la variable `EnableMcast_` del simulador con el valor entero 1, antes de que ningún nodo, enlace o agente sea creado. Esto se hace insertando una línea en el script que define la simulación como la que sigue:

```
$ns set EnableMcast_ 1
```

O bien añadiendo los parámetros `-multicast` on cuando se invoca el método `new` para crear la instancia del simulador. La instrucción correspondiente sería entonces:

```
set ns [new Simulator -multicast on]
```

Los nodos una vez creados pueden formar grupos de multicast mediante los métodos `join-group` (de un nodo) `allocaddr` (de la clase Node). Éste permite obtener un nuevo identificador de grupo y aquel, dado un agente de transporte del nodo y el grupo, añade el agente al grupo multicast. Si un agente de transporte tiene como destino el grupo, todos los agentes de transporte que formen parte del grupo recibirán el tráfico generado por aquel. Una manera de establecer el destino de un agente de transporte emisor es establecer el valor de su miembro variable `dst_`. Esto es diferente de utilizar connect, pues básicamente lo que hace éste método es establecer la variable`dst_` de los dos agentes que se conectan. Las siguientes líneas pueden aclarar parte de lo dicho:

```
set mi_audiencia [Node allocaddr]
$nodo_oyente1 join-group $transporte_oyente1 $mi_audiencia
$nodo_oyente2 join-group $transporte_oyente2 $mi_audiencia
..
$nodo_oyenteN join-group $transporte_oyenteN $mi_audiencia
$mi_transporte set dst_ $mi_audiencia
```

También es posible hacer que un miembro de un grupo lo abandone. Para ello se utiliza el método `leave-group` de un nodo. Este método tiene los mismos parámetros que el método `join-group`.

Los protocolos de routing multicast disponibles son Centralized Multicast (CtrMcast), Static Dense Mode (staticDM), dynamic Dense Mode (dynamicDM), Detailed Dense Mode (detailedDM), Protocol Independent Multicast-Dense Mode (pimDM), Shared Tree (ST) y Bidireccional Shared Tree (BST). Utilizando el nombre del protocolo como primer parámetro del método mrtproto del simulado se establece sobre una lista de nodos o sobre la totalidad de los nodos de la red (si no se pasa como segundo argumento la lista de nodos). La línea siguiente instala el protocolo un protocolo multicast en toda la red.

```
$ns mrtproto $mi_protocolo_multicast
```

En algunos casos es necesario configurar el protocolo para que funcione de manera correcta, pues todo esto está en fase experimental (consúltese la sección correspondiente en [Fall et al, 1998]).

3.2.3 Eventos

Como ya ha sido dicho, ns es un simulador de redes dirigido por eventos. Estos eventos son hechos en el tiempo tales como el inicio de la simulación, activación

o desactivación de agentes, reconfiguración de objetos, en general, llamadas a procedimientos.

Los eventos se planifican en el script mediante el método at del simulador. Este método toma como parámetros el instante de tiempo en el que se desea que ocurra el evento, y la instrucción que representa el evento en forma de string (cadena de caracteres). Las siguientes líneas insertadas debidamente en un script, provocan que a partir del comienzo de la simulación se escriba periódicamente por la salida estándar un mensaje con el tiempo actual de simulación. En estas líneas puede verse además cómo se obtiene el tiempo actual de simulación (mediante el método `now`) del simulador y como es posible utilizar variables globales en el ámbito de un procedimiento (mediante el procedimiento `global`).

```
proc notificar_tiempo () {
global ns
set ahora [$ns now]
puts "Tiempo actual: $ahora"
$ns at [expr $ahora+0.5] "notificar_tiempo"
}
$ns at 0.0 "notificar_tiempo"
```

El método `at`, además, devuelve un identificador de evento que puede ser utilizado, entre otras cosas, para cancelar el evento utilizándolo como argumento del método `cancel` del simulador.

Para dar comienzo a la simulación se debe invocar el método `run` del simulador. La simulación termina cuando no quedan eventos por procesar, o con una llamada al método `stop` del simulador o al procedimiento `exit`.

El simulador utiliza por defecto un planificador (scheduler) determinado, aunque es posible utilizar otro de entre los disponibles o implementar uno nuevo. El método `use-scheduler` del simulador obliga al simulador a utilizar el planificador pasado como argumento. Los planificadores disponibles son List, Heap, Calendar y Real-Time.

Los planificadores de ns son secuenciales, esto, es, seleccionan un evento, lo ejecutan hasta que finaliza y sólo entonces están dispuestos a ejecutar otro evento

3.2.4 Trazas y Monitores

Todo lo anterior nos permite definir una simulación y hacerla correr, pero sin saber nada de los resultados de la simulación: qué eventos se han dado, cómo ha sido el tráfico, ...

La herramienta ns provee mecanismos que ofrecen diferentes posibilidades para extraer información sobre el transcurrir de la simulación, para su posterior análisis y proceso. Trazas y monitores son dos tipos de objetos que ns provee para realizar estas tareas

Trazas

Las trazas se descargan en un fichero previamente abierto mediante el procedimiento `open`. Una línea como la siguiente, abre un fichero para escritura.

```
set mi_fichero [open nombre_fichero w]
```

Para forzar el volcado de la información de las trazas en los ficheros correspondientes, se ha de invocar el método `flush-trace` del simulador.

El método `trace-all` del simulador permite hacer traza (de todas los tipos) de todos los enlaces de la red. Toma como único argumento el fichero donde escribir la traza.

Los tipos de traza son `Enque` (llegadas de paquetes), `Deque` (salidas de paquetes) y `Drop` (descartes de paquetes).

El método `create-trace` del simulador toma como argumentos el tipo de traza, un fichero y dos nodos. Crea (y devuelve un handle a) un objeto de la clase Trace que descargará en el fichero la información de monitorización del tipo indicado de las colas de todos los enlaces entre los dos nodos.

El método `drop-trace` del simulador, toma como parámetros dos nodos y un objeto de la clase Trace, eliminando la traza que éste representa entre los dos nodos.

Para realizar una traza de la cola de un enlace debe utilizarse método `trace-queue`, indicando los nodos que forman el enlace y el fichero donde descargar la información de la traza. Si se quiere realizar una traza de la dinámica de un enlace (sus cambios de estado), se debe utilizar el método `trace-dynamics` del enlace pasando como argumentos, la instancia del simulador y el fichero donde se desean descargar la información.

El formato de los ficheros de traza no se explicará. Para saber de él, debe recurrirse a [Fall et al, 1998].

Monitores

Los monitores sirven para monitorizar conjuntos de contadores de llegadas, partidas y descartes de bytes y paquetes, además de soportar estadísticas sobre retardos en la cola. Antes de continuar con su tratamiento, se introducirán dos clases objetos: los objetos de la clase Sample y los objetos de la clase Integrator.

Los objetos de la clase Sample permiten computar algunos estadísticos. Cuentan con los métodos `mean`, `variance`, `count` y `reset`, que respectivamente obtienen la media, la varianza, el número de puntos considerados e inicializan el objeto Sample.

Los objetos Integrator calculan integrales continuas por sumas discretas. Su método `sum` devuelve el valor de la integral.

El método `monitor-queue` del simulador crea un monitor que controla el valor del tamaño de la cola en el enlace entre los nodos indicados como argumentos (se debe prestar atención al sentido del enlace). Devuelve un objeto de la clase Queue Monitor que será utilizado para obtener la información de monitorización del enlace.

Los objetos de la clase Queue Monitor cuentan con varios miembros variable que se describen a continuación:

size_: tamaño de la cola en bytes.

packets_: tamaño de la cola en paquetes.

parrivals_: paquetes recibidos.

barrivals_: bytes recibidos.

pdepartures_: paquetes enviados.

bdepartures_: bytes enviados.

pdrops_: paquetes descartados.

bdrops_: bytes descartados.

bytesInt_: objeto de la clase Integrator para bytes.

pktsInt_: objeto de la clase Integrator para paquetes.

El método `get-bytes-integrator` de un objeto de la clase Queue Monitor, devuelve un objeto de la clase Integrator que puede servir para integrar el tamaño de la cola en bytes, mientras que el método `get-pkts-integrator` devuelve un objeto Integrator que tiene en cuenta el tamaño de la cola en paquetes.

Además, los objetos de la clase Queue Monitor cuentan con el método `get-delay-samples` devuelve un objeto de la clase Sample, que sirve para registrar estadísticas sobre retardos en la cola. El método `set-delay-samples` toma como argumento el objeto de la clase Sample y lo obliga a que registre estadísticas sobre retardos en la cola.

Para inicializar todos los contadores (objetos Integrator y Sample de retardo incluídos) de un monitor puede utilizarse el método `reset` del objeto Queue Monitor.

3.2.5 Miscelánea

Finalizar una simulación.

Es habitual que la finalización de una simulación la realice un procedimiento que se programa para el tiempo de finalización. Resulta de gran utilidad hacerlo así, pues se puede aprovechar para cerrar ficheros, provocar el volcado de trazas, invocar el animador, etc. A continuación se presenta un ejemplo de un fichero de finalización.

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

3.3 Visualización de Resultados

3.3.1 NAM

NAM (Network Animator) es una herramienta que permite visualizar, a partir de ficheros de traza especiales, animaciones de simulaciones.

Los ficheros de traza para nam son generados por ns si en el script que define la simulación es invocado el método `namtrace-all` del simulador. El único argumento a pasar es un fichero previamente creado. Tampoco se debe olvidar que al finalizar la simulación debe provocarse el volcado de la información de traza mediante el método `flush-trace` del simulador.

Para lanzar la herramienta de animación, una vez obtenido el fichero de traza, puede optarse por invocar el programa `nam` desde el intérprete de comandos (shell), o desde el script mediante el procedimiento `exec`. Habitualmente suele incluirse una instrucción como la que sigue, en un procedimiento que se invoca al finalizar la simulación.

```
exec nam nombre_fichero_traza.nam &
```

Esto lanza el animador en paralelo con la ejecución de la simulación, que ya habrá terminado y estará en fase de cierre de ficheros, recogida de algún dato y terminación definitiva.

Al comando `nam` se le pueden pasar parámetros, pero ninguno de los cuales se considera de interés para este capítulo. Si se desea conocerlos, véase [Network Reseach Group, 1997].

La herramienta ofrece un interfaz de usuario bastante intuitivo que permite controlar con comodidad la animación de la simulación.

Son varios los objetos que forman parte de la animación. A continuación se presenta una breve relación de ellos. Deberían sonar familiares si se ha leído los capítulos anteriores de este capítulo.

Los Nodos se pueden representar mediante círculos (por defecto), cuadrados o hexágonos, y no pueden cambiar de forma durante la simulación. También pueden ser representados con distintos colores y cambiar de color durante la animación.

Los enlaces (duplex o simplex) se representas con líneas entre nodos. Pueden tener color y cambiarlo durante la ejecución.

Las colas se representan mediante paquetes apilados. El ángulo entre la horizontal y la pila de paquetes que representa la cola, se puede especificar en el fichero de traza.

Los paquetes se representan mediante un bloque con una flecha en la dirección en la que circulan. Paquetes encolados se representan con pequeños cuadraditos. Paquetes descartados se representan con cuadraditos rodantes que caen por la ventana de animación hasta desaparecer.

Los agentes se representan con su nombre encerrado en un cuadro y unido por una línea al nodo al que están asociados.

La disposición de la topología en pantalla, nodos y enlaces, puede ser definida manualmente o automáticamente. Manualmente se hace definiendo la orientación de los enlaces (el ángulo entre 0 y 2*PI). La herramienta nam trata de ser fiel a estas definiciones al representar la topología en la ventana de animación.

El método automático permite olvidarse de la engorrosa tarea de especificar ángulos y dejar que Nam encuentre una representación visualmente adecuada. Lo hace utilizando un algoritmo que puede ser ajustado en caso de que, tras sucesivos intentos, no se encuentre una estructura aceptable. C_a y C_r son dos constantes que definen la atracción y repulsión entre los nodos (con valor 0'15 por defecto) e *Iterations* es el número de iteraciones que el algoritmo realiza hasta obtener una disposición de los nodos y enlaces. Por defecto su valor es 10. Aumentando este valor puede obtenerse mayor éxito, esto es, representaciones más agradables.

El interfaz de usuario consta de dos ventanas: una para la animación y otra (llamada nam Console) que sirve para abrir nuevas animaciones, obtener una lista de estas y cuenta también con una pequeña ayuda sobre el control de las animaciones. Se hablará sobre aquella, pues está no merece explicaciones.

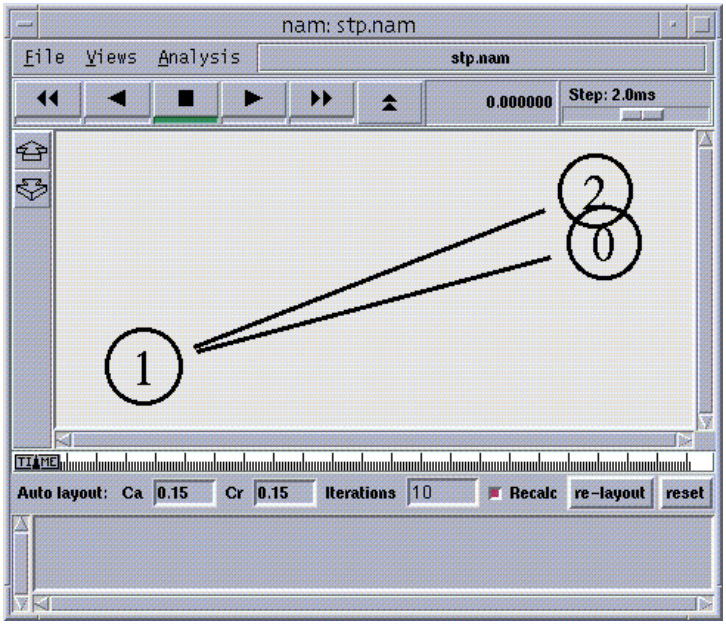

Figure 3.1: Ventana de nam.

La barra de menús cuenta con tres menús llamados **File**, **Views** y **Analysis** cuyas opciones se explican a continuación:

File/Save Layout: permite grabar representación de la topología.

File/Print: permite imprimir la instantánea de la animación que se ve en la pantalla de visualización de la animación.

File/Record Animation: graba la animación completa, esto es, graba cada uno de los frames o fotogramas como una imagen XWD.

File/Auto FastForward: no se ha encontrado información sobre esta opción.

File/Quit: cierra la aplicación.

Views/New View: abre una nueva ventana con la misma animación. Esta nueva ventana sólo tiene dos botones para hacer zoom y uno para cerrarla.

Views/Edit View: abre una ventana que permite reconfigurar la distribución de los nodos en pantalla manualmente.

Views/Highlight LeafTrees: supuestamente debería resaltar los nodos hoja, pero no funciona bien.

Views/Show Monitors: si está activado, permite visualizar la información de los monitores activados. Más adelante se habla sobre monitores.

Views/Show Autolayout: si está activado muestra botones y cajas que permiten configurar y ejecutar el algoritmo de representación de la topología.

Views/Show Annotations: si está activado, permite visualizar anotaciones en una lista en parte inferior de la ventana.

Views/Sync: no se ha encontrado información sobre esta opción.

A la derecha de la barra de menús aparece el nombre del fichero de traza utilizado.

Debajo de la barra de hay una barra de botones intuitiva que permite controlar la reproducción de la animación con los clásicos iconos de los reproductores de música y vídeo. Los botones son, concretamente(de izquierda a derecha):

(«): rebobina la animación 25 veces el tiempo entre fotogramas (más adelante se explica esto) cada vez que se pulsa.

(<): reproduce la animación hacia atrás en el tiempo.

(||): detiene la animación.

(>): reproduce la animación hacia adelante en el tiempo.

(»): avanza la animación 25 veces el tiempo entre fotogramas cada vez que se pulsa.

(∧): es equivalente a la opción quit del menú File. Cierra la aplicación.

A la derecha de este menú de botones hay un indicador del tiempo actual de la animación. Y aún más a su derecha, un indicador del tiempo de salto entre fotogramas con un botón deslizable para cambiar su valor que por defecto es 2'0 ms.

Debajo de todo esto está la ventana de visualización de la animación con dos botones para acercar o alejar la imagen. Pulsando el botón izquierdo sobre la ventana se obtiene información sobre el objeto que está en ese punto en forma de un menú emergente. Si se hace sobre agentes o paquetes aparece la posibilidad de añadir un monitor para controlar su comportamiento y si se hace sobre un enlace aparece una opción para obtener gráficos de ancho de banda y pérdidas en cada dirección del enlace. La experiencia ha demostrado que el uso de monitores y gráficos puede hacer que la aplicación se cierre de manera inesperada.

Debajo de los monitores o de la ventana de visualización de la animación si no está activada la opción de ver monitores, hay una gráfico de progreso del tiempo de la animación con una barra deslizante que permite situar la animación en un punto determinado con gran rapidez. Además, pulsando con el botón izquierdo a derecha o a izquierda de esta barra, se consigue el mismo efecto que con los botones de avance y rebobinado respectivamente.

Si está activada la opción correspondiente a visualizar el menú de representación automática de la topología (**Autolayout**) este aparecerá debajo de la barra de progreso del tiempo. Hay una caja editable por cada uno de los parámetros del algoritmo, un botón para ejecutar el algoritmo y otro para inicializarlo.

En la parte inferior de la ventana aparecerá, si así se ha dispuesto, una lista con anotaciones. Las anotaciones pueden proceder del fichero de trazas o ser añadidas durante la animación seleccionando la opción **Add** del menú que surge si se pulsa con el botón derecho sobre la lista de anotaciones. Aparecen en este menú dos opciones, una para eliminar anotaciones y otra para obtener

Figure 3.2: Ventana de XGraph.

información sobre ellas. Pulsar doblemente con el botón izquierdo sobre una anotación provoca que la animación se posicione si avanzar ni retroceder sobre el punto en el que la animación fue registrada.

3.3.2 XGraph

XGraph es una herramienta de visualización de gráficas que suele incluirse como módulo complementario al simulador ns. Es un programa que lee funciones de una variable y las representa gráficamente. Cada función debe estar representada en un fichero por los elementos del conjunto dominio ordenados y seguidos cada uno por su imagen (elemento del conjunto rango) con un espacio en blanco de separación. Cada par de elementos debe estar en una línea.

Al igual que con la herramienta NAM, XGraph puede ser invocado desde el intérprete de comandos o desde un script. El comando se llama `xgraph` y toma como parámetros los nombres de los ficheros que representan las distintas funciones.

Esta herramienta suele ser utilizada para representar gráficamente información de monitorización de la red como el throughput, tamaños de colas, etc.

En general, se definen procedimientos de recogida de resultados que se programan con el comienzo de la simulación y que se ejecutan periódicamente (ver apartado dedicado a eventos). Dichos procedimientos se encargan de obtener de los objetos que corresponda los datos necesarios (ver apartado dedicado a trazas y monitores), y de escribirlos en los ficheros.

En la figura 3.3.2 puede verse la ventana de XGraph con los datos de throughput obtenidos de una simulación. Como puede comprobarse, esta herramienta es bastante simple y puede servir para una primera visualizació gráfica de resultados. Estudios que requieran un mejor tratamiento de los datos deberían optar por otras herramientas de procesado de datos (hojas de cálculo, paquetes matemáticos, etc.).

Chapter 4

Manual de Funcionamiento Interno de ns

4.1 Introducción

El simulador de redes ns [Network Research Group, 1998] posee la capacidad de ser extendido, esto es, es posible modificar su código para cambiar su funcionamiento o añadir nuevo código para incorporar nuevas funcionalidades al simulador.

Para poder hacer esto, es necesario comprender cuál es la estructura y el funcionamiento interno de la herramienta. En [Fall et al, 1998] puede encontrarse información bastante completa (aunque no totalmente) sobre esto.

En este documento se presentan de manera un poco más general la estructura interna de ns así como los mecanismos que utiliza internamente. Se incluye además un capítulo que trata sobre la creación de un nuevo agente.

NOTA: No debe ser motivo de preocupación la falta de ejemplos a lo largo del capítulo, pues pueden encontrarse ejemplos la sección dedicada a la creación de un nuevo agente en [Fall et al, 1998], [Greis, 1998] y en el mismo código de ns.

4.2 Estructura

El simulador de ns está implementado en C++. Es, pues una aplicación orientada a objetos. Pero cuenta con la particularidad de tener como interfaz con el usuario un intérprete Otcl.

Deben distinguirse entonces dos jerarquías de clases: la escrita en C++ (que es llamada jerarquía compilada) y la del intérprete (que es llamada jerarquía interpretada) escrita en OTcl. Estas dos jerarquías están muy relacionadas entre sí. Aunque en ambas hay clases que no aparecen en la otra, muchas de las clases son comunes a las dos jerarquías y están enlazadas mediante ciertos mecanismos. Esto significa que objetos creados por el usuario a través del intérprete adquieren su *reflejo* en la jerarquía compilada, y objetos instanciados en ésta pueden ser reflejados en el intérprete.

En un siguiente capítulo se habla sobre el intérprete y la manera de enlazarlo con el simulador. Además en otro capítulo se trata la implementación del simulador.

4.3 El Intérprete

Hay varias clases definidas en ficheros sitos bajo el directorio `ns/tcl` que están relacionadas con el intérprete y su acceso desde C++. Se describe a continuación cada una de ellas. Saber de ellas es útil cuando se quiere añadir nuevo código en C++ que se desea hacer accesible desde el intérprete.

4.3.1 Clase Tcl (definida en tcl.cc)

Esta clase permite a quien programa código C++ acceder y comunicarse con el intérprete. Es, pues, una encapsulación del intérprete cuyos métodos permiten realizar las siguientes operaciones.

Existe una instancia de esta clase cuyos métodos son los que permiten realizar las distintas operaciones. Se debe primero obtener una referencia a esta instancia antes de nada. La siguiente línea de código C++ permite hacer esto.

```
Tcl& tcl=Tcl::instance();
```

Existen varios métodos disponibles para invocar procedimientos OTcl a través del intérprete. El funcionamiento de todos ellos es similar; cambia el interfaz, esto es, los parámetros de cada uno. Una cadena con el procedimiento a pasar al intérprete es evaluada por los métodos, que devuelven las constantes `TCL_OK` o `TCL_ERROR` según la ejecución haya sido exitosa o no. A continuación se presentan los prototipos y una breve explicación de cada uno de los métodos.

eval(char* s): invoca el método `TclGlobalEval()` que provoca la evaluación del procedimiento contenido en forma de string como parámetro.

evalc(const char* s): similar al anterior, pero con una cadena de caracteres literal.

eval(): supone que la cadena de caracteres está en un buffer global cuyo nombre es `bp_`. Para obtener una referencia a este buffer se utiliza el método `tcl_buffer(void)` del simulador.

evalf(const char* s,...): permite pasar como parámetro una cadena de caracteres con formato (tipo `printf`).

Para pasar resultados al intérprete se debe usar el método `result(const char* s)` o bien `resultf(const char* fmt,...)`, que utilizas una cadena de caracteres formateada. Para recoger los resultados del intérprete usa del método `char* result()`.

Es posible informar de errores de manera diferente a devolver el valor `TCL_ERROR`. Esta nueva forma, al contrario que la otra, no permite al usuario gestionar el error. El método `error(const char* s)` imprime el error por la salida estándar, lo pasa al intérprete y termina la ejecución con código de error 1.

La clase `Tcl` cuenta además con otros métodos que son utilizados por diferentes mecanismos internos como el mecanismo de *shadowing*, que es el mecanismo que mantiene la correspondencia entre objetos de las jerarquías compilada e interpretada.

Es posible escribir nuevas funciones para pasar al intérprete. Para ello se debe antes conseguir un handle al intérprete mediante el método `interp(void)`.

4.3.2 Clase TclObject (definida en tcl-object.tcl)

Todos los objetos creados a través del intérprete pertenecen a esta clase. En realidad pertenecen a una subclase de ésta, y por herencia, pertenecen a la clase `TclObject`.

Dada una clase C++ con nombre C y subclase de C1, subclase de C2, .., subclase de Cn, subclase de la clase TclObject, el nombre de la clase OTcl correspondiente es

```
Cn\Cn-1\..\C1\C
```

Cada objeto creado a través del intérprete como instancia de una clase de la jerarquía interpretada, tiene su correspondiente objeto instancia de la clase equivalente de jerarquía compilada. Mecanismos internos posteriormente explicados mantienen la correspondencia (*shadowing*) entre ambos objetos.

Existen dos métodos definidos para crear y destruir objetos a través del intérprete. Sus nombre son `new` y `delete`. Cuando el constructor es invocado, el intérprete se encarga de construir el objeto de la jerarquía interpretada y el de la jerarquía compilada y de establecer la correspondencia. Una llamada al destructor deshace esta correspondencia, así como elimina ambos objetos.

Objetos de ambas jerarquías pueden tener miembros variable de uso exclusivo para una de las partes, pero en muchos casos esto no es así. Existe un mecanismo que permite establecer una correspondencia bilateral entre miembros variable de objetos de una y otra jerarquía. Así, el valor de una variable de un objeto en una de las jerarquías tiene siempre el mismo valor que en el objeto correspondiente de la otra jerarquía. Es el constructor C++ de la clase el que debe establecer esta correspondencia mediante los métodos `bind`, `bind_time`, `bind_bw` y `bind_bool`, que dado el nombre de la variable en la parte interpretada en forma de string (cadena de caracteres) y la dirección de la variable en la parte compilada realiza la susodicha operación para variables de tipo entero(o real), de tiempo, de ancho de banda y booleano respectivamente. Las siguientes líneas muestran posibles usos de estos métodos.

```
Mi_Agente::Mi_Agente() {
bind_bool("activado_",&activado_);
bind_bw("throughput_",&throughput_);
bind_time("hora_fin_",&hora_fin_);
bind("mis_bytes_",&mis_bytes_);
};
```

Puede darse el caso de que no exista un constructor del objeto interpretado escrito en OTcl que cree las variables que se desean asociar. En tal caso, lo que si deben existir son instrucciones que creen las variables de la clase. Una manera de hacerlo es mediante el método `set` de la clase.

Las variables pueden tener valores por defecto con los que son inicializadas. Es posible desde el intérprete establecer valores por defecto para cualquier clase y por tanto para sus subclases que no tengan valor por defecto. Basta utilizar el método set de la clase correspondiente utilizando el valor como argumento. La siguiente línea inicializa las variables para las que se creaba una asociación en el ejemplo anterior.

```
MisAgentes/MiAgente set activado_ true
MisAgentes/MiAgente set throughput_ 0.0Kbs
MisAgentes/MiAgente set hora_ true 3.0s
MisAgentes/MiAgente set mis_bytes_ true 1500
```

Para la establecer y gestionar correspondencia entre variable de ambas jerarquías, se utiliza la clase InstVar. Un objeto de esta clase mantiene la relación entre ambas variables. Los objetos TclObject cuentan con un objeto de éstos por variable *atada*. Más adelante se habla sobre esta clase.

Para que los métodos escritos en C++ de un objeto creado en la jerarquía compilada puedan ser invocados, debe haberse definido antes un método `int command(int argc, const char*const* argv)` que se encargue de ejecutar el método que corresponda según los parámetros (ver simulaciones sencillas). El primer parámetro indica el número de argumentos y el segundo es la lista de parámetros en forma de string (se debe recordar que `argv[0]` tiene el valor `"cmd"`. Éste (`cmd`) es el método que implícita o explícitamente invoca el usuario para ejecutar un método del objeto y que se encarga de ejecutar el método C++ `command`. La forma en que el usuario invoca un método C++ desde el intérprete puede ser dos de las siguientes (que son las formas explícita e implícita antes mencionadas):

```
$mi_objeto mi_método $mi_parámetros

o bien

$mi_objeto cmd mi_método $mis_parámetros.
```

El método command debe devolver el valor `TCL_OK` en caso de que todo sea correcto, o `TCL_ERROR` si hay algún error. Si no hay ningún método con el nombre indicado, debe invocarse el método `command` de la clase inmediatamente superior con los mismos argumentos recibidos.

4.3.3 Clase TclClass

Esta clase realiza dos funciones: construir la jerarquía interpretada y mantener la correspondencia entre las jerarquías, y proveer métodos para crear nuevos objetos de la clase TclObject.

Cuando el simulador ns arranca, el constructor de un objeto es ejecutado. Éste llama al constructor TclClass con el nombre de la clase interpretada como argumento. El constructor TclClass almacena dicho nombre en una lista suya de objetos de la clase TclClass.

Durante la inicialización del simulador se ejecuta el método `TclAppInint(void)` que invoca a su vez el método `bind(void)` de la clase TclClass. Esto hace que cada objeto en la lista antes mencionada sea registrado, esto es, se establece la

jerarquía de clases creando las clases necesarias. Finalmente `bind` instancia dos métodos (`create-shadow` y `delete-shadow`) para esta nueva clase que sirven para establecer y borrar la correspondencia.

Nuevas clases derivadas de TclObject y definidas en C++ deben tener su correspondiente clase en la jerarquía interpretada. Para esto es necesario definir una subclase de la clase TclClass de la siguiente manera:

```
static class NuevaClass:public TclClass{
public:
NuevaClass():TclClass("Familia"){}
TclObject* create(int argc, const char*const* argv){
return(new Nueva());
}
} classs_nueva;
```

El método `NuevaClass` es el constructor de la clase. Familia indica el nombre de la clase en la jerarquía interpretada siguiendo las directrices anteriormente indicadas. El método create es el que se invoca cuando el usuario crea un nuevo objeto de esta clase a través del intérprete. Es llamado directamente por el procedimiento `create-shadow` del mecanismo de correspondencia.

4.3.4 Clase TclCommand

Esta clase provee métodos para hacer que procedimientos escritos en C++ puedan ser ejecutados por el intérprete de una manera global, como los procedimientos `ns-version` y `ns-random` (ver fichero `ns/misc.cc`).

Los pasos a seguir son los siguientes. Primero de todo, se debe crear una clase derivada de la clase TclCommand con dos métodos públicos: un constructor `procedimiento()` y otro con prototipo `int command(int argc, const char*const* argv)`. El constructor debe invocar TclCommand pasando como argumento el nombre que el procedimiento tendrá en la parte interpretada. El método `command` debe realizar la acción deseada. argc contiene el número de argumentos y argv una lista de éstos (`argv[0]` es el nombre del comando invocado).

4.3.5 Clase EmbeddedTcl

La clase EmbeddedTcl es utilizada para ejecutar código como parte de la inicialización de ns. Hay varias maneras de hacerlo. Se puede por un lado crear un script con el código que se desea añadir y realizar ciertas operaciones (que no se explicarán), o más sencillamente, añadir código al fichero `tcl-object`.tcl.

4.3.6 Clase Instvar

La clase Instvar es la que permite establecer y mantener la correspondencia entre variables pertenecientes a objetos que existen en ambas jerarquías. Hay cinco subclases de esta clase, una para cada tipo de variable: InstVarReal, InstVarInt, InstVarTime, InstVarBandwidth e InstVarBool. Todas ellas cuentan con un método para establecer la correspondencia: `bind` en el caso de enteros y reales, y `bindtime`, `bindbw` y `bindbool` en el caso de variables de tiempo, ancho de banda y booleanos respectivamente. Los parámetros de este método son el nombre de la variable en la parte interpretada y la dirección de la variable en

la parte compilada. Normalmente es invocado desde el constructor de la clase correspondiente. Al hacerlo se crea una variable de la subclase de InstVar que mantiene la correspondencia entre las variables del objeto durante su vida.

4.4 El Simulador

Esta sección trata sobre el funcionamiento interno del simulador y algunos de los objetos que forman parte de la simulación.

4.4.1 El Simulador

El simulador es el elemento central de las simulaciones. Toda simulación debe comenzar instanciando un objeto de la clase Simulator que contiene los métodos necesarios para definir la topología y la simulación.

Cuando este objeto es creado se realizan ciertas operaciones de inicialización: se inicializa el formato de los paquetes, se crea un scheduler (planificador) y se crea un agente nulo que servirá como sumidero de paquetes inservibles.

El scheduler (objeto de la clase Scheduler definida en `scheduler.cc` y `scheduler.h`) por defecto es uno llamado List Scheduler (clase Scheduler/List), pero hay otros implementados: heap scheduler (clase Scheduler/Heap), Calendar Queue Scheduler (clase Scheduler/Calendar) y Real Time Scheduler (Scheduler/RealTime).

La función del planificador es planificar el siguiente evento a ejecutar de entre los que están planificados. Actúa seleccionando un evento, ejecutándolo y cuando termina, eligiendo otro para ejecutar. La ejecución de eventos es siempre secuencial; no hay paralelismo ni concurrencia.

Estos eventos son creados por el usuario través del intérprete. La definición de la clase Event que usa el simulador internamente para representar los eventos puede encontrarse en el fichero `ns/scheduler.h`. Un evento se define básicamente por el tiempo en que debe ocurrir y las acciones que debe realizar. Los paquetes, por ejemplo, son implementados internamente como eventos.

4.4.2 Nodos

Los nodos están implementados como una clase exclusiva de la parte interpretada, y por lo tanto su código está escrito en OTcl (concretamente en el fichero `ns/tcl/lib/nodes.tcl`).

Un nodo tiene varios componentes, algunos de los cuales son objetos de la clase TclObject y tienen vida, por tanto, en ambas partes (compilada e interpretada).

El primer elemento que recibe un paquete en un nodo es uno llamado `entry_`. Éste apunta al objeto que debe tratar los paquetes. En el caso de nodos exclusivamente unicast, este objeto es de la clase Classifier y se almacena bajo el nombre `classifier_`, mientras que en nodos unicast es un objeto `switch_` que decide si se ha de enviar el paquete al clasificador unicast o al multicast.

Varios son los tipos de clasificadores utilizados. Su función básica es analizar los campos de los paquetes y decidir qué hacer con ellos. Mantienen una tabla de salidas para los paquetes. En el caso de nodos unicast, estas salidas son los enlaces y un clasificador de puertos almacenado bajo el nombre `dmux_` que se encargará de distinguir el agente al que debe pasar el paquete en función del

número de puerto. En nodos multicast, se utiliza además un clasificador de la subclase Replicator que no clasifica los paquetes sino que los envía a todos los elementos de su tabla de salidas.

Los elementos de la tabla del clasificador del puerto apuntan a una lista de agentes llamada `agents_` que forma parte del objeto nodo como miembro variable. Los agentes se añaden a esta lista mediante el método `attach` (invocado a su vez por el método `attach-agent`) que además se encarga de asignar un nuevo número de puerto al agente.

Las tablas además se organizan con los métodos `add-rout` y `add-routes` que dado el identificador del destino y la(s) entrada(s) en la tabla, establece ruta(s). El método `delete-routes` borra las rutas.

Además un nodo puede contar con otros objetos incorporados por otros mecanismos. Un ejemplo es el miembro variable `rtObject_` que obtienen los nodos si se activan mecanismos de routing dinámico.

4.4.3 Enlaces

En ns hay varios tipos de enlaces implementados. El enlace de la clase SimpleLink es el más comúnmente utilizado y su funcionamiento es el que se describirá aquí. Su implementación está únicamente en Otcl, pero cuenta con objetos de la clase TclObject que pueden estar implementados en C++.

El usuario crea un enlace entre dos nodos mediante el método `simplex-link` (o `duplex-link`, que crea dos enlaces de la clase SimpleLink) indicando además de los nodos, el ancho de banda, el retardo y el tipo de cola. Internamente el enlace tiene, por lo menos, cinco objetos almacenados como miembros variable bajo los nombres `head_`, `queue_`, `link_`, `ttl_` y `drophead_`. Tal como ocurría con los nodos, puede haber más objetos incorporados por otros mecanismos. A continuación se describen los cinco objetos básicos:

head_: apunta al primer objeto del enlace que procesará los paquetes.

queue_: es una referencia a la cola del enlace.

link_: es una referencia al elemento que modela el enlace con el ancho de banda y el retardo.

ttl_: es una referencia al objeto que manipula el ttl de cada paquete.

drophead_: referencia a la cabeza del objeto que procesa los descartes de paquetes.

El retardo y ancho de banda del enlace puede ser modelados con distintos conectores (objetos de la clase Connector), pero el tipo SimpleLink utiliza la clase C++ LinkDelay que simula ancho de banda y retardo constantes. Lo que hace un objeto Connector es recibir un paquete, procesarlo y enviarlo a su destino: que puede ser un nodo vecino o un lugar de descarte de paquetes).

La clase LinkDelay está implementada en los ficheros `ns/delay.h` y `ns/delay.cc`. El retardo se simula programando el evento de recepción en un instante de tiempo obtenido por la siguiente fórmula:

$T = h + (s/b) + d$

Donde h es la hora actual, s es el tamaño del paquete en bits, b es el ancho de banda en bits por segundo y d es el retardo de propagación.

Las colas pueden ser de diferentes tipos que están implementados bajo subclases de la clase Queue. Las diferencias entre los diferentes tipos están en la política de planificación de la cola, esto es, decidir qué paquetes son servidos y cuáles son descartados, y en la gestión de la ocupación de la cola.

El retardo en la cola es simulado bloqueándola y desbloqueándola con eventos programados. Las colas cuentan además de con un origen y un destino, que son los objetos de la clase Connector que se encargan de bloquear y desbloquear la cola, y con un destino de descartes donde enviar los paquetes que se decide sean descartados (por encontrarse la cola totalmente llena).

La clase Queue está definida en el fichero `ns/queue.h` como derivada de la clase Connector e ideada para que sirva de base a las subclases correspondientes a los diferentes tipos de cola.

4.4.4 Agentes

Los agentes representan entidades que generan o consumen paquetes. Pueden representar entidades de diferentes niveles, implementando protocolos , aplicaciones e incluso módulos de routing.

La clase base Agent está definida parte en C++ (ficheros `ns/agent.h` y `ns/agent.cc`) y parte en Otcl (fichero `ns/tcl/lib/ns-agent.tcl`). Esta clase sirve de base para otras clases de agente como la clase TCPAgent (llamada Agent/TCP en Otcl) o la clase Application, proveyendo variables y métodos generales.

El estado de un agente viene dado por sus diferentes miembros variable, que se describen a continuación:

addr_: dirección del agente. La dirección la forman 16 bits. Los ocho primeros son el nodo y los ocho siguientes, el puerto. En caso de haber expandido el número máximo de nodos a utilizar, los bits correspondientes al nodo se elevan hasta 22.

dst_: dirección del agente destino.

size_: tamaño de los paquetes que envía.

type_: tipo de paquetes que envía el agente (los tipos están definidos en el fichero `ns/packet.h`).

fid_: identificador de flujo IP.

prio_: campo de prioridad de IP.

flags_: flags de los paquetes.

defttl_: valor por defecto de TTL (Time To Live) en IP.

Los valores por defecto de estos parámetros están definidos en el fichero `ns/tcl/lib/ns-default.tcl`. Este fichero contiene además otras instrucciones que establecen valores por defecto de variables de muchas otras clases.

La clase Agent define además un par de métodos que se espera no sean especializados por las subclases y que sirven para crear paquetes a enviar. Sus prototipos son `Packet* allocpkt(void)` y `Packet *allockpkt(int)` (crea el paquete con carga de bytes).

Otros dos métodos, en cambio, se definen en espera de que sean especializados. Sus prototipos son `void timeout(int timeout_number)` que implementa un temporizador y `void recv(Packet*, Handler*)` que es el método que será invocado por entidades por debajo del agente (nodos u otros agentes que representen entidades de nivel inferior) para indicar la recepción de paquetes.

Existen diferentes protocolos implementados en ns. Todos ellos se definen en clases derivadas de la clase Agent. Bajo la clase Agent/TCP hay numerosas subclases que implementan diferentes versiones del protocolo de transporte orientado a conexión TCP, mientras que la clase Agent/UDP implementa el protocolo de transporte no orientado UDP. Además otras clases son Agent/LossMonitor (agentes sumidero especial para recoger estadísticas de pérdidas), Agent/rtProtoDV (agente de routing DV),...

Los agentes de transporte (TCP,UDP y SRM) están diseñados para tener por encima un agente de aplicación que genere y/o consuma tráfico. Para ello ofrecen un API que es el interfaz que utilizan las aplicaciones para comunicarse con ellos. Sobre este API se hablará cuando se hable de agentes de aplicación.

Los agentes de routing, en cambio, generan su propio tráfico y no llevan asociados ningún otro agente.

Se ofrece una clase abstracta en ns que pretende servir de base para clases que implementen agentes de aplicación concretos. Esta clase se llama Application y como derivadas de ellas se incluyen también en ns Application/FTP, Application/Telnet y diversos generadores de tráfico que siguen distribuciones aleatorias y que están implementados como subclases de la clase Application/Traffic (Application/Traffic/CBR, Application/Traffic/Uniform, Application/Traffic/Pareto y Application/Traffic/Exponential).

Estas últimas están pensadas para utilizar UDP como protocolo de transporte, mientras que las otras deberían usarse sobre TCP.

Sobre como asociar estos agentes de aplicación con sus correspondientes agentes de transporte ya se ha hablado en el capítulo 3. Aquí se hablará sobre cómo los agentes interactúan entre sí.

Agentes de transporte y de aplicación mantienen un puntero al agente de transporte de nivel inferior o superior según corresponda, que se crea con el método `attach` y se utiliza para la comunicación.

Los agentes de transporte se comunican con los de aplicación mediante el interfaz que define la clase Application, destacando los métodos `recv(int nbytes)`, que indica al agente la recepción de `nbytes número de bytes`, y `resume()` que indica que todo lo que deseaba enviar ha sido enviado (no asegura el éxito).

Mientras tanto, los agentes de aplicación utilizan el llamado API de transporte. Este API consiste en los siguientes métodos:

send(int nbytes): indica al agente de transporte que debe enviar nbytes bytes. Si este valor es -1, se envían datos indefinidamente.

sendmsg(int nbytes, const char*=0): similar a send, pero con un argumento de flags que puede servir, si toma el valor "MSG-EOF" para indicar que no se enviará nada más.

close(): cerrar la conexión.

listen(): hacer que el agente pase a estar en espera de peticiones de apertura de conexión (sólo para objetos de la clase Agent/TCP/FullTCP que implementa TCP bidireccional).

set_pktype(int pkttype): obligar al agente de transporte a utilizar un tipo de paquete determinado.

Todo esto no permite el envío real de datos, sino que se simula envío de datos, expresando el espacio que éstos ocuparían. Para permitir el envío real de datos hay una clase llamada TcpApp que ofrece la posibilidad de transferir datos reales mediante los métodos `send(int nbytes, char *str)` y `process(int size, char *data)`.

El método `connect(TcpApp *TcpApp)` debe ser usado para comunicar agentes de este tipo entre sí, si van a transferirse datos reales. Hasta el momento esto está en fase experimental bastante indocumentada y sólo la aplicación Web implementada bajo las clases Http, Http/Client, Http/Server y Http/Cache ha sido implementada utilizado intercambio de datos. El resto de las aplicaciones no transfiere datos.

4.4.5 Routing

En ns hay implementados dos tipos básicos de routing: unicast y multicast. En apartados separados se describe de manera general el funcionamiento interno de cada uno de estos tipos de routing.

Routing Unicast

El routing unicast permite la comunicación punto a punto entre agentes situados en distintos nodos de la red. En ns han sido implementados tres protocolos de routing unicast: Static, Session y DV (Dynamic Vector). La implementación de estos protocolos la forman no sólo clases específicas, sino también extensiones a otras clases ya existentes.

Las clases específicamente implementadas en para este tipo de routing son RouteLogic, rtObject, rtPeer y Agent/rtProto.

La clase RouteLogic define métodos para configurar el routing y para consultar información de encaminamiento.

La clase rtObject define una clase de objetos que cuando se utiliza routing dinámico, actúa de coordinador entre los diferentes protocolos de routing que operan en un nodo, actuando de acuerdo a cambios en las tablas de éstos y/o en la topología.

La clase rtPeer es usada por loa agentes de protocolo de routing para almacenar las direcciones de agentes destino así como la métrica y la preferencia de cada ruta hacia ese punto.

La clase Agent/rtProto sirve de clase base para las clases que implementan agentes de protocolo de routing.

Las clases que son extendidas para el routing son las clases Simulator, Node, Link y Classifier.

Routing Multicast

El routing multicast permite la comunicación punto a multipunto. De igual manera que con el routing unicast, el routing multicast ha sido implementado

definiendo varias clases y extendiendo otras (Simulator, Node, Classifier y Replicator).

La clase MCastProtocol es la clase que sirve de base para la definición de clases que implementan protocolos de routing multicast.

4.4.6 Otras cosas

Existen algunos otros elementos implementados en ns que se han considerado secundarios para la realización de este documento y sobre los cuales puede encontrase una mejor descripción que la que a continuación se presenta , que no pasa de ser una breve presentación.

Temporizadores

La clase abstracta TimerHandler sirve de clase base par la implementación de clases concretas de temporizadores, útiles en la implementación de agentes. Ofrece métodos para planificar, cancelar y controlar el estado de estos temporizadores, sí como actuar a su finalización.

Paquetes

Bajo la clase Packet están definidos los tipos de paquetes que sirven como unidad de intercambio de información entre agentes. Esta clase abstracta ofrece algunos métodos para crear paquetes, reservar espacio para el cuerpo, liberarlo, y variables que definen el formato de los paquetes.

ErrorModel

La clase ErrorModel implementa mecanismo para simular errores de nivel de enlace en la red.

LAN

En ns hay definidos varias clases que permiten definir una topología de red de área local. Se han implementado entidades de nivel físico, MAC y LL para simular este tipo de redes.

Soporte matemático

Las diferentes subclases de la clase RandomVariable y la clase RNG, implementan variables aleatorias que pueden resultar de utilidad para modelar situaciones del mundo real. Además la clase Intregrator implementa una clase de objetos que permiten realizar el cálculo de integrales por sumas discretas.

Trace and Monitoring

Dos mecanismos han sido implementados en ns para seguir el comportamiento de los elementos de la simulación: trazas y monitores, implementados en subclase de las clases Trace, SnoopQueue y QueueMonitor.

Dynamics

Las clases rtModel y rtQueue, así como extensiones a otras clases (DynamicLink, Link y SimpleLink) han sido incorporados a ns para hacer posible la dinámica de la topología, esto es, que nodos y enlaces puedan estar variablemente activos o inactivos.

Session-level Packet Distribution

Es ésta una extensión a ns para permitir simular redes multicast con gran número de nodos

Emulation

La herramienta ns posee la capacidad de ser introducida en una red real. Para ello se ha implementado complejos mecanismos.

4.5 Creación de un nuevo Agente

Como ya ha sido dicho, una de las características de ns es que es extensible, es decir, pueden realizarse modificaciones o añadidos para adecuarla a las situaciones que se deseen simular. Una de las extensiones más atractivas e interesantes es la creación de un nuevo agente. A continuación se presenta una explicación de lo que se ha de hacer para implementar un nuevo agente. Esta explicación es fundamentalmente teórica. Para llegar a comprenderla totalmente, quizá sea conveniente consultar los ejemplos presentados en [Fall et al, 1998], [Greis, 1998] y el capítulo 5.

Son diferentes los pasos a seguir para crear un nuevo agente: decidir la posición de la clase en la jerarquía de clases, definir la nueva clase, implementar los métodos necesarios, implementar los mecanismos de enlace con el intérprete y realizar los cambios necesarios en el código de ns.

4.5.1 Decidir su posición en la jerarquía

En principio uno puede decidir que la clase de agente que desea definir no se deriva de ninguna otra clase, pero esto sería un absurdo, pues existen clases ya definidas e ideadas para servir de base a nuevas clases de agente.

La clase base de la que derivan todos los agentes es la clase Agent. Ésta debería ser la elegida como padre de la nueva clase si el agente que se desea crear implementa entidades de nivel inferior al nivel de transporte o si las siguientes propuestas no parecen adecuadas.

Las clases Agent/UDP y Agent/TCP deberían ser las elegidas si se desea implementar protocolos de transporte UDP o TCP respectivamente. Si en concreto se quiere implementar una especialización de una de las múltiples versiones de TCP ya implementadas en ns, deberá elegirse la subclase adecuada.

La clase Application es una clase abstracta definida para servir de base a nuevas clases de aplicación, es decir, de nivel inmediatamente superior al nivel de transporte. Esta clase no contempla el envío real de datos. Si esto es algo que se desea forme parte del nuevo agente, se debería utilizar la clase TcpApp para derivar de ella la nueva clase.

Por último, nuevos agentes de protocolo deberían ser definidos en clases derivadas de las clases rtProto o McastProtocol según el nuevo protocolo de encaminamiento sea unicast o multicast.

4.5.2 Definir la nueva clase

Una vez decidida la clase de la que derivará la nueva clase, ésta ha de ser definida, preferiblemente en un fichero header propio, bajo el directorio de ns donde se encuentran definidas el resto de las clases. Se deberán incluir al comienzo de éste las cláusulas `include` necesarias.

La clase debe ser declarada como derivada pública de la elegida para tal efecto y entre sus métodos no deben faltar el constructor y uno con el prototipo `int command(int argc, const char*const argv)` que sirve para invocar comandos del objeto a través del intérprete y sobre el que se hablará más adelante. Estos dos deben ser declarados en la parte pública. Más miembros método y variable deberían ser declarados. Cuáles y en que parte (pública, protegida o privada), es algo que depende del tipo de agente a implementar.

Una clase abstracta proporciona los miembros imprescindibles (a veces sin implementar), pero pueden ser necesarios más. Incluso puede ser necesario crear otras clases para utilizar objetos de éstas como parte del nuevo agente. Este es el caso de los temporizadores en agentes de transporte.

4.5.3 Implementar los métodos necesarios

En un fichero separado se deben implementar los métodos de la clase del nuevo agente y de otras clases auxiliares si ha sido necesaria su inclusión. Este fichero ha de estar en el mismo directorio que el que contiene las declaraciones y tener por nombre el mismo pero con extensión `cc` en lugar de `h`. Como cláusula `include` debe tener, como mínimo, una que incluya al fichero de declaraciones.

A continuación deben ser implementados los métodos que definen el comportamiento del agente. Especial mención merecen el constructor y el método `command`, pues están relacionados con el enlace con el intérprete. Por ello serán tratados un poco más adelante.

Sobre el resto de los métodos nos se dirá nada más aquí, pues sólo quien desea implementar un nuevo agente sabe cuáles son y qué hacen. Si se desea alguna idea más clara, véanse los ejemplos de [Fall et al, 1998], [Greis, 1998] y el capítulo 5 y la implementación de agentes nativos de ns.

4.5.4 Implementar los mecanismos de enlace con el intérprete

Puesto que la creación de agentes, por parte del usuario es realizada a través del intérprete, se hace necesario definir algunas funciones que realicen las acciones necesarias para permitir el enlace entre ambas jerarquías, compilada e interpretada.

Primero de todo debe crearse una instancia de una clase nueva derivada de la clase TclClass. Esto permite crear objetos de la nueva clase a través del intérprete. Por simplicidad se presenta un prototipo del código que debe ser incluido en el fichero que implementa la clase (el fichero con extensión `cc`).

```
static class NuevaClaseClass:public TclClass{
public:
NuevaClaseClass:TclClass("Clase_en_Otcl"){}
TclObject* create(int argc, const char*const* argv){
return new NuevaClase());
}
}class_nuevaclase;
```

Téngase en cuenta que `Clase_en_Otcl` es el nombre de la clase en Otcl. Para saber debieran ser las relaciones entre los nombres de clases equivalentes en las jerarquías compilada e interpretada, véanse capítulos anteriores.

Otra de las cosas que deben realizarse para hacer posible el enlace con el intérprete es añadir las instrucciones necesarias en el constructor para establecer la correspondencia entre los miembros variable del objeto compilado o del objeto Otcl. Esto se hace por medio de los métodos `bind`, `bind_time`, `bind_bw` y `bind_bool`, según la variable sea de tipo entero o real, de tiempo, de ancho de banda o booleano respectivamente, y tomando como argumento primero, una cadena de caracteres con el nombre del miembro variable en la parte interpretada y segundo, la referencia al miembro variable de la parte compilada. Ambos nombres deberían ser el mismo y terminar con un carácter '_'. Esto último es sólo por convenio y no se cumple en todo el código de ns.

Por último, el método `command`, que permitirá al usuario ejecutar métodos C++ del objeto, debe ser implementado. Este método tiene como argumentos el número de parámetros y los parámetro pasados por el usuario en forma de cadena de astringes.

El método `command` define el interfaz que se desea ofrecer al usuario, por lo tanto debe reconocer el comando que introduce el usuario y realizar las acciones consecuentes. Quiere esto decir que los comandos que se ofrecen al usuario mediante este método no tienen por qué coincidir con los métodos del nuevo agente. Cuáles serán los comandos que se aceptarán y que acciones realizarán éstos es decisión del creador del nuevo agente. Sólo se llamará la atención sobre algunas cuestiones.

Primero, que el primer argumento de la lista de argumentos (`argv[0]`) contiene el valor `"cmd"` y el segundo (`argv[1]`)) el comando del usuario. El resto son los argumentos del comando.

El valor devuelto por esta función debe ser la constante `TCL_OK` si todo ha ido bien, `TCL_ERROR` si se ha detectado algún error o una invocación al método `command` de la clase inmediatamente superior en la jerarquía con los mismos argumentos recibidos, si no se reconoce el comando. Por ejemplo, si la clase inmediatamente superior es la clase Agent se debe añadir la siguiente línea donde corresponda:

```
return (Agent::command(argc, argv));
```

Siendo argc y argv los nombres de los parámetros recibidos por el método `command`.

4.5.5 Añadir cambios en el código de ns

Algunos cambios en el código de ns son obligados y otros no. Aunque en la documentación en la que se basa este documento [Fall et al, 1998] no se dice, la

experiencia ha demostrado que el valor por defecto de las variables que existen en ambas a jerarquías, debe ser definido en el fichero ns/tcl/lib/ns-default.tcl, pues de lo contrario se obtienen mensajes de error al ejecutar simulaciones que incluyan el uso del nuevo agente y no al compilar de nuevo ns, lo que puede resultar confuso. La razón está en que el asociar las variables compilada e interpretada no provoca la creación de la variable interpretada en caso de no existir. Por lo tanto, las variables interpretadas deben ser creadas antes de crear la asociación, y una forma de hacerlo es asignarles valores por defecto.

Definir el valor por defecto para variables de una clase se hace añadiendo líneas en el fichero mencionado, del tipo

```
NuevaClase set mi_variable_ 0
```

Donde NuevaClase es el nombre de la clase en la parte interpretada. Otros cambios en el código de ns pueden ser necesarios si, por ejemplo, el nuevo agente utiliza paquetes de un tipo que no forma parte de los provistos por ns. Para saber que hacer un tal caso, véase [Fall et al, 1998] y [Greis, 1998].

Además debe modificarse el fichero make que corresponda (en general, ns/Makefile), incluyendo el nombre del fichero objeto a obtener para una más cómoda compilación.

Chapter 5

Simulaciones Sencillas

5.1 Introducción

Muchos son los ejemplos incluídos en la distribución del simulador de redes ns [Network Research Group, 1998]. Presentan situaciones diversas que ayudan a comprender el funcionamiento de ns y, en muchos casos, pueden servir de patrones para desarrollar nuevas simulaciones. En la web de ns hay una página[1] que contiene una breve descripción de estos ejemplos así como la localización de los ficheros correspondientes a partir del directorio donde haya sido instalada la herramienta.

Además, el tutorial de ns [Greis, 1998], está basado en ejemplos, cuya complejidad aumenta gradualmente según se avanza en el seguimiento de este tutorial, lo que hace muy suave el arranque para nuevos usuarios.

En este capítulo se presentan varias simulaciones, que pretenden servir de complemento a los capítulos 4 y 3. Cubren los aspectos más importantes presentados en ambos capítulos.

Cada una de las simulaciones se describe en un capítulo de los que siguen a éste, incluyendo la intención de tal ejemplo, su diseño, su implementación (su código con los convenientes comentarios) y una explicación de los resultados (el transcurrir de la simulación, gráficos, etc.). Es de esperar que quien lea este capítulo pruebe cada simulación mientras sigue el correspondiente capítulo.

5.2 Simulación UNO (básica)

5.2.1 Intención

La intención de esta simulación es mostrar cómo definir una topología básica, en la que un par de agentes generadores de tráfico se comunican con dos sumideros utilizando UDP sobre una red en la que nodos y enlaces permanecen estáticos. Además se incluyen instrucciones específicas para el visualizador de animaciones NAM.

Figure 5.1: Animación de la simulación UNO

Figure 5.2: Animación de la simulación DOS

5.2.2 Diseño

La red tiene forma de estrella de tres puntas. En una de estas puntas (nodo 0) hay un sumidero general (clase Agent/Null), en otra (nodo 1) hay un generador de tráfico exponencial (clase Application/Traffic/Exponential) sobre un agente de transporte UDP (clase Agent/UDP), y en la última (nodo 2) hay un sumidero general (Agent/Null) y también un generador de tráfico constante (Application/Traffic/Constant) sobre UDP (clase Agent/UDP).

El sumidero del nodo 0 consume el tráfico del productor situado en el nodo 3, y el sumidero sito en este mismo nodo hace lo propio con el tráfico del productor que está en el nodo 1.

El nodo central (nodo 2) actúa como router y en él se cruzan los dos tráficos. Los enlaces son todos dúplex, pero con diferentes valores de retardo, ancho de banda y tipo de cola. El routing de la red es estático.

5.2.3 Implementación

El código que implementa la simulación puede encontrarse en el apartado dedicado al fichero `uno.tcl` dentro del capítulo 9.

5.2.4 Resultados

Tras unos instantes en los que no sucede nada, el agente generador de tráfico comienza su transmisión. Debido a la lentitud del enlace entre los nodos 0 y 2, el nodo 2 enseguida se satura y comienza a descartar paquetes. Puede verse la cola aumentando de tamaño hasta llegar a su máximo, instante en que comienzan a perderse paquetes.

Mientras tanto, el otro agente comienza su transmisión que transcurre con fluidez por la rapidez de los enlaces que forman parte de su ruta.

5.3 Simulación DOS (Dínamica)

5.3.1 Intención

La intención de esta simulación es mostrar un ejemplo en el que un enlace falla y cómo el uso de routing dinámico permite superar este fallo buscando una ruta alternativa. Se utiliza TCP como protocolo de transporte para la aplicación FTP que genera el tráfico.

5.3.2 Diseño

La red tiene la forma de un triángulo (nodos 1, 2 y 3) en el que de dos de sus vértices cuelgan dos nodos hoja (nodos 0 y 4). En estos dos nodos hay un

[1] `http://www-mash.CS.Berkeley.EDU/ns/ns-tests.html`

Figure 5.3: Animación de la simulación TRES

sumidero TCP (clase Agent/TCPSink) y una aplicación FTP (clase Application/FTP) sobre TCP (clase Agent/TCP) respectivamente. Así, existen dos caminos posibles para la comunicación entre los agentes de ambos nodos.

Todos los enlaces tienen los mismos valores de retardo, ancho de banda, retardo y tipo de cola, siendo además todos los enlaces dúplex. El enlace entre los nodos 1 y 2 cuenta con la particularidad de que durante un determinado intervalo de tiempo estará inactivo.

El protocolo de routing utilizado es DV (Dynamic Vector).

5.3.3 Implementación

El código que implementa la simulación puede encontrarse en el apartado dedicado al fichero `dos.tcl` dentro del capítulo 9.

5.3.4 Resultados

Al comienzo de la simulación puede verse como los nodos intercambian paquetes pequeñitos. Esto forma parte del protocolo de routing DV.

A partir del instante 0'5 (segundos), el agente FTP sito en el nodo 4 comienza su transmisión. En algunos momentos, puede verse como el tamaño de la cola del enlace 4-3 (enlace entre los nodos 4 y 3) crece. En estos primeros momentos la ruta que siguen los paquetes es 4-3-1-0, pero en el instante 1'0 el enlace 3-1 se rompe (cambia a color rojo), perdiéndose algunos de los paquetes que se estaban enviando.

Casi inmediatamente el protocolo de encaminamiento actúa y los paquetes siguen una nueva ruta (4-3-2-1-0), hasta que el enlace caído se restablece y el camino seguido por los paquetes vuelve a ser el original.

5.4 Simulación TRES (multicast)

5.4.1 Intención

La intención de esta simulación es mostrar el funcionamiento del routing multicast (punto a multipunto) en ns y el uso de monitores para generar datos que posteriormente puedan ser procesados por la herramienta XGraph.

5.4.2 Diseño

La red tiene la forma de una estrella de cuatro puntas. El nodo central hace las veces de router y, mientras uno de los nodos exteriores cumple funciones de emisor, el resto hacen de receptores. En el nodo emisor, hay un generador de tráfico constante (clase Application/Traffic/Constant) que envía datos al grupo multicast del que forma parte junto con los nodos receptores.

Poco a poco los miembros del grupo multicast irán dejando de formar parte de él.

Se utilizará un monitor para obtener periódicamente información sobre el flujo entrante y saliente del router.

Figure 5.4: Gráfica del throughput en la simulación TRES.

Figure 5.5: Animación de la simulación CUATRO

5.4.3 Implementación

El código que implementa la simulación puede encontrarse en el apartado dedicado al fichero `tres.tcl` dentro del capítulo 9.

5.4.4 Resultados

La animación tiene un transcurrir muy sencillo. Al principio los paquetes emitidos por el emisor llegan a los tres miembros del grupo. Cada miembro de este grupo lo abandona en un cierto instante de tiempo, y cuando lo hace deja de recibir paquetes.

De esta manera el throughput (medido en bits por segundo) entrante en el nodo router se mantiene constante mientras el emisor transmite y hay receptores, esto es, el grupo no está falto de miembros.

El throughput saliente es va descendiendo a medida que los miembros del grupo lo van abandonando. En el instante en que el grupo está vacío, el throughput entrante y el saliente son ambos nulos.

Los valores del throughput entrante y saliente a través del tiempo han sido recogidos en un fichero y se ha obtenido la siguiente gráfica con la herramienta XGraph.

5.5 Simulación CUATRO (extensión)

5.5.1 Intención

La intención de esta simulación es mostrar la capacidad de ns de ser extendido, presentando una nueva aplicación llamada PingPong.

5.5.2 Diseño

La topología (la forma de la red) es muy simple: dos nodos unidos entre sí por un enlace. En ambos nodos existe una aplicación PingPong (clase Application/PingPong) sobre un agente TCP bidireccional (clase Agent/TCP/FullTcp). Ambas aplicaciones forman parte de una misma conexión.

La aplicación PingPong se implementa como subclase de la clase Application. Cuando se arranca una aplicación de esta clase (método `start`), está envía una cantidad de datos. Cada vez que una aplicación reciba cierta cantidad de datos (cuando se invoque su método `recv`), los reenviará a su origen.

Así dos aplicaciones PingPong conectadas, intercambiarán continuamente datos si una de las dos da comienzo a su ejecución, y no terminarán con este intercambio hasta que una de las dos se detenga.

Se hace necesario el uso de TCP bidireccional porque los agentes de transporte que soportan la comunicación deben servir a entidades de aplicación que son productoras y consumidoras al mismo tiempo.

5.5.3 Implementación

El código que implementa la simulación puede encontrarse en los apartados dedicados a los ficheros `ns-2/pingpong.h`, `ns-2/pingpong.cc`, `ns/tcl/lib/ns-default.tcl`, `cuatro.tcl` dentro del capítulo 9.

5.5.4 Resultados

Tras lanzar la operación puede verse como un paquete es enviado en el instante 0'6 desde el nodo 0 al nodo 1. Es una solicitud de apertura de conexión que es contestada por el agente de transporte del nodo1 con un paquete de tamaño similar.

Una vez establecida la conexión, la aplicación PingPong del nodo 0 envía un mensaje de 512 bytes (que se mete en un simple paquete) a la del nodo1, está lo recoge y lo vuelve enviar de vuelta al nodo 0. La aplicación de este nodo recoge el paquete y lo vuelve a enviar y así están las dos aplicaciones hasta el instante en que se paran. Puede observarse que al primer paquete de 512 bytes enviado le acompaña un pequeño paquete asociado a TCP; es un reconocimiento (ack) de la confirmación de apertura de conexión (syn).

Chapter 6

Entornos Híbridos

6.1 Introducción

Un nuevo entorno de intercomunicación, surgido del cada vez más amplio uso de ordenadores portátiles y de las redes celulares de telefonía digital, está cobrando mayor importancia. Se trata de la utilización de los servicios de datos de las redes telefónicas celulares para acceder a servidores de información típicamente situados en Internet. Este nuevo entorno está basado en tecnología inalámbrica, al contrario que las redes tradicionales, lo cual hace de él algo especial.

Por un lado, surge el concepto de usuario móvil o nómada, que se mueve a lo largo de la red mientras se mantiene conectado solicitando diferentes servicios de información. Esta es una de las ventajas de este tipo de redes: la ubicuidad en el acceso, esto es, la posibilidad de acceder a la red desde cualquier punto de ésta. Además esta movilidad es gestionada por la red, por lo que resulta transparente a las aplicaciones que sobre ella actúen, y también a los diferentes protocolos de comunicaciones.

Las anteriores características suponen una ventaja de este tipo de acceso a datos frente a otros, sin embargo, la tecnología en la que se basan para ello tiene su precio a pagar. Los enlaces inalámbricos de las redes celulares poseen características diferentes de las de los enlaces alámbricos de las redes fijas. Mientras éstos pueden gozar de anchos de banda satisfactorios y retardos pequeños y constantes, aquellos resultan ser un recurso bastante pobre, ofreciendo un ancho de banda bajo y variable, junto con también variables y largos retardos. Añadido a esto, los enlaces inalámbricos suelen tener tasas de error considerables y ser propensos a desconexiones súbitas.

Enlaces de este tipo no fueron tenidos en cuenta cuando fueron definidos los protocolos de comunicaciones clásicos. Así, el uso de TCP sobre redes basadas en este tipo de enlaces resulta especialmente problemático. Diseñado con un ajuste fino para redes basadas en enlaces alámbricos, el control de la congestión de TCP no tiene un comportamiento adecuado en el nuevo tipo de redes. Este mecanismo malinterpreta las tardanzas de los acuses de recibo (ACK), tardanza debida a las pobres características de los enlaces inalámbricos, como pérdida debida a la excesiva congestión de la red, y actúa activando mecanismos que tienen como objetivo aliviar una congestión que no es tal y, como consecuencia, la reducción del flujo de emisión, dejando de aprovechar el ya de por sí escaso ancho de

banda disponible. A este asunto ya han sido dedicados estudios anteriores a este, estudios que han analizado el problema y propuesto diferentes soluciones. La sección 3 es el dedicado en este informe a analizar el problema, mientras la sección 4 presenta las soluciones ya propuestas en los mencionados estudios.

En la sección 2 se presenta el entorno concreto en el que se ha enfocado este estudio, este es, acceso a Internet a través de redes GSM. En este escenario, los clientes son equipos portátiles que acceden a servidores localizados en la red fija (Internet) a través de una red basada en esta tecnología ya nombrada (GSM). El interés por este sistema se debe a su masivo uso en Europa y a su progresiva expansión por el resto del mundo.

Pero este capítulo no sólo es un estudio sobre este tipo de entornos y sus problemas. Recogidas la sección 5, hay dos propuestas de mejora pensadas para el sistema concreto presentado en la sección 2, pero cuyas ideas pueden ser utilizadas en cualquier otro sistema de acceso a Internet a través de una red telefónica celular y, en general, en cualquier entorno de intercomunicación de datos en el que estén implicadas redes basadas en tecnologías alámbricas e inalámbricas. La primera de las propuestas de mejora consiste en utilizar el modelo indirecto de conexión, esto es, partir la conexión TCP en dos partes, de manera que cada una sea gestionada independientemente y que un elemento situado entre ambas partes las coordine. Cada una de estas partes representara a cada uno de los mundos que forman el tipo de situaciones que se desea estudiar: el mundo inalámbrico y el mundo alámbrico. Dividiendo la conexión, se espera aislar los problemas en la parte móvil de la comunicación. Esto puede resultar en un aumento de la eficiencia, en especial si se combina con la segunda propuesta de mejora, que consiste en la sustitución, en la parte móvil de la conexión, del protocolo de transporte. Si éste era antes TCP, se propone que sea ahora un nuevo protocolo, STP, diseñado especialmente para funcionar sobre enlaces inalámbricos. Una de las características de este protocolo es que no realiza control de la congestión. Sobre las razones de esto y sobre más detalles de las dos alternativas o propuestas de mejora se habla, como ya se ha dicho, en la sección.

En la sección 6, presenta el trabajo que se está realizando para dar aval a este estudio. Se trata de simulaciones de los diferentes sistemas, de las que se extraerán medidas con las que se espera demostrar que lo discutido y propuesto en este estudio tiene sentido.

En la última sección, se recogen las conclusiones del estudio.

6.2 Acceso a Internet a través de GSM

Los sistemas GSM de telefonía celular digital, predominan en Europa y se están extendiendo rápidamente en otros continentes. Ofrecen una nueva plataforma de acceso a datos muy atractiva en un momento en que la movilidad del usuario esta adquiriendo importancia y asiduidad. Las nuevas tecnologías en las que se basan no fueron tenidas en cuenta en el momento de definir los protocolos que actualmente se usan en las redes de ordenadores. Por ello, el uso de estos supone un problema de ineficiente uso de la ya pobre capacidad de unos sistemas diseñados para ofrecer, en un principio, servicios de telefonía móvil.

Figure 6.1: Acceso a Internet a través de GSM

6.2.1 Servicio de datos

GSM (Global System for Mobile Communications) es un estándar paneuropeo de telefonía celular digital de mayoritario uso en las redes europeas que ofrecen este tipo de servicio y que está extendiéndose rápidamente en el resto del mundo, especialmente en Asia y Australia. Este estándar contempla también la transmisión de datos, que supuso en el momento de su definición, un objetivo secundario. Sin embargo, las estimaciones indican que para el año 2000 la mitad del tráfico que circulará por estas redes será de datos[Rivanedeyra et al., 1998a].

El servicio que hoy en día ofrece este tipo sistemas es bastante pobre: conmutación de circuitos con un ancho de banda nominal de 9'6Kbps que en el mejor de los casos se reduce a 7 Kbps [Alanko et al., 1994]. No obstante nuevas partes del estándar (HSCSD y GPRS) estén siendo desarrolladas con el fin de conseguir alcanzar velocidades de transmisión más altas, el tiempo en que éstas lleguen a ser usadas queda aún lejano. Es posible también conseguir mayores índices de transferencias con el uso de conexiones totalmente digitales (conexión de GSM a RDSI) y compresión de datos, pero de momento el uso de la RTC es mayoritario para el acceso a Internet y además aquello no supone una mejora en el aprovechamiento del ancho de banda de los enlaces, que sigue siendo el mismo. Además, los retardos son de una magnitud considerable, al igual que las tasas de error, aunque el uso de ciertos protocolos puede hacer los errores transparentes al usuario, con un aumento en la variabilidad del retardo como consecuencia. La causas de pérdida de paquetes o error en la conexión, son principalmente los fallos en los enlaces inalámbricos, los handoffs y la falta de cobertura [Cáceres et al., 1995], y rara vez congestión de la red.

Los handoffs son los cambios de celda durante la comunicación. Gestionarlos adecuadamente forma parte del control de movilidad de cualquier red celular y, por tanto, de las redes GSM. En estas, pueden encontrarse elementos llamados MSC (Mobile Switching Center), que se usan para control del encaminamiento y de la localización y movilidad de los terminales. Para ello se usan dos bases de datos: VLR (Visitor Location Register) y HLR (Home Location Register). Típicamente, un MSC controla varias celdas y tiene asociado un VLR donde almacena información sobre los terminales móviles presentes en el área que controla. El HLR se utiliza para almacenar la información de localización de cada usuario móvil en la red. En lo que respecta a este estudio, lo importante es saber que cuando un terminal se mueve en el transcurso de una conexión, puede salir de la zona de cobertura de una celda para pasar a la de otra, activando entonces el sistema mecanismos que, utilizando los elementos antes presentados, mantienen la conexión, pero provocando una pausa que puede llegar a ser causa de excesivos retardos o pérdidas de paquetes [Cáceres et al., 1995].

6.2.2 Acceso a Internet

El acceso convencional a Internet a través de redes GSM sigue el modelo que puede verse en la figura 6.1. El terminal (típicamente ordenador portátil) se conecta a la red a través de un teléfono móvil usando una tarjeta que emula a un módem, de manera que el software clásico puede seguir siendo utilizado. La

Figure 6.2: Arquitectura actual de acceso a Internet a través de GSM

conexión del usuario al servidor debe atravesar tres redes. Primero de todo, la red GSM en la que una batería de módems sita en los MSC (Mobile Switching Center) permite conectarse a través de la red telefónica conmutada al servidor de acceso a Internet (esto es lo típico; el ISP, Internet Service Provider, podría estar también en una RDSI). Finalmente, este ISP se encarga de establecer la conexión con el servidor de información en Internet . Lo nuevo de esta situación frente a las clásicas de acceso a Internet es el uso de la red GSM. Es nueva precisamente por la tecnología en que están basadas este tipo de redes. Ésta limita el ancho de banda de los enlaces a 9'6 Kbps. Las características del retardo y la tasa de errores dependen del uso de uno de los modos de servicio que especifica GSM: modo transparente y modo no transparente. La transparencia se refiere a los errores de enlace.

Así, en el modo transparente, que sigue la especificación ISDN V.110, la tasa de errores ronda el 10^{-3}, siendo ancho de banda y retardo ambos constantes.

En el modo transparente, el uso de un protocolo de corrección de errores, basado en retransmisiones selectivas, llamado RLP (Radio Link Protocol), hace que la tasa de errores baje a 10^{-8}, pero como consecuencia, la latencia aumenta y ancho de banda y retardo se vuelven variables, sobre todo cuando las condiciones del enlace son pobres. Este último modo, será el tenido en cuenta a partir de ahora, pues es servicio de datos habitualmente elegido. La pila de protocolos usada en esta situación puede ser como la de figura 6.2.

En ésta, puede verse como el protocolo RLP es usado entre el equipo móvil y el MSC, mientras que un protocolo de módems estándar es utilizado para conectar a éste con el ISP. El protocolo PPP es utilizado para gestionar el enlace entre el equipo móvil y el ISP. Puede usarse tanto en el caso de que la dirección IP del cliente sea asignada dinámicamente como en el caso contrario (dirección IP fija). En este último caso, se puede utilizar, en lugar de PPP, el protocolo SLIP (Serial Line IP) o su versión comprimida (CSLIP). El resto de protocolos es el habitual en entornos TCP/IP.

Lo que realmente es de interés para este estudio es la existencia de dos mundos basados en tecnologías diferentes: alámbrica e inálambrica, y el uso de un protocolo de transporte TCP encargado de gestionar las conexiones que atravesarán ambos mundos. Por motivos que más adelante se explicarán, el elemento separador de ambos mundos será el ISP, situado no entre el mundo álambrico y el inálambrico, sino entre la red GSM y la red Internet. En cualquier caso lo que nos interesa es que un protocolo de transporte diseñado para un tipo de enlaces es utilizado para gestionar conexiones que implican enlaces de otro tipo, y que esto, como en breve se explicará es origen de problemas de ineficiencia.

6.3 Problemas con TCP

TCP es el protocolo de nivel de transporte estándar en Internet. Ofrece a los protocolos de nivel de aplicación un servicio fiable de transporte de datos orientado a conexión, es decir, garantiza el envío de extremo a extremo de la conexión de la información sin errores y en orden. Para poder garantizar este

servicio, las implementaciones de TCP cuentan con varios mecanismos.

Para asegurar la entrega de segmentos (unidad de intercambio de datos entre entidades de nivel de transporte), TCP hace uso de ACKs y retransmisiones. Un ACK es un acuse de recibo por parte del receptor; indica al emisor la recepción exitosa de segmentos. Segmentos sin acuse de recibo deberían ser retransmitidos. Las causas por las que no se recibe ACK de un segmento pueden ser varias.

Una de ellas es la congestión de la red, que puede haber causado el descarte del paquete enviado o del mismo ACK. Como es absurdo esperar indefinidamente la llegada de un ACK, se ha de hacer uso de temporizadores. Estos son utilizados por los mecanismos de control de congestión de TCP para reaccionar ante este problema. Enviado un paquete, un temporizador se activa. Cuando éste se dispare, es decir, cuando cierto tiempo haya transcurrido desde el envío del paquete, se considera el paquete por perdido achacándolo a la congestión de la red, pues así sucede mayoritariamente en redes alámbricas. Como acción correctiva, se activan ciertos mecanismos como slow-start y el algoritmo de Karn que tienen como respectivas consecuencias la reducción de la ventana de emisión (que determina el flujo de datos a enviar) y el aumento del valor de los temporizadores de retransmisión (hasta el doble del actual), en espera de aliviar así la congestión de la red. Este mecanismo de congestión es adaptativo; su parámetros se van ajustando a la situación de la conexión.

Diseñado en principio con acierto, pues la principal causa de tardanza de los ACKs en las redes tradicionales (álambricas) es la pérdida por congestión, su uso en entornos inalámbricos supone un problema de ineficiencia. En estos entornos, los retardos de los enlaces son de una magnitud considerable y sufren grandes variaciones, debida a los mecanismos de control de errores (RLP). Todo esto hace que el mecansimo de control de la congestión se comporte de manera inadecuada. Éste interpreta las excesivas tardanzas de los acuses de recibo como problemas de congestión, cuando la verdadera razón es una de las anteriormente citadas. Reacciona entonces este mecanismo reduciendo el flujo de datos, lo que supone un menor uso del ya escaso ancho de banda.

Resumiendo, el mecanismo de control de la congestión de TCP reduce significativamente la eficiencia en entornos inalámbricos como el que es objeto de estudio en este capítulo. Además otras características de TCP, como la redundancia en las cabeceras de los paquetes, contribuyen al uso ineficiente del pobre ancho de banda de los enlaces inalámbricos.

6.4 Estudios anteriores

Ya varios estudios sobre el tema que nos ocupa han sido realizados. El presente estudio se basa en los informes correspondientes a tales estudios. En esta sección se ha intentado recoger las ideas más interesantes que en estos informes puede encontrarse, sobre todo en lo que se refiere a mejoras de las comunicaciones entre clientes situados en una red celular y servidores en una red fija.

6.4.1 Medidas de rendimiento

Los artículos [Alanko et al., 1994] y [Laamanen, 1995] recogen medidas de rendimiento en el acceso a redes fijas (o alámbricas: como Internet) a través de la red GSM.

En realidad [Alanko et al., 1995] estudia también otro sistema de telefonía celular análogo a GSM, llamado NMT, pero lo que es de interés para el presente informe son las medidas de rendimiento sobre el primero de los sistemas. El rendimiento es medido en tres tipos de operaciones: establecimiento de conexión en la parte inálambrica, intercambio de mensajes petición respuesta, y transmisión continua de datos. Con la primera de las operaciones se recogen medidas sobre el tiempo invertido en realizar la conexión en el enlace GSM. Los resultados son que el tiempo típico se sitúa en torno a los 28 y 31 segundos, nunca superando los 35 segundos. Para medir los tiempos de round-trip, tiempo que transcurre entre la emisión de una petición y la recepción de la respuesta, se utiliza el segundo tipo de operación mencionado. El tiempo de round-trip da una idea del retardo del enlace GSM. Diferentes valores de este parámetro han sido obtenidos dependiendo del tamaño de las peticiones y las respuestas. Para mensajes de tamaño mínimo, el tiempo de round-trip tiene un valor medio de aproximadamente 1 segundo, mientras que valores mayores de los mensajes pueden elevar el round-trip medio hasta 1.5, 2.2 e incluso 3.0 segundos. La última prueba permite recoger valores de throughput en la transmisión de datos. Los valores throughput obtenidos son de 7'11 Kbps en condiciones ideales, 5'6 Kbps en buenas condiciones y 2'133 Kbps en condiciones malas.

El artículo [Laamanen, 1995] recoge también las medidas del anterior, pero enfocando el problema a situaciones más concretas. De éstas no se dirá nada, pues no aportan nada nuevo en lo que incumbe al presente estudio. Como curiosidad sí se dirá que [Laamanen, 1995] recoge además medidas de satisfacción de usuarios de un sistema de acceso a redes de datos fija a través de GSM, medidas que indican que los usuarios valoran mayoritariamente el rendimiento del sistema y de la red como satisfactorio o bueno.

En [Cáceres et al., 1995] se mide el tiempo de pausa de la comunicación cuando ocurre un handoff en 0.8 segundos.

6.4.2 Propuestas de Mejora

La mayoría de los artículos utilizados proponen mejoras del rendimiento en los entornos de comunicación móviles, todos ellos identificando los mecanismos de control de la congestión como principal causa de ineficiencia. Las soluciones de mejora que se proponen pueden agruparse en mejoras en el protocolo TCP, sistemas de comunicación indirectos y mejoras a niveles inferiores al nivel de transporte. Aparte de estos tres grupos, otro se ha añadido en el que se recogen propuestas que pretenden mejorar el rendimiento de comunicaciones web.

Mejoras en TCP de extremo a extremo

Un número considerable de estudios proponen el uso de diferentes técnicas y arquitecturas, que manteniendo la utilización de TCP de extremo a extremo, mejoran el rendimiento cuando entornos inalámbricos están implicados. Se describen a continuación algunas de éstas.

Compresión de las cabeceras de TCP/IP Varios estudios (por ejemplo, [Degemark et al., 1996] y [Jacobson, 1990] han propuesto diferentes técnicas de compresión de las cabeceras TCP/IP basándose en el hecho de que para secuencias consecutivas de paquetes, las variaciones en la información de las

cabeceras son mínimas. Básicamente la idea es que el emisor envía al principio una cabecera completa que el receptor almacena. Paquetes sucesivos sólo contienen los cambios en la cabecera, con el consiguiente ahorro en el ancho de banda utilizado.

Retransmisiones selectivas (SACKs) Habitualmente las implementaciones de TCP usan acuses de recibo o ACKs acumulativos, esto es, el receptor un ACK si ha recibido exitosamente el paquete siguiente al anteriormente recibido. Esto no ofrece suficiente información al emisor, para una eficiente retransmisión. Puede darse el caso de que de una secuencia de paquetes, sólo el primero se ha perdido, habiendo llegado el resto con éxito al receptor. Éste no enviará ACK alguno y el emisor dará por pérdida la secuencia completa, haciendo retransmisiones inútiles. La utilización de SACKs (Selective Acknoledgements), evitaría estas retransmisiones. Esta técnica supone una ventaja cuando se dan situaciones de pérdidas múltiples de paquetes, según algunos estudios han demostrado [Fall et al., 1996]. Existen dos propuestas concretas presentadas en [Keshav et al., 1996] y [Mathis et al., 1996].

Notificación explícita de error En [Bakshi et al., 1995] y [Balakrishnan et al., 1996] se proponen dos técnicas diferentes de notificación explícita de errores: Explicit Bad State Notification (EBSN) y Explicit Loss Notification (ELN). La idea común es enviar información al TCP del emisor sobre pérdidas de paquetes con el fin de que no active los mecanismos de control de la congestión.

Variación de los tamaños de los paquetes En [Bakshi et al., 1995] se estudia la influencia que tiene sobre el rendimiento el tamaño de los paquetes utilizados en la red inalámbrica a nivel de enlace. Se concluye que una buena elección del tamaño puede suponer una mejora en el rendimiento del 30

Fast Retransmission Ciertas implementaciones de TCP incorporan un mecanismo de retransmisión rápida (fast retransmission) que se activa al detectar el inequívoco signo de pérdida de paquete que supone el recibir ACKs triplicados. Lo que hace este mecanismo es retransmitir el último paquete sin ACK, reducir la ventana de emisión al mínimo e iniciar el algoritmo slow-start, todo ello sin esperar a que salte ningún temporizador. En [Cáceres et al., 1996] se propone una técnica que activa este mecanismo cuando un handoff es completado, con el fin de recuperar rápidamente la comunicación .

Mejoras que implican el uso de modelos indirectos de comunicación

Puesto que los problemas de ineficiencia se producen por causa de la naturaleza de los enlaces inalámbricos, diferentes estudios han propuesto el uso del modelo de comunicación indirecto, típicamente a nivel de transporte. La idea básica, tratada con mayor profundidad en la sección 5, consiste en partir la conexión en varias partes, según la conexión sea sobre el mundo inálambrico o el alámbrico, con lo que se consigue aislar los problemas allí donde tienen su origen, e implementar protocolos específicos que corrijan o alivien esos problemas. Se pierde a cambio la semántica de comunicación de extremo a extremo de la conexión

y la transparencia de la movilidad del usuario. Se presenta a continuación una descripción de diferentes técnicas asociadas al uso de este tipo de sistemas.

Situación del intermediario El intermediario, es el elemento separador de las diferentes partes de la comunicación. Existen diferentes propuestas de ubicación de este elemento. En [Rivanedeyra et al., 1998a] se propone que el intermediario sea un elemento al que llaman GSN (Gateway Support Node) cumpliendo tareas adicionales de ISP. De esta manera la división de mundos es entre el mundo GSM e Internet, no entre el mundo inalámbrico y el alámbrico como propone [Wang et al., 1998], al colocar el intermediario en las estaciones base (Base Stations) de la red celular. La arquitectura Mowgli, presentada en [Kojo et al. 1995] propone situar el intermediario en el servidor de acceso a Internet (ISP), haciendo de esta manera que la división sea entre la red de acceso y la red Internet .

Funciones del proxy Las aplicaciones que se ejecutan en el intermediario reciben el nombre de filtros o Proxy. Son diferentes las funciones de filtro que se pueden realizar, desde una simple repetición, hasta resegmentación o descarte de información. En la sección 5 se trata esto con mayor profundidad.

Reducción de cabeceras Las propuestas [Rivanedeyra et al., 1998] y [Wang et al., 1995] coinciden en proponer el uso de cabeceras de tamaño reducido en el protocolo de transporte usado en las partes inalámbricas de la conexión. [Rivanedeyra et al., 1998] no dice más que esto, pero [Wang et al., 1995] es más explícito y presenta la cabecera utilizado por el protocolo de transporte que se propone (METP), y habla no sólo de reducción sino también del uso de la compresión de cabeceras ya presentada anteriormente.

Omitir el control de la congestión Con la propuesta de dividir la conexión se consigue aislar las partes en las que el control de la congestión es principal causa de pobre rendimiento. Pero además se ha conseguido que en éstas no haya congestión alguna, haciendo inútil el uso de mecanismos de control de la congestión. Los protocolos de transporte propuestos en [Rivanedeyra et al., 1998], [Wang et al., 1995] y [Kojo et al., 1997], no incluyen entre sus funciones el control de la congestión.

Omitir el control de errores Cuando se usa el modo no transparente en accesos a datos a través de GSM, el protocolo RLP se encarga de corregir los errores. Se podría, por tanto, omitir el control de errores a nivel de transporte en aras de conseguir un mayor rendimiento en la comunicación, tal como se dice en [Rivanedeyra et al., 1998].

Control de flujo En [Rivanedeyra et al., 1998] aparece la idea de que el control de flujo debe adecuarse a las características del enlace sobre el cual actual el protocolo de transporte propuesto. Concretando un poco más, se dice que el uso de ACK debe ser minimizado. En [Kojo et al., 1997] se concretiza todavía más. La política *lazy-ack* es la que se usa en la solución propuesta en ese artículo. La idea es mandar ACKs no individuales, sino para grupos de paquetes recibidos. De esta manera se pretende ahorrar ancho de banda.

Por contra, en [Wang et al., 1998] el protocolo de transporte propuesto, envía los acuses de recibo, tan pronto como se recibe un paquete (o segmento). La retroalimentación es usada, en los dos estudios previamente referenciados, en el control del flujo, para mantener informado al emisor de nivel de transporte, sobre la ocupación del receptor.

Retransmisiones Selectivas Propuestas estudiadas en [Balakrishnan et al., 1996] y [Balakrishnan et al., 1997], mantienen el protocolo TCP en las partes móviles de la comunicación, pero utilizando la ya presentada técnica de retransmisiones selectivas.

6.4.3 Mejoras a nivel de enlace

Snoop Protocol En [Balakrishnan et al., 1997] se presenta un protocolo que actúa por debajo de TCP monitorizando los paquetes enviados y recibidos por los equipos móviles. El agente se sitúa en las estaciones base y realiza funciones de cache y de filtro (no propaga ACKs duplicados).

SACK a nivel de enlace La técnica ya presentada de retransmisiones selectivas puede ser utilizada también a nivel de enlace.

Mejoras en tráfico web

Se incluyen aquí varias propuestas de mejora del rendimiento de conexiones TCP cuando tratan con tráfico Web. El interés está en que la Web es la aplicación por excelencia de Internet, y no será ignorada por usuarios de los entornos sobre los que trata este estudio. Aunque los estudios de los que se han recogido las propuestas de mejora no están dedicados a estos entornos, las ideas en ellos recogidas pueden ser utilizadas.

Enhanced TCP Loss Recovery Una de las técnicas presentadas en [Balakrishnan et al., 1997] tiene como objetivo mejorar el comportamiento de los mecanismos de recuperación ante errores cuando se trata con conexiones web únicas. Básicamente la idea consiste en enviar un nuevo segmento y esperar la recepción de ACKs duplicados para verificar la pérdida de paquetes y comenzar las retransmisiones.

Integrated Connection También en [Balakrishnan et al., 1997] se proponen técnicas para mejorar situaciones en las que existen múltiples conexiones TCP. Habitualmente esto es lo que ocurre en conexiones web, cuando servidor y cliente utilizan una conexión para intercambiar cada componente de una página. En [Balakrishnan et al., 1997] se llama la atención sobre el impacto negativo que este paralelismo implica, y se propone la integración del control de la congestión y de la recuperación ante errores de las diferentes conexiones TCP, manteniendo independiente para cada conexión todo lo relacionado con envío fiable y en orden de datos (como el control de flujo).

TCP Fast Start Esta técnica propuesta en [Padmanabhan et al., 1998] pretende aumentar el rendimiento en transferencias web cortas. La idea es que el emisor de tráfico web almacene información de ciertos parámetros de la red (valores antiguos de la ventana de congestión y del tiempo de round-trip estimado) para evitar tener que ejecutar el algoritmo slow-start para cada página web.

6.5 Propuestas de mejora del proyecto

El proyecto de cuyo conjunto de entregables forma parte este capítulo tiene como objetivo avalar por medio de simulaciones las propuestas recogidas en [Rivanedeyra et al., 1998a] y [Rivanedeyra et al., 1998b].

La alternativa de mejora del sistema actual de acceso a Internet a través de GSM que en dichos documentos se propone, incluye varias de las ideas presentadas en la sección anterior. Frente a mantener la conexión de extremo a extremo, se opta por la utilización del modelo indirecto de comunicación, utilizando en la parte móvil un nuevo protocolo de transporte al que se llamará STP (Simple Tranfer Protocol) para cuyo diseño se tienen en cuenta técnicas como omisión del control de errores, utilización de cabeceras reducidas, etc.

A continuación se hace una descripción más detallada de las dos técnicas en las que se basa la propuesta de mejora: el modelo indirecto de comunicación y el protocolo STP.

6.5.1 Modelo Indirecto

La idea del modelo indirecto de comunicación, utilizada a nivel de conexión, consiste en dividir las conexiones en partes de tal manera que cada conexión sea gestionada independientemente y elementos situados entre las partes se encarguen de coordinarlas. Las ventajas aparecen cuando cada una de las partes de la conexión tiene lugar en un entorno de diferente naturaleza a la de las otras partes, pues de esta manera es posible realizar un tratamiento especifico de cada una de las partes con el fin de lograr mayor eficiencia.

Lamentablemente, esto tiene sus desventajas, pues la semántica de extremo a extremo de las conexiones, se pierde. Para tener una idea de las consecuencias, piénsese, por ejemplo, que con el uso de TCP, el emisor recibirá ACKs de un intermediario y no del receptor, no sabiendo entonces cuando éste ha realmente recibido con éxito la información.

Cuando se decide utilizar este tipo de sistemas, deben tenerse en cuenta dos importantes aspectos de diseño: cómo se dividirán las conexiones, esto es, dónde se ubicarán los intermediarios, y qué funciones realizarán éstos. A continuación se presenta la arquitectura de acceso a Internet a través de GSM propuesta como alternativa a la arquitectura actual, definiendo los dos citados aspectos de diseño: ubicación y funciones de los intermediarios.

Siguiendo [Rivanedeyra et al., 1998a] y [Rivanedeyra et al., 1998b], el intermediario se coloca en un nodo al que se llama GSN (Gateway Support Node). Este nodo forma parte de la red GSM y realizará funciones adicionales de ISP, lo cual hará aumentar el valor añadido del operador de la red GSM al añadir a la red un servicio más. La red telefónica conmutada deja por tanto de ser utilizada. La división de mundos que se hace no es entre entornos alámbricos e

Figure 6.3: Propuesta de arquitectura de acceso a Internet a través de GSM.

inalámbricos, sino entre la red GSM e Internet. La pila de protocolos resultante es la de la figura 6.3.

La principal diferencia con el sistema actual de acceso a Internet presentado en la sección 2 es que en la parte móvil el protocolo de transporte es, no ya TCP, sino un nuevo protocolo llamado STP. Las características de éste se presentan más adelante.

En el intermediario, y a nivel de aplicación, habrá un conjunto de programas llamados filtros. Estos filtros son conocidos en conjunto como Proxy y son los encargados de coordinar las conexiones en ambas partes en las que está dividida. En principio la tarea que el Proxy realizará será la de puente, esto es, simple repetidor. Funciones adicionales que esta entidad podría realizar serían descarte selectivo de información de cabeceras (por ejemplo en paquetes SMTP), prorrogar el envío de datos (por ejemplo con tráfico POP mail), resegmentación de datos, reducir tráfico *frívolo* enviado al cliente móvil (por ejemplo replicando peticiones de eco ICMP) [Rivanedeyra et al., 1998a].

6.5.2 Simple Transfer Protocol

La anterior propuesta logra aislar los problemas en la parte móvil. Dichos problemas, ya se ha contado, se deben a lo inapropiado del uso de TCP en entornos inalámbricos, que tiene como consecuencia un ineficiente uso del escaso ancho de banda de los enlaces.

Como segunda mejora, se propone la sustitución del protocolo de transporte TCP por otro diseñado específicamente para funcionar en la parte inálambrica de la comunicación de la manera mas eficiente posible. Para ello se debe tener en cuenta las ya descritas características de los enlaces inalámbricos, esto es, escaso ancho de banda y largos retardos, con el añadido de la variabilidad en ambos parámetros.

El nuevo protocolo de transporte se llamará Simple Transport Protocol (STP) e incluirá algunas de las siguientes técnicas ya presentadas en un anterior capítulo: reducción de cabeceras para no desaprovechar ancho de banda con información prescindible, omisión del mecansimo de control de congestión para evitar los problemas que su uso acarrea y omisión también del mecanismo de control de errores, inútil por encargarse ya de ello el nivel de enlace y por ser la comunicación punto a punto.

Independientemente de las características del nuevo protocolo de transporte, la introducción de éste sólo supondría cambios en el cliente y no en el servidor. Los cambios pueden localizarse en la aplicaciones, pero lo más normal sería que únicamente fuera necesario cambiar la biblioteca que implementa el servicio de transporte (típicamente sockets), con lo que solamente haría falta recompilar o ni siquiera eso si la carga es dinámica.

6.6 Trabajos Futuros

El presente informe pretende satisfacer uno de los objetivos del proyecto en el cual está enmarcado. Este objetivo es presentar un conjunto viable de alterna-

tivas a la actual arquitectura de comunicaciones de acceso a Internet a través de GSM. Estas alternativas, serán posteriormente validadas mediante simulaciones. Esto es también objetivo del citado proyecto. Se espera poder modelar sencilla pero fielmente las situaciones objeto de estudio con una herramienta de simulación de redes y obtener resultados que avalen este estudio.

Los resultados de las simulaciones podrán encontrase en el capítulo 8, que forma parte del conjunto de entregables del proyecto.

6.7 Conclusiones

Se ha presentado un emergente entorno de comunicaciones basado en tecnologías inalámbricas, poniendo de relevancia los problemas que en el surgen debido al mal comportamiento de los protocolos tradicionales (especialmente TCP) en estos entornos. Dichos problemas repercuten en la eficiencia en el uso de un recurso que es escaso en el nuevo entorno: el ancho de banda. Como mejora se ha propuesto una arquitectura alternativa basada el uso del modelo indirecto de comunicación, esto es, partir las conexiones en dos, y en un nuevo protocolo de transporte como sustituto a TCP en la parte móvil de la comunicación. Estas dos técnicas han sido elegidas de entre un conjunto recogido de diferentes estudios sobre problemas similares. Todo esto que se ha discutido teóricamente, se espera sea validado por pruebas de simulación que se realizarán como parte del mismo proyecto del que forma parte este informe.

Chapter 7

Elementos Necesarios

7.1 Introducción

En el capítulo 8 se presenta un conjunto de simulaciones cuyo fin es avalar el estudio realizado en el capítulo 6, informe que trata sobre un determinado tipo de entorno de comunicación de datos. Como herramienta de simulación se ha utilizado el simulador de redes ns desarrollado por el Lawrence Berkeley National Laboratory [Network Research Group, 1998].

El modelo del mundo *real* que se ha simulado se ha considerado compuesto por varios componentes a los que se ha llamado Elementos Necesarios. Estos elementos son enlaces, aplicaciones, y protocolos. En este capítulo se presenta la caracterización (o diseño) de éstos, así como su implementación.

Algunos de los elementos necesarios ya estaban incorporados en la herramienta y han sido utilizados, realizando en algunos casos modificaciones necesarias. Otros de los elementos no proporcionados por la herramienta de simulación han debido ser implementados enteramente o tomados de otra fuente, modificándolos para adecuarlos a las necesidades.

7.2 Enlaces

7.2.1 Diseño

En la topología que se utiliza para simular las situaciones de los entornos presentados en el capítuloEntornos Hibridos, sólo dos enlaces tienen cabida: uno al que llamaremos enlace alámbrico y otro al que llamaremos enlace inalámbrico.

Enlace alámbrico

El enlace alámbrico une al nodo en el que está situado el servidor de la red fija con el nodo que representa al ISP (para más detalles ver capítuloEntornos Hibridos). En el mundo real éste no es un enlace (es una comunicación TCP/IP clásica), pero en el modelo es uno sólo que representa todos los enlaces de esta comunicación.

En el mundo real el camino entre un terminal y otro a través de Internet, como en el caso de el servidor y el ISP, está compuesto por enlaces fiables y rápidos en general. A la hora de modelar este conjunto de enlaces en uno sólo

se debe elegir valores adecuados de ancho de banda y retardo. En [Wang et al., 1998] se utilizan valores de 1'5 Mbps y 50 ms para ambos parámetros del enlace. Se han realizado varias pruebas (transmisiones FTP, accesos a páginas web, uso de ping), y se ha optado por los valores de 100 Kbps y 100 ms).

Despreciamos la tasa de errores y cualquier causa de retransmisión que pudiera provocar este enlace. Despreciamos así errores y congestión, no incluyéndolos en el modelo.

Enlace inalámbrico

El enlace inalámbrico une al nodo que representa al cliente con el nodo que representa al ISP. En este caso en el mundo real sí que hay un único enlace entre ambas entidades.

Las características de tal enlace en las situaciones de acceso a Internet a través de GSM que se desean simular son una ancho de banda de 9'6 Kbps y un retardo largo y variable.

El valor del tiempo de round-trip ha sido objeto de estudio del artículo [Alanko, 1994]. Teniendo en cuenta estos datos podemos hacernos una idea los valores que puede tomar el retardo.

La variabilidad en este parámetro se debe al uso del modo no transparente. Cuando se usa este modo un protocolo especial se encarga de corregir los errores provocando a cambio la variabilidad en el retardo (ver capítuloEntornos Hibridos). Puede considerarse que el enlace pasa por estados en los que no hay errores y el retardo permanece constante con un valor "normal" y estados en los cuales hay errores cuya corrección provoca un aumento en el retardo.

Tomando ideas del artículo [Nguyen, 1996] y lo dicho hasta ahora, modelamos el enlace de manera parecida a un modelo de Markov con dos estados: un estado el que ocurren errores (estado E) y un estado en el que no ocurren errores (estado L). Al contrario que en [Nguyen, 1996] donde las duraciones de permanencia en cada estado vienen dadas por probabilidades de transición, aquí las duraciones se ajustarán a distribuciones aleatorias. En cada estado el retardo seguirá una cierta distribución.

Se ha decidido que los tiempos de duración de permanencia en los estados E y L sigan distribuciones exponenciales con media t_E y t_L respectivamente.

El retardo en el estado L seguirá una distribución constante con valor r_L y el retardo en el estado E seguirá una distribución uniforme entre los valores a_m y a_M.

Los valores que se han elegido para diferentes parámetros son los siguientes: $t_L = 50s$, $t_E = 10s$, $r_L = 400ms$, $a_m = 4rL$, $a_M = 6rL$

Son valores bastante arbitrarios que quizá hagan el modelo no demasiado real. No obstante, hasta el momento no se cuenta con los suficientes datos como para obtener un modelo que se aproxime más al mundo real. Se tiene la sospecha de que el modelo que se ha definido es más benévolo que el mundo real, es decir, que los enlaces reales tienen un comportamiento peor del de los enlaces modelados. Esto implica que resultados positivos en las simulaciones lo serán aún más en el mundo real.

7.2.2 Implementación

En ns el retardo de un enlace se modela mediante un objeto de la clase Connector. Por defecto, los enlaces utilizan objetos de la clase Delay, una subclase de la clase Connector que modela retardos constantes. Esta clase está implementada en los ficheros `ns/delay.h` y `ns/delay.cc`.

Existen dos ficheros llamados ns/delaymodel.h y ns/delaymodel.cc que se cree sean implementaciones de una subclase de la clase Connector que modelan un retardo no constante sino ajustado a distribuciones aleatorias. No obstante, esta clase es ignorada en la documentación como también lo es cualquier indicación sobre cómo hacer que un enlace utilice una clase u otra para modelar el retardo.

En un principio se pensó en una implementación *elegante* que consistía en crear una subclase de la clase enlace que utilizará una clase diferente a DelayLink para modelar el retardo. Esta opción hubo de ser descartada debido a falta de documentación y a la falta de familiarización con el código OTcl.

Finalmente se optó por modificar la clase para que incluyera mecanismos que simularan retardo variable y una vía para activar dichos mecanismos. De esta manera el comportamiento de un enlace es por defecto el original (retardo constante). Conseguir que el retardo de un enlace sea variable, en la manera como se ha diseñado en un apartado anterior, requiere activar el método `inicializar` del objeto que modela el retardo. El nombre de la variable que referencia a dicho objeto se llama `link_`. El método mencionado tiene cuatro parámetros `t_E`, `t_L`, `am` y `aM` que se corresponden con los parámetros t_E, t_L, am y aM mencionados en el apartado dedicado al diseño del enlace inalámbrico. El parámetro rL se toma del valor inicial de retardo, que se define junto al ancho de banda, al crear el enlace.

No se debe olvidar que los enlaces dúplex están implementados en ns como un par de enlaces simplex, por lo que, si se quiere activar la variabilidad en un enlace entre dos nodos, debe actuarse sobre los enlaces correspondientes a cada uno de los sentidos. Como la variabilidad se rige por cierta aleatoriedad, el comportamiento del retardo en ambos sentidos será diferente. Afortunadamente esto no supone un problema, sino que se ajusta a la realidad que se desea simular, donde fallos en los enlaces que provocan la variabilidad en los retardos tienen causa en interferencias que pueden afectar de manera diferente a cada uno de los canales correspondientes a ambos sentidos del enlace.

Para detalles de implementación, consúltese el capítulo 9, concretamente los apartados dedicados a los ficheros `ns/delay.h`, `ns/delay.cc`, `ns/tcl/lib/ns-default.tcl` y `definiciones.tcl`. Éste último fichero contiene definiciones comunes a todas las simulaciones, incluida la definición de los enlaces. Una sección de este mismo capítulo trata sobre estas definiciones comunes.

7.3 Comunicaciones

7.3.1 Diseño

En las simulaciones son probadas tres tipos de comunicación: comunicación tipo Telnet, comunicación tipo FTP y comunicación tipo WWW.

Comunicación tipo Telnet

La comunicación tipo Telnet es única y está caracterizada por mensajes cortos y bastantes separados entre sí, a intervalos variables. El cliente Telnet estará en el terminal móvil, mientras que el servidor Telnet estará en el servidor fijo.

El simulador utilizado provee una aplicación Telnet que genera tráfico que pretende asemejarse al generado por aplicaciones Telnet reales. Esta aplicación no ha sido considerada adecuada y se ha decidido modificar su código para más fielmente simular una aplicación Telnet.

En la nueva aplicación Telnet, hay dos tipos de entidades: cliente Telnet y servidor Telnet. La comunicación comienza cuando el cliente envía al servidor un mensaje pequeño que representa una petición. El Servidor recibe la petición y al instante responde con un mensaje de tamaño medio que representa la respuesta. Siguientes peticiones del cliente suceden cierto tiempo después de haber recibido la respuesta a la última petición.

De esta manera se pretende que el mayor rendimiento de los sistemas que se probarán, tenga impacto en la aplicación. Antes, el cliente enviaba peticiones sin que influyese el tiempo que tardaban en arribar a su destino. Ahora el cliente envía peticiones al servidor después de recibir las respuestas y tras un cierto intervalo de tiempo que representa el tiempo que el usuario tarda en asimilarla y en pensar en la siguiente petición. Cuanto más rápida vaya la comunicación, cliente y servidor intercambiarán mensajes con mayor frecuencia.

Comunicación tipo FTP

La comunicación tipo FTP es única y está caracterizada por mensajes largos y seguidos. El emisor será el servidor fijo y el receptor el cliente móvil.

Al igual que en el caso anterior, la herramienta de simulación provee una aplicación que pretende asemejarse al mundo real, y en este caso se ha considerado adecuada. Ésta simula una comunicación FTP real en la que es posible enviar mensajes de cierto tamaño o provocar el envío infinito (en realidad de tamaño máximo) de datos.

Comunicación tipo WWW

La comunicación WWW es múltiple. Las comunicaciones consisten en peticiones (mensajes pequeños) del cliente (situado en la parte móvil) a las que el servidor (situado en la parte fija) responde con mensajes de distinto tamaño representando las distintas partes de la página solicitada. Cada una de estas partes será enviada en una conexión TCP.

La aplicación provista por la herramienta de simulación utilizada ha sido descartada luego de mucho estudiarla, e incluso probarla e intentar modificar su implementación. Su comportamiento no era el adecuado y su implementación resultaba complicada y difícil de integrar con el resto de componentes de las simulaciones.

Se ha encontrado entre las contribuciones de usuarios de la herramienta de simulación ns recogidas en la página web de ns [Network Reseach Group, 1998], un módulo que implementa un generador de tráfico HTTP [Henderson et al, 1998a]. Su funcionamiento, que se ha considerado adecuado, se describe, a grandes rasgos, a continuación.

Se definen dos tipos de agentes generadores de tráfico: WWWClient que emula navegadores web y WWWServer que emula servidores web. El primer tipo de agente envía mensajes cortos que representan una petición al servidor al que está conectado. Cuando recibe la respuesta por parte del servidor, el cliente espera un tiempo aleatorio antes de enviar la siguiente petición. Los mensajes que llegan al servidor, son interpretados como solicitudes de página. La respuesta a tales peticiones es el envío de un número aleatorio de mensajes de tamaño aleatorio también. Importante es que cada mensaje, sea petición o respuesta, utiliza una conexión TCP independiente. Las distribuciones aleatorias han sido derivadas del estudio de Bruce Mah sobre modelos de tráfico HTTP [Mah, 1997].

7.3.2 Implementación

Comunicación tipo Telnet

La aplicación Telnet está implementada en los ficheros `ns/telnet.h` y `ns/telnet.cc`, que han sido modificados para incorporar los cambios deseados.

La clase Application/Telnet se ha utilizado para implementar los clientes Telnet. El único cambio requerido es que el temporizador que antiguamente se programaba tras la emisión de una petición, ahora se programa al recibir un mensaje de respuesta. Se considera recibido un mensaje de respuesta cuando se ha recibido cierta cantidad de bytes indicada por la variable `size_of_response_`. El tamaño de los mensajes de petición lo indica la variable `size_`.

Se ha introducido una nueva clase para implementar los servidores Telnet. Dicha clase se ha llamado Application/TelnetServer. El comportamiento de un agente de esta clase es simple: cuando recibe cierta cantidad de bytes (indicada por la variable `size_of_request_`), considera llegada una respuesta y envía un mensaje de la cantidad de bytes indicada por `size_`).

El valor de la variable `size_of_request_` de un servidor Telnet debe ser igual al valor de la variable `size_` del cliente Telnet. Lo mismo debe ocurrir entre las variables `size_` y `size_of_response_` pertenecientes al servidor y al cliente respectivamente. Lo contrario sería un absurdo. Los valores por defecto asignados han sido 64 bytes para peticiones y 512 bytes para respuestas. De momento, el tamaño de los mensajes permanecerá constante. Como futura mejora puede pensarse en hacer que los valores de tamaño de paquetes se ajusten más al mundo real, dónde casi seguro los tamaños no permanecen constantes.

Instalar la aplicación es sencillo. Simplemente debe crearse un objeto de la clase Application/Telnet y otro de la clase Application/TelnetServer y asociarlos a los agentes de transporte que corresponda. La aplicación se lanza con el método `start` del agente que representa al cliente Telnet..

En la sección de definiciones comunes se pueden encontrar más detalles.

Comunicación tipo FTP

La aplicación ftp está implementada en los ficheros `ns/tcl/lib/ns-source.tcl`. Ha sufrido cambios provocados por la implementación de la comunicación tipo WWW. Para más detalles consúltese el apartado correspondiente a dicha aplicación.

Al igual que con la aplicación Telnet, instalar la aplicación es sencillo. Simplemente debe crearse un objeto de la clase Application/FTP en esta ocasión y

asociarlo al agente de transporte que corresponda. La aplicación se lanza con el método `start` del objeto aplicación.

En la sección de definiciones comunes se pueden encontrar más detalles.

Comunicación tipo WWW

La aplicación WWW está implementada en el fichero `httpModel.tcl`. Utiliza además un conjunto de ficheros (con extensión `cdf`) para mecanismos internos. Ha requerido cambios para hacerlo compatible con el Proxy (elemento definido en una sección posterior) y para que utilice agentes de transportes bidireccionales, como lo son agentes de las clases Agent/TCP/FullTcp (protocolo por defecto) y Agent/TCP/STP (nuevo protocolo definido en una sección posterior), en lugar de agentes de transporte unidireccionales.

Los cambios relativos al Proxy consisten en que en el código original de la aplicación WWW, clientes y servidores utilizan la misma clase de protocolos de transporte. Se ha echo las modificaciones necesarias (en el fichero `httpModel.tcl`) para que aplicaciones cliente y servidor usen su propia clase de agente de transporte que podrá coincidir o no, y para que las conexiones entre agentes de transporte incluyan indicación del mundo al que pertenece cada agente (para más detalles consúltese la sección dedicado al Proxy).

Para que los agentes de transporte sean además bidireccionales, se ha debido modificar nuevamente el fichero httpModel.tcl para que en cada conexión se utilicen agentes de este tipo, creándolos como instancias de la clase que corresponda (la del cliente o la del servidor), conectándolos y poniendo el receptor en modo listen.

La aplicación utiliza agentes FTP para enviar los mensajes que representan las peticiones de los clientes y las respuestas de los servidores. Sin embargo, dichos mensajes no incluyen un indicador de fin de mensaje necesario para activar ciertos mecanismos internos de la aplicación WWW cuando se usan agentes de transporte bidireccionales. Dichos mecanismos permiten activar al servidor cuando un cliente ha terminado de enviar con éxito un mensaje de petición, y a éste cuando el servidor ha terminado de enviar con éxito todas las respuestas a una petición. Se ha modificado la implementación de la aplicación FTP (fichero `ns/tcl/lib/ns-default.tcl`) para que cada invocación al método send sea interpretada como el envío del un único mensaje de la conexión e incluya, por tanto, un indicador de cierre implícito de la conexión. Este cambio no afecta de momento a la comunicación FTP. Si ésta cambiara y se viera afectada, podría pensarse en añadir a la aplicación FTP provista por ns un método especial para indicar el fin de envío de mensajes.

En la implementación de la aplicación se incluye también código que cambia la implementación de las clases Agent/TCP/FullTcp y Agent/TCP/STP, añadiendo a cada una la implementación de un método y un par de variables que hacen posible el mecanismo interno antes mencionado. Se trata del método `done` (nombrado en [Network Research Group, 1998]) que se invoca cuando un agente de transporte es consciente de que el correspondiente receptor ha recibido todo lo enviado. Éste invoca a su vez el método `timeout` de la aplicación correspondiente que le hace saber que su envío ha sido exitoso. Dicha aplicación se guarda en el objeto de transporte bajo una variable `callback_` establecida en el momento de crear el agente. Si esta variable no ha sido instalada, es que el agente ha sido creado por el proxy y el *callback* debe hacerse por tanto in-

vocando el método`resume` (para más detalles sobre esto consúltese la sección dedicado al Proxy). La razón por la que se usa este mecanismo de *callbacks* es porque en ns cada agente de aplicación sólo puede tener asociado un agente de transporte y una aplicación WWW necesita más de un agente de transporte para enviar las distintas partes de la página.

En la sección de definiciones comunes y en el capítulo 9 se pueden encontrar más detalles.

7.4 Proxy

7.4.1 Diseño

En principio, el proxy no es más que una aplicación que hace de puente (relay o repetidor tonto) entre pares de conexiones TCP. Más adelante quizá se incorporen más funciones.

El proxy se instalará sobre un nodo de la red. Interceptará las peticiones de conexión entre entidades de nivel de transporte, creando dos agentes de transporte que se conectarán a los otros dos agentes de transporte fuente y sumidero de la petición de conexión. Se ha elegido la opción de interceptar conexiones frente a que las aplicaciones realicen solicitudes explícitas de conexión indirecta, esto es, usando el proxy, para reducir el impacto en las aplicaciones.

No obstante lo anterior, si se desea distinguir el mundo al que pertenecen fuente y sumidero asociados a una petición de conexión, puede hacerse necesario implementar algún mecanismo no implícito. Esto se verá en la implementación.

Una vez en marcha un par de conexiones a través del Proxy, éste se dedicará a repetir la información que le llegue por una de las partes de la conexión en la otra. Además si se le notifica sobre el cierre de la conexión en una de las partes, cerrará también la otra (cuando haya finalizado con la repetición).

En principio, consideramos que el Proxy tiene capacidad ilimitada a la hora de recibir información y de soportar conexiones, y que el envío es inmediato a la recepción (latencia 0). La capacidad ilimitada a la hora de recibir información se refiere al tamaño del buffer.

7.4.2 Implementación

El proxy ha sido implementado en los ficheros `ns/proxy.h` y `ns/proxy.cc` como una nueva clase de aplicación (subclase de la clase Application) a la que se ha llamado Proxy (Application/Proxy en el intérprete).

La clase Proxy incluye un método accesible a través del intérprete que permite asociar un par de agentes de transporte a un objeto de la clase. Dicho método se llama attach-agents. Una vez invocado, se crean dos objetos de la clase Iproxy (se le asocia a cada uno de los agentes de transporte) y se añaden a una lista de objetos IProxy.

La clase IProxy está también implementada en el mismo fichero como subclase de la clase aplicación. Los objetos de esta clase actúan como intermediarios entre el proxy y los agentes de transporte. Cuando un objeto IProxy recibe una cantidad de información por parte de su agente de transporte (mediante el método `recv(int nbytes))`, se lo notifica al proxy junto con su identificador, de manera que el proxy se encarga de ordenar al objeto IProxy del otro lado de

la conexión que envíe la misma cantidad de información a través de su agente de transporte. Sin un objeto IProxy recibe notificación de fin de conexión (mediante el método `resume()`), se lo hace saber al proxy, que a su vez lo notifica al otro objeto IProxy. Éste se encarga de cerrar la conexión en su parte tras haber terminado de enviar lo que le restaba por enviar.

La semántica del método `resume()` ha sido modificada. Originalmente indicaba a las aplicaciones emisoras que todo lo que habían enviado hasta el momento había sido enviado por el agente de transporte. Este método no es utilizado por ninguna de las aplicaciones de ns. Por esto y por su necesidad para un mejor comportamiento del proxy, se ha modificado su semántica y su implementación en la clase Agent/TCP/FullTCP (fichero `ns/tcp-full.cc`). La nueva semántica del método resume() es que éste es invocado cuando una aplicación ha recibido toda la información de una conexión y ésta ha sido cerrada.

Se ha añadido un método global install-proxy (fichero `ns/tcl/lib/ns-lib`) que dado un nodo y dos clases de protocolo de transporte, instala un proxy en dicho nodo, que utilizará cada clase de protocolo de transporte para cada mundo o parte de las conexiones que gestionará.

Se ha modificado el método connect (implementado en `ns/tcl/lib/ns-lib`) usado para conectar agentes de transporte, para que en caso de que el proxy esté instalado y situado en un nodo directamente enlazado con los nodos correspondientes a ambos agentes de transporte, cree una conexión partida. Además se ha añadido un parámetro opcional al método para indicar que la asociación de mundos a agentes es inversa a la asumida por defecto. Por defecto se asume que el primer agente pasado como parámetro al método `attach-agents` pertenece al mundo 1 y el segundo agente, al mundo 2, si se invoca `connect` con un parámetro distinto de cero la asociación de mundos es inversa. La identificación de mundos es de especial interés a la hora de recoger resultados.

Pueden encontrarse más detalles de implementación en el capítuloCambios.

7.5 TCP

7.5.1 Diseño

El simulador ns provee varias versiones del protocolo de transporte TCP. La versión elegida para las simulaciones ha sido la única implementación de TCP bidireccional disponible hasta el momento, esto es, Tahoe TCP bidireccional (clase Agent/TCP/FullTcp).

De todos los parámetros que rigen el comportamiento de TCP, fijamos dos: el crédito inicial y el tamaño de segmento. Estos dos parámetros tendrán los mismos valores sin que importe si las entidades TCP rigen conexiones de extremo a extremo o en alguna de las dos partes de una conexión bipartida: parte alámbrica y parte inalámbrica.

El valor elegido para el crédito inicial es de 8192 bytes. Este valor ha sido observado en varias conexiones TCP hechas por FTP.

El tamaño de segmento (MSS) será igual a 1460 bytes. El valor surge del tamaño de trama de Ethernet habitual en cuando se usa (1500 bytes), al que se ha de restar el tamaño de las cabeceras TCP/IP (40 bytes).

Elegir agentes bidireccionales ha sido una apuesta de futuro, pues es evidente que su mayor versatilidad hará que desaparezcan los agentes de transporte uni-

direccionales. Hay que decir que los agentes de transporte bidireccional están en aún fase experimental. Esto implica el riesgo de que tengan algún *bug*, pero es un riesgo asumible, pues la herramienta de simulación está en continuo desarrollo e incorporar una nueva versión de TCP bidireccional puede ser tan sencillo como obtener los ficheros que contienen su código fuente y recompilar la herramienta.

7.5.2 Implementación

Antes de crear los agentes, se deberán establecer los valores de los parámetros de crédito inicial y de tamaño de segmento. Las dos siguientes instrucciones realizan lo deseado:

```
Agent/TCP/FullTcp set segsize_ $mss
Agent/TCP/FullTcp set windowInit_ [expr $crédito_inicial/$mss]
```

Los miembros variable `segsize_` y `windowInit_` contienen los valores de tamaño de segmento y de crédito (que permanece fijo y con el valor inicial) respectivamente. El primero de los parámetros se mide en bytes y el segundo en segmentos. Se ha de suponer que las variables `mss` (maximum segment size) y `crédito_inicial` contienen los valores definidos en el apartado anterior.

Nótese que el valor del crédito real será igual o menor que el crédito definido en la variable `credito_inicial` por el truncamiento. Esto ocurre debido a que el crédito se mide en segmentos y, si el tamaño de segmento no es múltiplo del tamaño de crédito que deseamos utilizar, no hay manera de que el crédito real sea el mismo. Hay que elegir entonces entre hacerlo mayor o menor, y se ha optado por la segunda.

7.6 STP

7.6.1 Diseño

El protocolo STP debe ofrecer un servicio de transporte orientado a conexión que se adapte a las características del enlace inalámbrico.

Se ha decidido que realizará tan sólo funciones de apertura/cierre de conexiones, fragmentación/reensamblado y control de flujo haciendo uso además de cabeceras de tamaño reducido si es posible. El control de congestiones y el control de errores no se contarán entre sus funciones.

Para la fragmentación es importante elegir el tamaño de segmento que se utilizará. En el mundo real los segmentos STP se montarán sobre datagramas IP, éstos sobre tramas PPP que a su vez irán sobre tramas RLP. Se utilizará como tamaño de segmento 1500 bytes menos el tamaño de la cabecera de STP, de manera similar a como se hacía con TCP.

El control de flujo se realizará con un algoritmo de ventanas (o créditos) sin retransmisiones, eligiendo un tamaño de ventana tal que, conocidas las características de ancho de banda y retardo del enlace inalámbrico, se consiga transmisión continua de datos. Los 8192 bytes fijados para TCP como valor de crédito inicial (y constante) valen aquí también.

Cabecera: debe contener información sobre puertos de destino y origen, apertura/cierre de conexión, número de secuencia de los datos enviados y renovación del crédito.

Figure 7.1: Interfaz de STP

Interfaz

Los protocolos de transporte orientados a conexión que se implementan en el simulador se valen de servicios ofrecidos por entidades de nivel de red, y ofrecen servicios a entidades de nivel de aplicación. La interfaz entre aplicaciones y agentes de transporte está definido en un apartado específico de la documentación de ns [Fall et al, 1999]. Por el contrario, la interfaz entre el nivel de red y el nivel de transporte no está claramente definido en ningún apartado.

Se presenta en este apartado el interfaz entre el protocolo STP y los niveles con los que interactúa, mediante la figura 7.1 y una breve descripción de los métodos incluidos.

La cabecera de STP ocupa 8 bytes organizados de la siguiente manera: 4 bytes para los puertos de entrada y salida 28 bits para los números de secuencia, y cuatro bits para codificar los indicadores de apertura de conexión (syn), cierre de conexión (fin), datos (data), y renovación de crédito (ack).

recv(bytes): indica a una aplicación la recepción de cierta cantidad de datos medida en bytes.

resume(): indica a una aplicación que la conexión ha terminado. NOTA: la semántica original en ns no es esta. Ha sido cambiada por conveniencia a la hora de implementar otros elementos de las simulaciones como el Proxy y el generador de tráfico HTTP. Más detalles sobre esto pueden encontrarse en la implementación de estos elementos.

send(bytes): indica a un agente de transporte que debe enviar cierta cantidad de bytes. Si este valor es -1, el agente debe actuar como si continuamente recibiera datos para enviar.

sendmsg(bytes,flags): indica a un agente de transporte que debe enviar cierta cantidad de bytes. El parámetro flags se utiliza actualmente sólo para indicar un cierre implícito de la conexión.

close(): indica a un agente de transporte que debe cerrar la conexión establecida con otro agente de transporte.

listen(): indica a un agente de transporte que debe prepararse para recibir peticiones de conexión. En la práctica las conexiones entre agentes de transporte son cosa de dos. Además las aplicaciones que se probarán utilizan conexiones únicas con cada par de agentes. Teniendo en cuenta estos dos factores, `listen()` indicará a un agente de transporte que debe prepararse para recibir una petición de conexión.

recv(packet, handler): indica a un agente de transporte la recepción de un paquete.

send(packet, connector): indica a la red que se debe enviar un paquete.

Figure 7.2: Máquina de estados de STP.

Máquina de estados

El autómata que define el comportamiento del protocolo STP se presenta en la figura 7.2, seguido breve explicación.

CLOSED: se considera cerrada toda conexión. Una llamada a `send()` o `sendmsg()` se considera apertura implícita de conexión con el agente de transporte al que está asociado. Se envía entonces la petición de apertura (syn) incluyendo el primer envío de datos en el paquete. Si la petición de envío incluye un cierre implícito de conexión (eof), se pasa al estado SYN_WAIT2, en caso contrario se pasa al estado SYN_WAIT1. Una llamada a `listen()` provoca una transición al estado de escucha (LISTEN).

SYN_WAIT1: Se está en espera de recibir la confirmación de apertura de conexión (syn) para pasar al estado ESTABLISHED. Se aceptan peticiones de envío, que en caso de incluir un cierre implícito (eof) provocan una transición al estado SYN_WAIT2.

SYN_WAIT2: Se está en espera de recibir la confirmación de apertura de conexión (syn) para pasar al estado FINISHING.

LISTEN: Se está en espera de recibir petición de conexión (syn) para pasar al estado ESTABLISHED. Una llamada a close() o la recepción de un paquete en el que se indique apertura y cierre de conexión (syn+fin) provoca una transición al estado CLOSED y el envío de confirmación de cierre (fin).

ESTABLISHED: Se reciben y envían paquetes (mientras haya crédito). Los paquetes recibidos pueden incluir datos y/o renovación de crédito. En caso de incluir petición de cierre de conexión, se pasa al estado CLOSED (enviando la confirmación de cierre) si se han recibido todos los paquetes (se comprobará mediante los números de secuencia), sino se pasa a estado REMAINING.

FINISHING: Se ha recibido una petición de cierre implícita y por tanto no se aceptarán nuevas peticiones de envío. Se permanece en este estado recibiendo paquetes de igual manera que en el estado ESTABLISHED, hasta que se ha enviado todos los datos que quedaban pendientes, momento en el que se invoca `close()` para provocar una transición al estado FIN_WAIT.

FIN_WAIT: Se ha enviado una petición de cierre y se está a la espera de recibir la confirmación para pasar al estado CLOSED. Se acepta en todo momento la recepción de datos y renovación de crédito, pero no peticiones de envío.

REMAINING: Se ha recibido un paquete con fin de mensaje, pero aún quedan paquetes por llegar cuya emisión fue anterior a aquel. Mientras se reciben datos y queden paquetes por recibir (se comprueba mediante los números

de secuencia), se permanece en este estado. Cuando todos los datos han sido recibidos, se pasa a estado CLOSED.

7.6.2 Implementación

La implementación de STP puede encontrarse en los apartados dedicado a los ficheros `ns/tcp-stp.h`, `ns/tcp-stp.cc` y `ns/tcl/lib/ns-default.tcl`.

7.7 Recogida de Resultados

7.7.1 Diseño

Las medidas que se utilizarán para medir el rendimiento de los sistemas en las diferentes situaciones, serán dos clásicos en este tipo de estudios: el throughput y el goodput.

Throughput: mide la cantidad de información por tiempo de comunicación que se recibe a nivel de aplicación. Da una idea de la calidad de servicio que el usuario percibe. Este parámetro se medirá en Kbps.

Goodput: mide la eficiencia en el uso de los enlaces de una comunicación como el cociente entre la cantidad información de usuario recibida y la cantidad de datos que son insuflados en los enlaces. Este parámetro se medirá en tanto por uno.

En estas medidas influyen factores como tamaños de cabecera y número de retransmisiones, factores que serán diferentes según el sistema que se utilice. Por lo tanto su uso se considera muy adecuado para medir cuán eficientes son los diferentes sistemas que se van a probar.

El throughput y el goodput se recogerán en dos formas que se llamarán acumulado y periódico.

Acumulado significa que en cada instante de tiempo se mide el valor medio del parámetro entre el comienzo de la comunicación y ese instante.

Periódico significa que en cada instante de tiempo se mide el valor del parámetro entre ese instante y el instante de tiempo correspondiente al último registro del parámetro.

Además, cuando en las situaciones que se use el modelo indirecto de comunicación, se medirá también los valores de estos parámetros en cada uno de los dos enlaces: alámbrico e inalámbrico.

7.7.2 Implementación

Para poder medir el throughput y el goodput es necesario recoger medidas de los bytes recibidos por las aplicaciones. Por esto, todas las clases de aplicación utilizadas han sido modificadas para que incluyan una variable donde almacenar los bytes recibidos, y el código necesario para ir actualizando el valor de dicha variable.

Las clases afectadas han sido Application/Telnet y Application/TelnetServer (implementadas ambas en `ns/telnet.h` y `ns/telnet.cc`), Application/FTP (implementada en `ns/tcl/lib/ns-source.tcl`), Application/Proxy (implementada en `ns/proxy.h` y `ns/proxy.cc`), y WWWClient y WWWServer (implementadas ambas en `httpModel.tcl`).

Ha sido creada además una clase Application/Monitor (implementada en `ns/monitor.h` y `ns/monitor.cc`). Dicha clase tiene como única función ser sumidero de aplicaciones fuente como Telnet y FTP. La clase ha sido definida como subclase de la clase Application y se vale del método `recv(int nbytes)` para contar los bytes que recibe.

La variable que recoge el número de bytes recibidos ha tomado el nombre `bytes_recibidos_` en todos los casos, salvo en el proxy, donde se ha hecho distinción entre los bytes recibidos en cada mundo y las variables han sido nombradas `bytes_recibidos_mundo1_` y `bytes_recibidos_mundo2_`

Se han incluido además mecanismos similares para recoger la cantidad de bytes enviados por si fuera necesario en un futuro. Las mismas clases han sido afectadas y los nombres de las variables son los mismos pero cambiando *enviados* por *recibidos*.

En el fichero `definiciones.tcl` se incluye la definición de procedimientos que se encargan de recolectar las diferentes medidas definidas. Para más detalle sobre esto y sobre la implementación de los mecanismos de recogida de parámetros consúltese la sección dedicado a las definiciones comunes y el capítulo 9.

7.8 Definiciones Comunes

Para simplificar la definición de cada una de las simulaciones que se realizarán se ha creado un fichero (`definiciones.tcl`) que incluye definiciones comunes a todas ellas. La topología, procedimientos para instalar el proxy, el protocolo STP o alguna de las comunicaciones (Telnet, FTP o WWW) así como procedimientos de registro de parámetros están incluidos en este fichero. Se presenta a continuación una descripción de estas definiciones comunes.

7.8.1 Topología

La topología, en lo que se refiere a nodos y enlaces, es similar en todos los sistemas que se desea simular. Hay tres nodos cliente, servidor e ISP (también llamado RAS, por Remote Access Server) y los dos enlaces alámbrico e inalámbrico definidos en una sección anterior, que unen respectivamente a cliente con ISP y a éste con el servidor.

La única diferencia, en lo que respecta a la topología, entre los diferentes sistemas que se simularán está cuando se desee mantener constante el retardo del enlace inalámbrico. Al ejecutar el fichero de definiciones se instalará la topología con los enlaces definidos de la manera en la que han sido descritos en una anterior sección. En caso de que la variable `retardo_cte` tenga valor distinto de cero al ejecutar el fichero, el retardo inalámbrico se comportará como si permaneciera siempre en el estado libre de errores.

7.8.2 Transporte

Para instalar el sistema indirecto de comunicación se define un procedimiento llamado `instalar_proxy`. Este procedimiento instala un Proxy en el nodo que representa al ISP que actuará como intermediario en toda conexión entre cliente y servidor.

Por defecto se utiliza TCP en ambas partes de la conexión. Si se desea sustituir TCP por STP en la parte del enlace inalámbrico, basta con invocar `instalar_stp` tras la instalación del proxy.

Los parámetros de los agentes de transporte se definen en un procedimiento global cuya ejecución se produce durante la ejecución del fichero de definiciones comunes. El nombre de dicho procedimiento es `inicializar_parametros_transporte`.

7.8.3 Comunicaciones

Para poder instalar las tres diferentes comunicaciones que se probarán se han definido tres procedimientos. Sus nombres son `instalar_telnet`, `instalar_ftp` e `instalar_www` e instalan respectivamente las comunicaciones tipo Telnet, tipo FTP y tipo WWW ya definidas anteriormente.

Instalar una aplicación implica crear las entidades necesarias de aplicación y transporte, hacer las correspondientes conexiones y programar el comienzo de la comunicación.

Los parámetros más importantes de las comunicaciones son la duración máxima de la comunicación (variable global `duracion_comunicacion`), el número de peticiones que efectuará el cliente Telnet (variable global `peticiones_telnet`), el tamaño de fichero a transmitir por FTP (variable `global tamano_fichero`) y el número de nuevas páginas (no en la cache) que solicitará el cliente web (global `variable peticiones_WWW`).

7.8.4 Recogida de resultados

Los procedimientos de registro de los parámetros definidos en una sección anterior han sido incluidos en el fichero de definiciones comunes. En los ficheros que definan las simulaciones sólo restará programar su ejecución en el tiempo deseado. El instante de comienzo debería ser tomado de la variable `comienzo_registro`.

Los procedimientos tienen los siguientes nombres y funciones:

registrar_throughput_acumulado: registra el throughput de extremo a extremo de la comunicación y en ambos sentidos, esto es, *suma* el throughput del cliente y el servidor (esto sólo tiene importancia en el caso de la comunicación WWW, donde ambos cliente y servidor reciben datos). Mide tanto la evolución del throughput como su valor por intervalos, almacenando los resultados en ficheros diferentes.

registrar_throughput_alambrico: como `registrar_throughput`, pero midiendo únicamente en el enlace alámbrico.

registrar_throughput_inalambrico: como `registrar_throughput`, pero midiendo únicamente en el enlace inalámbrico.

registrar_goodput_acumulado: registra el goodput de extremo a extremo de la comunicación y en ambos sentidos, esto es, *suma* el goodput del cliente y el servidor (esto sólo tiene importancia en el caso de la comunicación WWW, donde ambos, cliente y servidor reciben datos). Mide tanto la evolución del goodput como su valor por intervalos, almacenando los resultados en ficheros diferentes.

registrar_goodput_alambrico: como `registrar_goodput`, pero midiendo únicamente en el enlace alámbrico.

registrar_goodput_inalambrico : como `registrar_goodput`, pero midiendo únicamente en el enlace inalámbrico.

Se ha añadido además el procedimiento `registrar_retardo` que registra el retardo del enlace inalámbrico en ambos sentidos, por si fuera necesario hacer comprobaciones del comportamiento de este enlace.

Además se han añadido instrucciones que instalan el método `resume` de las clases Application/Telnet, WWWClient y FTP para que cuando finalice la comunicación de cada tipo, se provoque un último registro de los parámetros (procedimiento `ultimo_registro`) y se finalice (invocando el procedimiento `finalizar`).

El procedimiento finalizar se encarga de cerrar ficheros, de provocar el volcado de los datos de monitorización y traza en los ficheros que corresponda, y de invocar la ejecución de procedimientos de visualización de resultados como el procedimiento `informe`. Este procedimiento escribe por la salida estándar los bytes recibidos y enviados, junto con el throughput del cliente y el servidor. Además presenta el throughput global y el goodput de extremo a extremo y de cada enlace (en caso de haber utilizado el sistema indirecto de comunicación).

NOTA: Al realizar los experimentos se ha comprobado que la carga que supone la ejecución periódica de los procedimientos de recogida de resultados es significativa y afecta a los resultados. Se han modificado por tanto los procedimientos para que no se reprogramen. Activar y desactivar la periodicidad en la ejecución de los procedimientos de recogida de resultados es algo simple (ver apartado dedicado al fichero `definiciones.tcl` en el capítuloCambios.

Chapter 8

Simulaciones

8.1 Introducción

En el capítulo 6 se ha discutido sobre determinadas situaciones en el mundo de las redes de computadores. Concretamente se ha hablado sobre aquellas en las que clientes móviles acceden a través de una red inalámbrica a un servidor fijo (situado en una red terrestre). Se ha hablado sobre la inconveniencia de usar el sistema actual y se han presentado alternativas que teóricamente debieran ser mejores, esto es, más eficientes, que aprovechen mejor los recursos disponibles. Como aval de este estudio, se presentan en el presente documento los resultados obtenidos mediante simulaciones basadas en los sistemas descritos.

Dichos sistemas han sido modelados de la manera más adecuada que ha sido posible e implementados en el simulador de redes ns: UCB/LBNL Network Simulator [Network Research Group, 1998]. Para ello, diferentes elementos han debido ser diseñados e implementados, integrándolos en la herramienta de simulación de redes, bien por no existir en ésta o bien porque los existentes no resultaban adecuados. Estos elementos han sido enlaces de retardo variable, diferentes tipos comunicación, un proxy, un nuevo protocolo de transporte llamado STP y la recogida de resultados. El diseño e implementación de todos estos elementos puede encontrarse en el capítulo 7. Además el capítulo 9 están recogidos los cambios en el código de la herramienta de simulación que han sido necesarios para implementar estos elementos.

A sabiendas de que en el mundo real son diferentes los tipos de comunicación que se dan y de que probar únicamente un tipo de comunicación podría llevar a resultados engañosos, pues la bondad de un sistema de comunicación depende en gran medida del tipo de comunicación, se ha decidido utilizar varias comunicaciones en las simulaciones. Éstas han sido tres: comunicación tipo FTP, comunicación tipo Telnet y comunicación tipo WWW. La elección es debida a su diversidad, y a que son las más habituales hoy en día. Han sido consideradas como parte de los elementos necesarios y por ello, su caracterización e implementación han sido incluídos en el mismo capítulo que el resto de estos elementos. En el presente capítulo hay una sección dedicado a las diferentes sesiones de cada tipo de comunicación que han sido utilizadas durante los experimentos.

Los resultados de cada simulación se espera sean indicadores de la bondad

del sistema en cada diferente situación (determinada por el tipo de comunicación utilizado y por los parámetros de la comunicación). Los parámetros medidos son el throughput y el goodput. Su definición y el diseño e implementación de los mecansismos de recogida de estos parámetros es algo que forma parte del capítulo 7.

8.2 Sesiones

Los tres tipos de comunicaciones implementadas para las simulaciones (Telnet, FTP y WWW) son parametrizables. Se han elegido diferentes configuraciones de cada tipo de comunicación, reflejando diferentes situaciones habituales en el mundo real.

Cada sesión ha sido simulada una única vez. Lo ideal hubiera sido hacier varias simulaciones de una misma sesión y obtener promedios (ver capítulo 10), pero debido a que la aleatoriedad es *fija* (las semillas de los generadores de números a aleatorios tienen siempre el mismo valor) siempre se obtienen los mismos resultados.

8.2.1 Sesiones Telnet

Se han distinguido tres tipos de sesiones Telnet: sesión Telnet corta, sesión Telnet media y sesión Telnet larga.

Una sesión Telnet corta pretende emular un acceso corto en duración (aproximadamente dos minutos). Un acceso de este tipo puede ser el de un usuario a su buzón de correo para ver si tiene algún mensaje nuevo, leerlo y terminar.

Una sesión Telnet media pretende emular un acceso de duración media (aproximadamente veinte minutos). Un ejemplo de este tipo de acceso es el de un usuario que pretende escribir un mensaje de correo.

Una sesión Telnet larga pretende emular un acceso de larga duración (más tres horas). En un acceso de este tipo, el usuario pasa un buen rato conectado en su cuenta, escribiendo mensajes de correo, editando ficheros, etc.

El parámetro diferenciador de las tres sesiones es el número de peticiones que realiza el cliente Telnet. En el primer caso es de 50 peticiones, en el segundo de 500 peticiones, y en el tercero, 5000 peticiones.

8.2.2 Sesiones WWW

Las sesiones WWW que se han probado han sido tres: sesión WWW corta, sesión WWW media y sesión WWW larga.

Una sesión WWW corta emula un acceso típico a páginas web en el que el usuario sólo consulta un par de páginas para ver si hay alguna novedad, etc.

Una sesión WWW media emula un acceso web de duración media como cuando se accede a la edición electrónica de un periódico, etc.

Una sesión WWW larga emula un acceso más largo en el que un cliente está un buen rato navegando por la web.

El parámetro diferenciador de los tipos de sesión es el número de peticiones de páginas nuevas que efectúa el cliente. En el primer caso se ha fijado en 2 páginas, en el segundo en 10, y en el tercero en 18. Debe tenerse en cuenta que las páginas no son planas, esto es, están compuestas de más de un objeto (texto,

Figure 8.1: Escenarios: retardo constante vs retardo variable.

Figure 8.2: Efectos de la variabilidad sobre comunicaciones Telnet.

gráficos, etc.) para cuya descarga se utiliza una conexión independiente (para más detalles, consúltese el capítulo 7.

8.2.3 Sesiones FTP

Se han probado cuatro tipos de comunicación FTP diferenciadas por el tamaño de fichero. Los tamaños de fichero probados han sido 10 MB, 1 MB, 100 KB, 10 KB y 1 KB.

8.3 Efectos de la Variabiliad

8.3.1 Objetivo

El objetivo de este bloque de medidas es comprobar que la variabilidad del retardo en el enlace inalámbrico es realmente causa de pérdida de eficiencia por las razones ya discutidas en el capítuloEntornos Hibridos.

8.3.2 Escenarios

Para ver el efecto que tiene la variabilidad sobre las sesiones previamente definidas, éstas han sido probadas sobre los dos escenarios cuyo esquema puede observarse en la figura 8.1.

En ambos, la comunicación entre el cliente móvil y el servidor fijo es directa (sin proxy), y tiene lugar sobre los dos enlaces definidos en el capítulo 7. La diferencia está en el que el primer escenario se ha anulado la variabilidad del retardo del enlace inalámbrico (enlace entre cliente e ISP) haciendo que éste permanezca siempre constante con el valor correspondiente al estado libre de errores del enlace. NOTA: en caso de dudas, consúltese el capítulo 7.

8.3.3 Resultados

Comunicación Telnet

En la gráfica de la figura 8.2 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes sesiones Telnet. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

El valor del throughput no es demasiado interesante en esta comparación, pues es evidente que tal como se ha modelado la variabilidad, el tiempo de transmisión aumentará si se prueban las simulaciones introduciendo la variabilidad en el retardo del enlace inalámbrico. Si no resulta tan evidente el aumento en el tiempo de transmisión (y el consiguiente descenso en el troughput), piénsese en el peso que tiene el retardo de propagación en una comunicación Telnet y que la

Figure 8.3: Efectos de la variabilidad sobre comunicaciones FTP.

Figure 8.4: Efectos de la variabilidad sobre comunicaciones WWW.

variabilidad implementada introduce un aumento en el retardo de propagación medio.

Así pues, lo que másinteresa es lo que sucede con el goodput. Este parámetro nos da una idea de la eficiencia con que se usan los elances.

Puede observarse un notable descenso del goodput en los tres tipos de sesión. Puesto que el sistema utilizado es en ambos escenarios el mismo, esto es, conexiones TCP de extremo a extremo, el único motivo de descenso de la eficiencia en el segundo escenario es que en éste ha habido necesidad de retransmitir ciertos paquetes. Puesto que el enlace está libre de errores y de congestiones, la causa de las retransmisiones no puede ser sino la variabilidad en el retardo, que provoca retardos suficientemente altos como para disparar los mecanismos de control de la congestión de TCP.

Se confirma, por tanto, que la variabilidad en el retardo del enlace inalámbrico supone una pérdida del rendimiento en comunicaciones tipo Telnet.

Comunicación FTP

En la gráfica de la figura 8.3 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes pruebas de transmisión de ficheros. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

Las transmisiones de ficheros pequeños (entre 1 KB y 100 KB) parecen haber sido lo suficientemente cortas en duración como para no verse afectadas por la varibilidad en el retardo. En estos casos se han obtenidos los mismos valores, tanto throughput como de gooput, en ambos escenarios.

La transmisión de ficheros de mayor tamaño (1 MB y 10 MB) sí ha sufrido en cambio los efectos de la variabilidad, que se han visto reflejados en ligeros descensos en el goodput y en el throughput.

El efecto de la variabilidad en el enlace inalámbrico sobre comunicaciones FTP, no parece ser notablemente negativo, llegando a ser nulo en transmisiones de ficheros menores de 100 KB.

Comunicación WWW

En la gráfica de la figura 8.4 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes sesiones WWW. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

El descenso del throughput en todas las sesiones (y el consiguiente aumento de la duración de la sesión) es evidente.

En el caso de la sesión de corta duración, y a la vista de lo que sucede con el goodput, que no ha sufrido ningún descenso, el descenso en el throughput se debe al aumento del retardo medio que introduce la varibilidad y que en

Figure 8.5: Escenarios: comunicación directa vs comunicación indirecta.

comunicacicones WWW, al igual que en comunicaciones Telnet, tiene gran peso en la duración (y por tanto en el throughput).

El descenso del goodput es notable en el caso de sesiones WWW de duración media y larga. Este hecho, que es prueba de la existencia de retransmisiones, sumado al aumento del retardo provocado por la introducción de la variabilidad en el retardo del enlace inalámbrico, provocan el descenso en el throughput.

La variabilidad en el enlace inalámbrico supone, en conclusión, un descenso del rendimiento en comunicaciones WWW.

Conclusiones

Por lo visto en las diferentes pruebas realizadas, al introducir el factor de variabilidad en el enlace inalámbrico, el throughput sufre un descenso que es notable en comunicaciones WWW y Telnet. En comunicaciones FTP, este descenso es poco apreciable y sólo afecta a transmisiones de ficheros de gran tamaño (más de 1 MB).

El aumento del retardo medio, que implica la variabilidad en el retardo del enlace inalámbrico, es causa del descenso del throughput, en mayor o menor medida según el caso.

El comportamiento del goodput acompaña al del throughput en la mayoría de los casos. La única excepción es en comunicaciones WWW de corta duración, donde el gooput no desciende, pero el throughput sí (debido al aumento del retardo medio).

En los casos en los que el goodput desciende, debe deducirse la existencia de retransmisiones provocadas por los mecanismos de control de la congestión de TCP. No puede haber otra causa del descenso del goodput, pues en ambos escenarios se utiliza el mismo modelo de comunicación, esto es, conexiones TCP de extremo a extremo.

Como conclusión se dirá que la variabilidad en el retardo del enlace inalámbrico conlleva, en general, un descenso en el rendimiento de la comunicaciones.

8.4 Efectios del Uso del Modelo Indirecto

8.4.1 Objetivo

El objetivo de este bloque de medidas es comprobar que el uso de conexiones indirectas supone una mejora en el rendimiento frente al uso de conexiones directas.

8.4.2 Escenarios

Las sesiones que se han utilizado para comparar el rendimiento de los dos sistemas serán las mismas que en el anterior bloque de medidas.

Los escenarios, cuyo esquema puede verse en la figura 8.5, y sobre los que se han ejecutado las comunicaciones, son aquel del bloque anterior en el que el enlace entre el cliente y el ISP sufría variaciones en el retardo, y otro que se

Figure 8.6: Efectos del uso del modelo indirecto sobre comunicaciones Telnet.

Figure 8.7: Efectos del uso del modelo indirecto sobre comunicaciones FTP.

diferencia de éste en que la conexión entre cliente y servidor es indirecta. Como elemento intermediario en la conexión virtual, actúa un Proxy sito en el nodo correspondiente al ISP.

8.4.3 Resultados

Comunicación Telnet

En la gráfica de la figura 8.6 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes sesiones Telnet. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

Los valores de goodput que se presentan son el goodput total en el caso de las pruebas sobre el primer escenario, y el goodput de los enlaces alámbrico e inalámbrico en el caso de las pruebas sobre el segundo escenario.

Puede verse que en todos los casos el goodput medido en el enlace alámbrico es mayor que el medido en el enlace inalámbrico, y que el goodput total del primer escenario permanece entre ambos. Esto es un dato que confirma que los problemas de eficiencia se localizan en la parte inalámbrica de la comunicación.

El throughput aumenta en todos los casos con el uso de comunicaciones indirectas, por lo que debe concluirse que su efecto es un aumento en el rendimiento de comunicaciones de tipo Telnet.

Comunicación FTP

En la gráfica de la figura 8.7 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes pruebas de transmisión de ficheros. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

Los valores de goodput que se presentan son el goodput total en el caso de las pruebas sobre el primer escenario, y el goodput de los enlaces alámbrico e inalámbrico en el caso de las pruebas sobre el segundo escenario.

Se puede observar un ligero aumento en el throughput en la mayoría de los casos.

El goodput del enlace inalámbrico no es apreciablemente menor al goodput del enlace alámbrico. Esto confirma lo dicho en el anterior bloque de medidas, y es que la variabilidad en el retardo no parece tener mucha influencia en el rendimiento de las comunicaciones FTP.

Como conclusión se dirá que el uso del modelo indirecto supone una ligera mejora frente al uso de comunicaciones directas, cuando se trata de transferencias de ficheros.

Figure 8.8: Efectos del uso del modelo indirecto sobre comunicaciones WWW.

Figure 8.9: Escenarios: TCP vs STP.

Comunicación WWW

En la gráfica de la figura 8.8 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes sesiones WWW. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

Los valores de goodput que se presentan son el goodput total en el caso de las pruebas sobre el primer escenario, y el goodput de los enlaces alámbrico e inalámbrico en el caso de las pruebas sobre el segundo escenario.

Los resultados en este tipo de comunicaciones parecen más verosímiles y esperanzadores. En ambos tipos de sesión, el throughput aumenta, y también en ambos casos el goodput de los enlaces alámbricos es superior al de los enlaces inalámbricos dando una nueva prueba sobre la localización de los problemas.

Las conclusiones no pueden ser otras sino que el uso del modelo indirecto supone una mejora en comunicaciones WWW.

Conclusiones

El uso del comunicaciones indirectas tiene como efecto una mejora en el rendimiento de las comunicaciones que es menor en el caso de transmisiones de ficheros, y más notable en el caso de comunicaciones WWW y Telnet.

Además se confirma que los problemas de ineficiencia se localizan en el enlace inalámbrico.

8.5 Effectios de la Sustitución de TCP por STP

8.5.1 Objetivo

El objetivo de este bloque de medidas es demostrar que el uso de un protocolo de transporte sencillo caracterizado básicamente por no realizar control de congestión (protocolo definido en [Elementos Necesarios] y bautizado como STP) como sustituto de TCP en la parte móvil de la conexión, esto es, entre el cliente móvil y el proxy, supone una mejora del rendimiento frente al uso de TCP en ambas partes de la comunicación usando el modelo indirecto.

8.5.2 Escenarios

Las comunicaciones que se han utilizado para comparar el rendimiento de los dos sistemas serán las mismas que en el anterior bloque de medidas.

En la figura 8.9 puede verse un esquema de los escenarios que se comparan. La diferencia entre los dos está en el protocolo de transporte usado en la conexión entre el cliente móvil y el Proxy. Mientras en el primer escenario, el protocolo utilizado es TCP, en el segundo se utiliza el protocolo STP.

Figure 8.10: Efectos de la sustitución de TCP sobre comunicaciones Telnet.

Figure 8.11: Efectos de la sustitución de TCP sobre comunicaciones FTP.

8.5.3 Resultados

Comunicación Telnet

En la gráfica de la figura 8.10 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes sesiones Telnet. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

El goodput que se presenta es únicamente el del enlace inalámbrico, pues se espera observar una mejora del rendimiento en este enlace.

Se puede comprobar, viendo el notable aumento del goodput del enlace inalámbrico, que el protocolo STP, sustituto de TCP en este enlace, hace un uso más eficiente de él. A este aumento en la eficiencia con que se usa el enlace inalámbrico, le acompaña un aumento en el throughput (que conlleva una menor duración de las sesiones) que es menor a media que la duración de la sesión aumenta.

Se debe decir, por tanto, que la sustitución de TCP por el protocolo STP, aumenta el rendimiento de comunicaiones Telnet.

Comunicación FTP

En la gráfica de la figura 8.11 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes pruebas de transmisión de ficheros. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

El goodput que se presenta es únicamente el del enlace inalámbrico, pues se espera observar una mejora del rendimiento en este enlace.

Los resultados obtenidos en estas pruebas de transmisón de ficheros son realmente esperanzadores. Tanto el throughput como el goodput del enlace inalámbrico aumentan en todos los casos, aunque la diferencia se hace menor con el aumento del tamaño del fichero.

La sustitución de TCP por STP tiene un efecto positivo sobre el rendimiento en comunicaciones FTP.

Comunicación WWW

En la gráfica de la figura 8.12 pueden verse los valores de throughput y goodput recogidos en simulaciones de las diferentes sesiones WWW. Se contrastan en este gráfico las medidas de throughput y goodput obtenidas en los dos escenarios anteriormente mencionados.

El goodput que se presenta es únicamente el del enlace inalámbrico, pues se espera observar una mejora del rendimiento en este enlace.

Figure 8.12: Efectos de la sustitución de TCP sobre comunicaciones WWW.

Los resultados son concluyentes: throughput y goodput del enlace inalámbrico mejoran sensiblemente.

La sustitución de TCP por STP en el enlace inalámbrico aumenta considerablemente el rendimiento de comunicaciones WWW.

Conclusiones

Debe concluirse que el objetivo buscado, esto es, comprobar que la sustitución de TCP en el enlace inalámbrico por un protocolo (STP) específicamente diseñado para este enlace, mejora el rendimiento de las comunicaciones, se cumple claramente en todos los casos.

Además el nuevo protocolo ha demostrado una gran estabilidad en el uso del enlace inalámbrico como puede observarse en los valores de goodput sobre dicho enlace obtenidos en todas las pruebas realizadas.

8.6 Conclusiones

Las pruebas realizadas mediante simulaciones confirman que la variabilidad en el retardo del enlace inalámbrico es, tal como se pensaba, causa de ineficiencia en entornos de acceso a datos en redes fijas a través de redes móviles. Esta ineficiencia parece ser mayor en comunicaciones tipo Telnet y WWW, que en comunicaciones tipo FTP.

El uso del modelo indirecto mejora el rendimiento de las comunicaciones, especialmente las que mayor sufrían por causa de la variabilidad. Las pruebas realizadas para estudiar este sistema de comunicación han permitido comprobar que los problemas de ineficiencia se localizan en el enlace inalámbrico.

Sustituyendo el protocolo de transporte TCP por el STP que, entre otras cosas, no realiza control de la congestión, se observa una notable mejoría en el rendimiento de todos los tipos de comunicación. Esta mejoría tiene precisamente causa en el más eficiente uso del enlace inalámbrico por parte del nuevo protocolo de transporte, que además demuestra un comportamiento altamente estable.

Los resultados obtenidos confirman, pues, las ideas expuestas en el capítulo 6.

Chapter 9

Cambios en ns

9.1 Introducción

Este documento recoge parte del código fuente utilizado para la realización del proyecto Simulación de Redes Híbridas. En este proyecto se ha utilizado la herramienta de simulación de redes ns: UCB/LBNL Network Simulator.

La herramienta de simulación ha debido ser extendida para poder simular los entornos de intercomunicación de datos descritos en el capítulo 6. Dicha extensión a consistido en modificar y añadir nuevo código fuente. En el capítulo 7 se describen todos los elementos cuya implementación ha requerido estos cambios en el código que aquí se recogen.

Las simulaciones realizadas el capítulo 8] para dar aval al estudio presentado en el capítulo 6 han supuesto escritura de código que también es recogido aquí.

Otro código que se incluye en este capítulo es la implementación de las simulaciones presentadas en el capítuloSimulaciones Sencillas.

El código ha sido agrupado en tres categorías, a cada una de las cuales se ha dedicado un capítulo. Las categorías son: código original modificado, código ajeno incorporado y modificado, y código incorporado. Dentro de cada capítulo hay un apartado dedicado a cada fichero y cuyo nombre es el nombre del fichero incluyendo el directorio a partir de donde está instalada la herramienta. En el caso de ficheros de simulaciones que pueden ser situados en cualquier directorio, sólo aparece el nombre del fichero.

El código fuente se presenta con un formato diferente al del texto normal para facilitar su lectura.

Todo el código presentado puede encontrarse en el disco que forma parte del conjunto de entregables del proyecto. Debe tenerse en cuenta que puede ser que en este documento el código contenga más comentarios que el contenido en el disco.

9.2 Código original modificado

Se recoge aquí todo el código original de ns que ha sido modificado. Partes de código no modificadas, esto es, procedimientos, definiciones, métodos, etc. no modificados no se incluyen. Los puntos suspensivos indican precisamente la existencia de partes de código no modificadas.

9.2.1 ns/lib/tcl/ns-lib.tcl

Este fichero contiene definiciones de métodos del simulador. Ha sido modificado para incluir código relacionado con el Proxy.

El primer cambio ha consistido en añadir un procedimiento que instala un proxy en un nodo utilizando los tipos de protocolo de transporte indicados como argumentos, y otro que lo desinstala.

```
...
Simulator instproc install-proxy {node t1 t2} {
#Crea un proxy en el nodo node usando t1 como protocolo de transporte fuente
#y t2 como protocolo de transporte destino
    $self set Proxy_ [new Application/Proxy]
    $self set ProxyNode_ $node
    $self set EnableProxy_ 1
    $self set ProxyTClass1_ $t1
    $self set ProxyTClass2_ $t2
}

Simulator instproc uninstall-proxy {} {
    $self set EnableProxy_ 0
}...
```

El siguiente cambio introducido ha sido modificar la implementación del procedimiento connect que realiza la conexión entre agentes de transporte, para que realice una conexión indirecta a través del Proxy en caso de que éste haya sido instalado y que los agentes de transporte estén en nodos adyacents al nodo en el que está el Proxy.

```
Simulator instproc connect {src dst {reverse 0}} {
# A menos que reverse indique lo contrario, src se considera
# perteneciente al mundo 1 y dst al mundo 2
# src es cliente y dst servidor
    $self instvar EnableProxy_
    if {$EnableProxy_==0} {
$self simplex-connect $src $dst
$self simplex-connect $dst $src
return $src
    }
    $self instvar ProxyNode_
    #Direccion de un agente:
    #8 bits superiores -> nodo
    #8 bits inriores -> agente
    set srcnode [expr [$src set addr_]/256]
    set dstnode [expr [$dst set addr_]/256]
    set link1 [$self getlink [$ProxyNode_ id] $dstnode]
    set link2 [$self getlink $srcnode [$ProxyNode_ id]]
    if {$link1==-1 || $link2==-1} {
$self simplex-connect $src $dst
$self simplex-connect $dst $src
return $src
    } else {
$self instvar Proxy_
#T1 se concectara a la fuente; hara listen si es FullTCP
$self instvar ProxyTClass1_
$self instvar ProxyTClass2_
if {$ProxyTClass1_!="Agent/TCP/FullTcp" && $ProxyTClass1_!="Agent/STP" && $reverse==0} {
    set duplex 0
    set PTF Agent/TCPSink
} elseif {$reverse==0} {
    set duplex 1
    set PTF $ProxyTClass1_
}
if {$ProxyTClass2_!="Agent/TCP/FullTcp" && $ProxyTClass2_!="Agent/STP" && $reverse==1} {
    set duplex 0
    set PTF Agent/TCPSink
} elseif {$reverse==1} {
    set duplex 1
    set PTF $ProxyTClass2_
}
if {$reverse==0} {
    set T1 [new $PTF]
    set T2 [new $ProxyTClass2_]
} else {
    set T1 [new $PTF]
    set T2 [new $ProxyTClass1_]
}
$self color 100 Yellow
$T1 set fid_ 100
$T2 set fid_ 100
$self attach-agent $ProxyNode_ $T1
$self attach-agent $ProxyNode_ $T2
if {$reverse==0} {
    $Proxy_ attach-agents $T1 $T2
} else {
    $Proxy_ attach-agents $T2 $T1
}
$self simplex-connect $src $T1
$self simplex-connect $T1 $src
$self simplex-connect $T2 $dst
$self simplex-connect $dst $T2
```

```
#Deberia comprobar si es FullTCP
if {$duplex==1} {
    $T1 listen
}
return $src
    }
}
```

9.2.2 ns/lib/tcl/ns-source.tcl

Este fichero contiene definiciones relativas a generadores de tráfico. Ha sido modificado para que los agentes de la clase Application/FTP recogan estadísticas de su actividad. Además se ha añadido código para implemetar un agente FTP que se comporte enviando mensajes con aleatoriedad en intervalo y tamaño. Dicho agente nunca ha llegado a ser utilizado, pero parte de su implementación aparece aquí.

El método constructor ha sido modificado para inicializar las variables de recogida de resultados.

```
..
Application/FTP instproc init {} {
    $self set bytes_enviados_ 0.0
    $self set bytes_recibidos_ 0.0
    $self set tamano [new RandomVariable/Exponential]
    $self set intervalo [new RandomVariable/Exponential]
    $self next
}
...
```

El método `send` se ha modificado para que junto con los bytes se envíe un indicador de fin de mensaje.

```
...
Application/FTP instproc send {nbytes} {
    $self instvar bytes_enviados_
    set bytes_enviados_ [expr $bytes_enviados_ + $nbytes]
    [$self agent] sendmsg $nbytes "MSG_EOF"
}
...
```

9.2.3 ns/lib/tcl/ns-default.tcl

Este fichero contiene instrucciones que crean variables y/o asignan valores por defecto.

Las simulaciones recogidas en el capítulo 5 sólo han requerido la siguiente línea.

```
...
Application/PingPong set stop_ 1
...
```

Las siguientes instrucciones inicializan variables relacionadas con la recogida de resultados.

```
..
Application/Monitor set bytes_recibidos_ 0.0
Application/Monitor set bytes_enviados_ 0.0
Application/Telnet set bytes_recibidos_ 0.0
Application/Telnet set bytes_enviados_ 0.0
Application/Telnet set size_ 64
Application/Telnet set peticiones_ 5000
Application/Telnet instproc resume {} {}
Application/TelnetServer set request_size_ [Application/Telnet set size_]
Application/TelnetServer set bytes_recibidos_ 0.0
Application/TelnetServer set bytes_enviados_ 0.0
Application/TelnetServer set size_ 512
Application/Telnet set response_size_ [Application/TelnetServer set size_]
Application/TelnetServer instproc resume {} {}
Application/FTP set bytes_recibidos_ 0.0
Application/FTP set bytes_enviados_ 0.0
Application/Proxy set bytes_recibidos_mundo1_ 0.0
Application/Proxy set bytes_recibidos_mundo2_ 0.0
Application/Proxy set bytes_enviados_mundo1_ 0.0
Application/Proxy set bytes_enviados_mundo2_ 0.0
Application/Proxy set enableResume_ 1
Application/Monitor set enableResume_ 1
...
```

Las instrucciones que siguen inicializan variables del protocolo de transporte STP.

```
...
Agent/STP set stpip_base_hdr_size_ 28
Agent/STP set seqno_ 0
Agent/STP set segment_size_ 1000
Agent/STP set credit_size_ 2048
Agent/STP set initial_credits_ 4
Agent/STP set debug_ 0
...
```

9.2.4 ns/lib/delay.h y ns/delay.cc

Estos dos ficheros implementan la clase Delay que modela los retardos en los enlaces. Ha sido modificada para que los retardos sean tal y como son definidos en [Elementos Necesarios].

Se ha hecho necesario incluir el fichero que define la clase que implementa los temporizadores y declarar dos nuevas clases de temporizador, pues se utilizarán para implementar las transiciones de estado del enlace.

```
..
.
#include "timer-handler.h"

class timer_error_class;
class timer_libre_class;
...
```

La definición de la clase ha sido modificada para que incluya los mecanismos necesarios para implementar retardos variables.

```
class LinkDelay : public Connector {
 public:
...
//El retardo es ahora  una una variable aleatoria (exponencial).
//delay representa su media
//El metodo delay() debe devolver su valor
//Se establece la media de la distribucion
//Para evitar modificaciones en infinitos sitios
double delay() {
if(cte) return delay_;
else
return delayp->value();
}
inline double txtime(Packet* p) {
return (8. * hdr_cmn::access(p)->size() / bandwidth_);
}
...
void estado_libre();
void estado_error();
 protected:
...
//delay representa ahora representa la media del retardo
double delay_; /* line latency */
...
private:
...
int cte;
RandomVariable *delayp;
ConstantRandomVariable *delay_L;
UniformRandomVariable *delay_E;
ExponentialRandomVariable *L_L;
ExponentialRandomVariable *L_E;
timer_error_class *timer_error;
timer_libre_class *timer_libre;
void inicializar(double t_L, double t_E, double am, double aM);
};
...
```

Los nuevas clases de temporizador han sido definidos en las siguientes líneas.

```
...
class timer_error_class : public TimerHandler {
public:
timer_error_class(LinkDelay *o){o_=o;}
virtual void expire(Event *e);
protected:
LinkDelay *o_;
};

class timer_libre_class : public TimerHandler{
public:
timer_libre_class(LinkDelay *o){o_=o;}
virtual void expire(Event *e);
protected:
LinkDelay *o_;
};
...
```

El constructor inicializa la variable que indica si el enlace permanece constante o variable.

```
LinkDelay::LinkDelay() : Connector(), dynamic_(0), itq_(0), nextPacket_(0)
{
...
cte=1;
}
```

El método `command` ha sido modificado para que a través del intérprete se pueda consultar el valor del retardo y hacer que éste sea variable.

```
...
int LinkDelay::command(int argc, const char*const* argv)
{
if (argc == 2) {
...
if(strcmp(argv[1],"retardo")==0){
Tcl::instance().resultf("%f",delay());
return TCL_OK;
}
} else if (argc==6){
if(strcmp(argv[1],"inicializar")==0){
inicializar(atof(argv[2]),atof(argv[3]),atof(argv[4]),atof(argv[5]));
return TCL_OK;
}
...
}
...
```

El siguiente método ha sido modificado para que no se utilice directamente la variable `delay_`, sino que se consulte el retardo a través del método previamente definido.

```
...
void LinkDelay::recv(Packet* p, Handler* h)
{
...
if (dynamic_) {
...
e->time_ = s.clock() + txt + delay();
..
} else {
...
s.schedule(target_, p, txt + delay());
...
}
...
}
...
```

Las siguientes líneas implementan los temporizadores y el procedimiento que instala la variabilidad en el enlace.

```
...
void timer_error_class::expire(Event *e){
o_->estado_error();
}

void timer_libre_class::expire(Event *e){
o_->estado_libre();
}

void LinkDelay::estado_libre(){
delayp=delay_L;
timer_error->sched(L_L->value());
}

void LinkDelay::estado_error(){
delayp=delay_E;
timer_libre->sched(L_E->value());
}

void LinkDelay::inicializar(double t_L,double t_E,double am, double aM){
cte=0;
L_L=new ExponentialRandomVariable(t_L);
L_E=new ExponentialRandomVariable(t_E);
delay_L=new ConstantRandomVariable(delay_);
delay_E=new UniformRandomVariable(am,aM);
timer_error=new timer_error_class(this);
timer_libre=new timer_libre_class(this);
estado_libre();
}
...
```

9.2.5 ns/telnet.h y ns/telnet.cc

Estos dos ficheros han sido modificados para adecuar la aplicación Telnet a las necesidades.

La clase Telnet ha sido modificada añadiendo miembros para la recogida de resultados y para almacenar los valores de los tamaños de los paquetes.

```
..
.
class TelnetApp : public Application {
 public:
...
double bytes_enviados_;
double bytes_recibidos_;
int peticiones_;
...
 protected:
...
int size_;
int response_size_;
private:
int peticiones;
int bytes_respuesta;
};
...
```

La clase TelnetServer ha sido añadida para implementar servidores Telnet.

```
...
class TelnetServer : public Application {
 public:
TelnetServer();
void recv(int nbytes);
void send(int nbytes);
double bytes_enviados_;
double bytes_recibidos_;
 protected:
int size_;
int request_size_;
 private:
int respuestas;
int bytes_peticion;
};
...
```

Los constructores incluyen inicializaciones de variables.

```
...
TelnetApp::TelnetApp() : running_(0), timer_(this)
{
bind("interval_", &interval_);
//CAMBIO
bind("bytes_enviados_", &bytes_enviados_);
bind("bytes_recibidos_", &bytes_recibidos_);
bind("size_",&size_);
bind("response_size_",&response_size_);
bind("peticiones_",&peticiones_);
peticiones=0;
bytes_respuesta=0;
//FIN CAMBIO
}

static class TelnetServerClass : public TclClass {
 public:
TelnetServerClass() : TclClass("Application/TelnetServer") {}
TclObject* create(int, const char*const*) {
return (new TelnetServer);
}
} class_app_telnet_server;


TelnetServer::TelnetServer()
{
bind("bytes_enviados_", &bytes_enviados_);
bind("bytes_recibidos_", &bytes_recibidos_);
bind("size_",&size_);
bind("request_size_",&request_size_);
respuestas=0;
bytes_peticion=0;
}
...


void TelnetApp::start()
{
bytes_enviados_=0.0;
bytes_recibidos_=0.0;
...
}
...
```

Provocar la detención del cliente Telnet implica ahora cerrar la conexión.

```
..
.
void TelnetApp::stop()
{
...
agent_->close();
}
...
```

Cuando se dispara el temporizador y se emite una petición se cuentan los bytes enviados, y cuando se recibe un paquete se actualizan los bytes recibidos y se comprueba si se ha recibido un respuesta completa para programar la siguiente petición. En caso de haber superado el máximo de peticiones, la aplicación se detiene.

```
void TelnetApp::timeout()
{
if (running_) {
bytes_enviados_+=size_;
agent_->sendmsg(size_);
}
}

void TelnetApp::recv(int nbytes) {
bytes_recibidos_+=nbytes;
bytes_respuesta+=nbytes;
if(bytes_respuesta<response_size_){
return;
}
bytes_respuesta=0;
peticiones++;
if (peticiones>peticiones_){
Tcl::instance().evalf("%s resume", this->name());
}
double t = next();
timer_.resched(t);
}
...

void TelnetServer::recv(int nbytes){
bytes_peticion+=nbytes;
if(bytes_peticion<request_size_){
return;
}
bytes_peticion=0;
respuestas++;
bytes_recibidos_+=nbytes;
send(size_);
}

void TelnetServer::send(int nbytes){
bytes_enviados_+=nbytes;
agent_->send(nbytes);
}
```

9.2.6 ns/agent.cc

Esta clase sirve de base a todas las clases de agente. Ha sido modificada para introducir el cambio en la semántica del método `resume`.

Périodos de inactividad no deben provocar invocaciones a `resume`.

```
...
void Agent::idle()
{
//CAMBIO
//if (app_)
//app_->resume();
//FIN CAMBIO
}
...
```

Si el agente (que será de transporte) invoca su propio método `resume`, se invoca el método `resume` de la aplicación a la que sirve.

```
..
void Agent::resume(){
if (app_){
app_->resume();
}
}
...
```

9.2.7 ns/tcp-full.cc

Se ha descubierto un problema en la implementación de TCP bidireccional y se ha corregido. El problema está en que no se admiten nuevas peticiones de envío si la conexión no está aún abierta. Esto es algo que puede suceder cuando se utilizan conexiones indirectas si una de la conexiones es más rápida que la otra. Además peticiones de envío de cero bytes eran antes ignoradas, sin pensar en que quizá la aplicación esté interesada en los flags. Por ello si se detecta una cantidad de bytes igual a cero se fuerza a enviar paquetes.

```
void FullTcpAgent::advance_bytes(int nb)
{
if (state_ > TCPS_ESTABLISHED) {
...
} else if (state_ == TCPS_ESTABLISHED) {
...
send_much(nb==0, REASON_NORMAL, maxburst_);
} else if (state_ < TCPS_ESTABLISHED) {
closed_ = 0;
if (curseq_ < iss_)
curseq_ = iss_;
curseq_ += nb;
}
```

El método command ha sido modificado para que se acepten llamadas a resume a través del intérprete.

```
int FullTcpAgent::command(int argc, const char*const* argv)
{
if (argc == 2) {
...
if (strcmp(argv[1], "resume") == 0) {
resume();
return (TCL_OK);
}
}
...
}
```

9.3 Código ajeno incorporado y modificado

Se incluye en este capítulo el código realizado por personas ajenas al grupo que ha desarrollado la herramienta ns y con el que han cedido para uso público de forma totalmente desinteresada. Las referencias pueden encontrarse en el capítulo 7.

9.3.1 httpModel.tcl

Este fichero contiene la implementación de un generador de tráfico HTTP que ha sido modificado para adecuarlo a las necesidades. Los principales cambios han sido debidos a la recogida de resultados, y la compatibilidad con conexiones indirectas y con agentes de transportes bidireccionales

```
...
WWWClient set TRANSPORTCL_ Agent/TCP/FullTcp
WWWClient set MAX_REQUESTS_ 10
...

WWWClient instproc init {cnode ns} {
    ...
    WWWClient instvar TRANSPORTCL_
    set tcpType_ $TRANSPORTCL_
    $self set bytes_recibidos_ 0.0
    $self set bytes_enviados_ 0.0
    set requestLength_ 0
    $self set num_requests_ 0
    WWWClient instvar MAX_REQUESTS_
    $self set max_requests_ $MAX_REQUESTS_
}

WWWClient instproc resume {} {
}

WWWClient instproc start server {
    ...
    $self instvar max_requests_
    $self instvar num_requests_
    set num_requests_ [expr $num_requests_+1]
    if {$num_requests_>$max_requests_} {
$self resume
    }
    ...
    $self instvar requestLength_
    ...
    set reqTcp_ [new $tcpType_]
    $reqTcp_ set_callback $self
    $reqTcp_ set_pkttype 28
    #Comento esta linea
    #$reqTcp_ reset ; # Necessary for later connections
    set reqTcpSink_ [new [$server set tcpType_]]
    $reqTcpSink_ set_callback $self $reqTcp_
    ...
    ...
```

```
$ns_ at [expr [$ns_ now] + $think] "$ns_ connect $reqTcp_ $reqTcpSink_; \
    $reqTcpSink_ listen; \
    $reqTcp_ set packetSize_ $requestLength_; \
        $reqFtp_ send $requestLength_"
    incr numTrans_
    $self instvar bytes_enviados_
    set bytes_enviados_ [expr $bytes_enviados_ + $requestLength_]
}

WWWClient instproc timeout tcp {
    # comento las tres siguientes lineas
    #$reqTcp_ reset
    #$ns_ detach-agent $cnode_ $reqTcp_
    #$ns_ detach-agent [$server_ set snode_] $reqTcpSink_
    ...
    $self instvar requestLength_
    $server_ instvar bytes_recibidos_
    set bytes_recibidos_ [expr $bytes_recibidos_ + $requestLength_]
    ...
}
...

WWWServer set TRANSPORTSV_ Agent/TCP/FullTcp

WWWServer instproc init {snode ns} {
    ...
    WWWServer instvar TRANSPORTSV_
    set tcpType_ $TRANSPORTSV_
    $self set bytes_recibidos_ 0.0
    $self set bytes_enviados_ 0.0
}

WWWServer instproc start client {
...
set repTcp_${i}_ [new $tcpType_]
...
#[set repTcp_${i}_] reset
...
set repTcpSink_${i}_ [new [$client set tcpType_]]
[set repTcpSink_${i}_] set_callback $self [set repTcp_${i}_]
...

#Para incorporar la longitud del mensaje enviado
set connList_([set repTcp_${i}_]) [list [set repTcpSink_${i}_] \
[set repFtp_${i}_] $client $replyLength_]
...
#Usamos FullTCP e indicamos mundos
$ns_ connect [set repTcp_${i}_] [set repTcpSink_${i}_]  1
[set repTcpSink_${i}_] listen
...
[set repFtp_${i}_] send $replyLength_
$self instvar bytes_enviados_
set bytes_enviados_ [expr $bytes_enviados_+$replyLength_]
...
}

WWWServer instproc timeout tcp {
    ...
    $client instvar bytes_recibidos_
    set replyLen [lindex $conn 3]
    set bytes_recibidos_ [expr $bytes_recibidos_ + $replyLen]
    ...
    #$ns_ detach-agent $snode_ $tcp
    #$ns_ detach-agent [$client set cnode_] $repTcpSink
    ...
}

Random instproc init seed {
    #Semilla igual para poder comparar
    set seed_ 13
    ...
}

Agent/TCP/FullTcp instproc set_callback {callback {callbacker null}} {
    if {$callbacker=="null"} {
set callbacker $self
    }
    $self set callback_ $callback
    $self instvar callback_
    $self set callbacker_ $callbacker
    $self instvar callbacker_
}

Agent/TCP/FullTcp instproc done {} {
    $self instvar callback_
    $self instvar callbacker_
    if [info exists callback_] {
$callback_ timeout $callbacker_
    } else {
   $self resume
    }
}
Agent/STP instproc set_callback {callback {callbacker null}} {
    if {$callbacker=="null"} {
set callbacker $self
    }
    $self set callback_ $callback
    $self instvar callback_
    $self set callbacker_ $callbacker
```

```
    $self instvar callbacker_
}

Agent/STP instproc done {} {
    $self instvar callback_
    $self instvar callbacker_
    if [info exists callback_] {
$callback_ timeout $callbacker_
    } else {
##$self resume
    }
}
```

9.4 Código incorporado

Se recoge aquí la totalidad del código desarrollado durante el proyecto.

9.4.1 ns/proxy.h y ns/proxy.cc

Estos dos ficheros implementan el agente Proxy.

```
#include "app.h"

#define MAX_IPROXYS 1000

class Proxy;

class IProxy : public Application {
 public:
  IProxy(int identificador, Proxy* p);
  void IProxy::atar(Agent* agente);
  void send(int nbytes);
  void recv(int nbytes);
  void resume();
  void cerrar();
 protected:
  Agent *agent_;
 private:
  int id;
  int cerrado;
  Proxy* proxy;
};

class Proxy : public Application {
 public:
  Proxy();
  void recv(int identificador, int nbytes);
  void cerrar(int identificador);
  int bytes_recibidos_mundo1_;
  int bytes_recibidos_mundo2_;
  int bytes_enviados_mundo1_;
  int bytes_enviados_mundo2_;
 protected:
  int command(int argc, const char*const* argv);
 private:
  int next_id;
  IProxy* IProxys[MAX_IPROXYS];
};


#include "proxy.h"
#include "tcp.h"
#include "tclcl.h"

IProxy::IProxy(int identificador, Proxy* p) {
  id=identificador;
  cerrado=0;
  proxy=p;
}

void IProxy::atar(Agent* agente){
  agent_=agente;
  agent_->attachApp(this);
}

void IProxy::recv(int nbytes){
  proxy->recv(id,nbytes);
}

void IProxy::send(int nbytes){
  agent_->send(nbytes);
}

void IProxy::resume(){
  cerrado=1;
  proxy->cerrar(id);
}

void IProxy::cerrar(){
  if(!cerrado){
```

```
    agent_->sendmsg(0,"MSG_EOF");
    cerrado=1;
  }
}

Proxy::Proxy(){
  bind("bytes_enviados_mundo1_",&bytes_enviados_mundo1_);
  bind("bytes_enviados_mundo2_",&bytes_enviados_mundo2_);
  bind("bytes_recibidos_mundo1_",&bytes_recibidos_mundo1_);
  bind("bytes_recibidos_mundo2_",&bytes_recibidos_mundo2_);
  next_id=0;
}

void Proxy::recv(int identificador, int nbytes){
  if(identificador%2==0){
    bytes_recibidos_mundo1_+=nbytes;
    IProxys[identificador+1]->send(nbytes);
    bytes_enviados_mundo2_+=nbytes;
  }
  else {
    bytes_recibidos_mundo2_+=nbytes;
    IProxys[identificador-1]->send(nbytes);
    bytes_enviados_mundo1_+=nbytes;
  }
}

void Proxy::cerrar(int identificador){
  if(identificador%2==0){
    IProxys[identificador+1]->cerrar();
  }
  else {
    IProxys[identificador-1]->cerrar();
  }
}

static class ProxyClass : public TclClass {
public:
  ProxyClass():TclClass("Application/Proxy"){}
  TclObject* create(int argc, const char*const* argv){
    return(new Proxy);
      }
} class_proxy;

int Proxy::command(int argc, const char*const* argv){
  if(argc==4){
    if(strcmp(argv[1],"attach-agents")==0){
      if(next_id==MAX_IPROXYS){
Tcl& tcl=Tcl::instance();
tcl.resultf("cmd=%s: Proxy sobrecargado",argv[1]);
return TCL_ERROR;
      }
      IProxys[next_id]=new IProxy(next_id,this);
      IProxys[next_id+1]=new IProxy(next_id+1,this);
      IProxys[next_id]->atar((Agent*) TclObject::lookup(argv[2]));
      IProxys[next_id+1]->atar((Agent*) TclObject::lookup(argv[3]));
      next_id+=2;
      return(TCL_OK);
    }
  }
  return (Application::command(argc,argv));
}
```

9.4.2 ns/stp.h y ns/stp.cc

Estos dos ficheros implementan el protocolo de transporte STP.

```
#include "agent.h"
#include "packet.h"


// Definicion de los estados
#define STP_CLOSED 0
#define STP_LISTEN 1
#define STP_SYN_WAIT1 2
#define STP_SYN_WAIT2 3
#define STP_ESTABLISHED 4
#define STP_FINISHING 5
#define STP_FIN_WAIT 6
#define STP_REMAINING 7

// Definicion del maximo numero de bytes enviable para envio infinito
#define STP_MAX_BYTES 1073741824

// Definicion de las razones por las que se solicita envio
#define STP_REASON_SYN 0x0001
#define STP_REASON_DATA 0x0010
#define STP_REASON_MORE_CREDIT 0x0100
#define STP_REASON_FIN 0x1000

//Definiciones relacionadas con los segmentos
#define STP_SEGMENT_SIZE 1024

//Definiciones relacionadas con los creditos
#define STP_INITIAL_CREDITS 3
#define STP_CREDIT_SIZE (2*STP_SEGMENT_SIZE)
```

```
//Definicion de la cabecera
struct hdr_stp {
  int seqno_;
  int flags_;

  int& seqno() { return (seqno_); }
  int& flags() { return (flags_); }
};



class STP: public Agent {
public:
STP();

virtual void sendmsg(int nbytes, const char *flags);
virtual void send(int nbytes) { sendmsg(nbytes, ""); }
virtual void recv(Packet *pkt, Handler *);
virtual void close();
virtual void listen();
virtual int command(int argc, const char*const* argv);

// To make base Tcp happy
virtual void timeout(int) {}

protected:
int seqno_;
int flags_;
int off_stp_;
int stpip_base_hdr_size_;
int segment_size_;
int credit_size_;
int initial_credits_;
int debug_;
private:
int last_seq_rcv;
int last_seq_sent;
int bytes_rcv;
int state;
int sndbuffer;
int rcvbuffer;
int credit;
int consumed;
int done;

void init();
void error(char *message);
void change_state(int new_state);
void flush();
void open_connection();
void close_connection();
void produce_more(int nbytes);
void consume_more(int nbytes);
int is_syn(Packet *pkt);
int is_more_credit(Packet *pkt);
void get_more_credit(Packet *pkt);
int is_data(Packet *pkt);
int data(Packet *pkt);
int is_fin(Packet *pkt);
int seq(Packet *pkt);
void act_seq(Packet *pkt);
int get_bytes();
void send_much(int reason);
void send_packet(int bytes, int flags);
void debug_message(char *message);
void debug_message(char *message,int p1);
void debug_message(char *message,int p1, int p2);
};

#include <stdlib.h>
#include "tclcl.h"
#include "packet.h"
#include "ip.h"
#include "app.h"
#include "stp.h"

class STPHeaderClass : public PacketHeaderClass {
public:
        STPHeaderClass() : PacketHeaderClass("PacketHeader/STP",
     sizeof(hdr_stp)) {}
} class_stphdr;

static class STPClass : public TclClass {
public:
STPClass() : TclClass("Agent/STP") {}
TclObject* create(int, const char*const*) {
return (new STP());
}
} class_stp_agent;

STP::STP(): Agent(PT_STP), seqno_(0)
{
  state=STP_CLOSED;
  bind("seqno_", &seqno_);
  bind("flags_", &flags_);
  bind("off_stp_", &off_stp_);
  bind("stpip_base_hdr_size_", &stpip_base_hdr_size_);
  bind("segment_size_",&segment_size_);
```

```
  bind("credit_size_",&credit_size_);
  bind("initial_credits_",&initial_credits_);
  bind("debug_",&debug_);
  init();
  debug_message("Creando un agente STP...\n");
}

void STP::listen(){
  debug_message("Procesando listen...\n");
  switch(state){
  case STP_CLOSED:
    change_state(STP_LISTEN);
    break;
  default:
    error("La conexion no esta cerrada");
  }
}

void STP::close(){
  debug_message("Procesando close...\n");
  switch(state){
  case STP_CLOSED:
  case STP_REMAINING:
    error("La conexion ya esta cerrada");
    break;
  case STP_SYN_WAIT1:
  case STP_SYN_WAIT2:
    error("La conexion no esta abierta");
    break;
  case STP_FIN_WAIT:
    error("La conexion esta cerrandose");
    break;
  case STP_FINISHING:
  case STP_ESTABLISHED:
    close_connection();
    change_state(STP_FIN_WAIT);
    break;
  case STP_LISTEN:
    flush();
    change_state(STP_CLOSED);
    break;
  default:
    error("Estado no definido");
  }
}

void STP::sendmsg(int bytes, const char *flags){
  debug_message("Procesando sendmsg\n");
  done=0;
  if (bytes == -1) {
    bytes=STP_MAX_BYTES;
  }
  switch(state){
  case STP_LISTEN:
    error("La conexion no esta abierta");
    break;
  case STP_SYN_WAIT2:
  case STP_FIN_WAIT:
  case STP_FINISHING:
    error("La conexion esta cerrandose");
    break;
  case STP_SYN_WAIT1:
    if(flags && strcmp(flags, "MSG_EOF")==0){
      change_state(STP_SYN_WAIT2);
      produce_more(bytes);
    } else {
      produce_more(bytes);
    }
    break;
  case STP_CLOSED:
  case STP_REMAINING:
    if(flags && strcmp(flags, "MSG_EOF")==0){
      change_state(STP_SYN_WAIT2);
      produce_more(bytes);
      open_connection();
    } else {
      change_state(STP_SYN_WAIT1);
      produce_more(bytes);
      open_connection();
    }
    break;
  case STP_ESTABLISHED:
    if(flags && strcmp(flags, "MSG_EOF")==0){
      change_state(STP_FINISHING);
      produce_more(bytes);
    } else {
      produce_more(bytes);
    }
    break;
  default:
    error("Estado indefinido");
  }
}

void STP::recv(Packet *pkt, Handler *){
  debug_message("Procesando recv en estado %d...",state);
  switch(state){
  case STP_CLOSED:
    error("La conexion esta cerrada");
```

```
    break;
  case STP_SYN_WAIT1:
    if(!is_syn(pkt)){
      error("Se esperaba confirmacion de apertura de conexion");
      break;
    }
    change_state(STP_ESTABLISHED);
    if(is_more_credit(pkt)){
      get_more_credit(pkt);
    }
    send_much(STP_REASON_DATA);
    break;
  case STP_SYN_WAIT2:
    if(!is_syn(pkt)){
      error("Se esperaba confirmacion de apertura de conexion");
      break;
    }
    change_state(STP_FINISHING);
    if(is_more_credit(pkt)){
      get_more_credit(pkt);
    }
    send_much(STP_REASON_DATA);
    break;
  case STP_LISTEN:
    if(!is_syn(pkt)){
      error("Se esperaba confirmacion de apertura de conexion");
      break;
    }
    if(is_data(pkt)){
      act_seq(pkt);
      consume_more(data(pkt));
    }
    if(is_fin(pkt)){
      flush();
      change_state(STP_CLOSED);
    } else {
      change_state(STP_ESTABLISHED);
      open_connection();
    }
    break;
  case STP_ESTABLISHED:
  case STP_FINISHING:
    if(is_data(pkt)){
      act_seq(pkt);
      consume_more(data(pkt));
    }
    if(is_more_credit(pkt)){
      get_more_credit(pkt);
    }
    debug_message("Last seq %d, bytes %d",last_seq_rcv,bytes_rcv);
    if(is_fin(pkt) && bytes_rcv==last_seq_rcv){
      flush();
      close_connection();
      change_state(STP_CLOSED);
    } else if (is_fin(pkt)) {
      change_state(STP_REMAINING);
    }
    break;
  case STP_FIN_WAIT:
    if(is_data(pkt)){
      act_seq(pkt);
      consume_more(data(pkt));
    }
    if(is_more_credit(pkt)){
      get_more_credit(pkt);
    }
    if(is_fin(pkt)){
      flush();
      change_state(STP_CLOSED);
    }
    break;
  case STP_REMAINING:
    if(is_data(pkt)){
      act_seq(pkt);
      consume_more(data(pkt));
    }
    debug_message("Last seq %d, bytes %d",last_seq_rcv,bytes_rcv);
    if(bytes_rcv==last_seq_rcv){
      flush();
      close_connection();
      change_state(STP_CLOSED);
    }
    break;
  default:
    error("Estado indefinido");
  }
  Packet::free(pkt);
}

void STP::init(){
  //debug_message("Inicializando...\n");
  rcvbuffer=0;
  sndbuffer=0;
  //debug_message("Creditos iniciales:%d\n",initial_credits_);
  //debug_message("Tamano de credito:%d\n",credit_size_);
  credit=initial_credits_*credit_size_;
  done=1;
  consumed=0;
  last_seq_rcv=0;
```

```
  last_seq_sent=0;
  bytes_rcv=0;
}

void STP::error(char *message){
  debug_message(message);
  debug_message("\n");
  exit;
}

void STP::change_state(int new_state){
  debug_message("Cambiando a estado %d\n",new_state);
  if(new_state==STP_CLOSED){
    init();
  }
  state=new_state;
}

void STP::flush(){
  debug_message("Flushing %d bytes...\n",rcvbuffer);
  if (rcvbuffer>0){
    recvBytes(rcvbuffer);
    rcvbuffer=0;
  }
  debug_message("Resuming...\n");
  //Doble interfaz por las diferentes aplicaciones
  if(done) Tcl::instance().evalf("%s done", this->name());
  resume();
}

void STP::open_connection(){
  debug_message("Solicitando apertura de conexion..\n");
  send_much(STP_REASON_SYN);
}

void STP::close_connection(){
  debug_message("Solicitando cierre de conexion...\n");
  send_much(STP_REASON_FIN);
}

void STP::produce_more(int nbytes){
  debug_message("Produciendo bytes...\n");
  sndbuffer+=nbytes;
  debug_message("Capacidad del buffer de emision: %d\n",sndbuffer);
  if(state==STP_FINISHING || state==STP_ESTABLISHED){
    send_much(STP_REASON_DATA);
  }
}

void STP::consume_more(int nbytes){
  debug_message("Consumiendo bytes...\n");
  rcvbuffer+=nbytes;
  bytes_rcv+=nbytes;
  debug_message("Capacidad actual de buffer: %d\n",rcvbuffer);
  if(rcvbuffer>0){
    recvBytes(rcvbuffer);
    consumed+=rcvbuffer;
    rcvbuffer=0;
  }
  if(consumed>=credit_size_){
    consumed-=credit_size_;
    send_much(STP_REASON_MORE_CREDIT);
  }
}

int STP::is_syn(Packet *pkt){
  debug_message("Es SYN? ");
  int flags;

  hdr_stp *stp_header=(hdr_stp*)pkt->access(off_stp_);
  flags=stp_header->flags();
  (flags&STP_REASON_SYN)?debug_message("SI\n"):debug_message("NO\n");
  return(flags&STP_REASON_SYN);
}

int STP::is_more_credit(Packet *pkt){
  debug_message("Es MORE_CREDIT? ");
  int flags;

  hdr_stp *stp_header=(hdr_stp*)pkt->access(off_stp_);
  flags=stp_header->flags();
  (flags&STP_REASON_MORE_CREDIT)?debug_message("SI\n"):debug_message("NO\n");
  return(flags&STP_REASON_MORE_CREDIT);
}

void STP::get_more_credit(Packet *pkt){
  debug_message("Aumentando credito...\n");
  credit+=credit_size_;
  send_much(STP_REASON_DATA);
}

int STP::is_data(Packet *pkt){
  debug_message("Es DATA? ");
  int flags;

  hdr_stp *stp_header=(hdr_stp*)pkt->access(off_stp_);
  flags=stp_header->flags();
  (flags&STP_REASON_DATA)?debug_message("SI\n"):debug_message("NO\n");
  return(flags&STP_REASON_DATA);
```

```
}

int STP::data(Packet *pkt){
  hdr_cmn *common_header=(hdr_cmn*)pkt->access(off_cmn_);
  return(common_header->size()-stpip_base_hdr_size_);
}

int STP::is_fin(Packet *pkt){
  debug_message("Es FIN? ");
  int flags;

  hdr_stp *stp_header=(hdr_stp*)pkt->access(off_stp_);
  flags=stp_header->flags();
  (flags&STP_REASON_FIN)?debug_message("SI\n"):debug_message("NO\n");
  return(flags&STP_REASON_FIN);
}

int STP::seq(Packet *pkt){
  hdr_stp *stp_header=(hdr_stp*)pkt->access(off_stp_);
  return(stp_header->seqno());
}

void STP::act_seq(Packet *pkt){
  if (seq(pkt)>last_seq_rcv){
    last_seq_rcv=seq(pkt);
  }
}


void STP::send_packet(int bytes, int flags){
  debug_message("Enviando paquete...\n");
  Packet* pkt=allocpkt();
  hdr_stp *stp_header=(hdr_stp*)pkt->access(off_stp_);
  stp_header->flags()=flags;
  last_seq_sent+=bytes;
  stp_header->seqno()=last_seq_sent;
  hdr_cmn *common_header = (hdr_cmn*)pkt->access(off_cmn_);
  common_header->size() = bytes + stpip_base_hdr_size_;
  Agent::send(pkt,0);
}

int STP::get_bytes(){
  debug_message("Obteniendo bytes de buffer (capacidad %d) ",sndbuffer);
  debug_message("(credit %d) (seg %d)\n",credit,segment_size_);
  int bytes;

  //bytes=min(*buffer,segment_size,credit);
  bytes=sndbuffer;
  if(credit<bytes){
    bytes=credit;
  }
  if(segment_size_<bytes){
    bytes=segment_size_;
  }
  sndbuffer-=bytes;
  debug_message("Obtenidos %d bytes",bytes);
  return bytes;
}

void STP::send_much(int reason){
  debug_message("Tratando de enviar paquetes (buffer:%d, credit:%d)...\n",sndbuffer,credit);
  int bytes;

  if(reason==STP_REASON_SYN){
    debug_message("Enviando paquete con SYN...\n");
    send_packet(0,STP_REASON_SYN);
    return;
  }
  if(reason==STP_REASON_MORE_CREDIT){
    debug_message("Enviando paquete con MORE_CREDIT...\n");
    send_packet(0,STP_REASON_MORE_CREDIT);
    return;
  }
  if(reason==STP_REASON_FIN && credit>segment_size_ && sndbuffer>0){
    debug_message("Enviando paquete con FIN+DATA...\n");
    bytes=get_bytes();
    credit-=bytes;
    send_packet(bytes,STP_REASON_FIN+STP_REASON_DATA);
    return;
  } else if(reason==STP_REASON_FIN){
    debug_message("Enviando paquete con FIN...\n");
    send_packet(0,STP_REASON_FIN);
    return;
  }
  while(credit>0 && (sndbuffer>0 || (sndbuffer==0 && state==STP_FINISHING))){
    bytes=get_bytes();
    credit-=bytes;
    if(sndbuffer==0 && state==STP_FINISHING){
      debug_message("Enviando paquete con FIN+DATA...\n");
      send_packet(bytes,STP_REASON_FIN+STP_REASON_DATA);
      change_state(STP_FIN_WAIT);
      return;
    }
    send_packet(bytes,STP_REASON_DATA);
    debug_message("Enviando paquete con DATA (%d bytes)...\n", bytes);
  }
}

int STP::command(int argc, const char*const* argv)
```

```
{
// Copy FullTcp's tcl interface

if (argc == 2) {
if (strcmp(argv[1], "listen") == 0) {
listen();
return (TCL_OK);
}
if (strcmp(argv[1], "close") == 0) {
close();
return (TCL_OK);
}
if (strcmp(argv[1], "resume") == 0) {
resume();
return (TCL_OK);
}
if (strcmp(argv[1], "done") == 0) {
return (TCL_OK);
}
}
if (argc == 3) {
if (strcmp(argv[1], "advance") == 0) {
  //error("advance no definido");
  //advanceby(atoi(argv[2]));
return (TCL_OK);
}
if (strcmp(argv[1], "advanceby") == 0) {
  //error("advanceby no definido");
  //advanceby(atoi(argv[2]));
return (TCL_OK);
}
if (strcmp(argv[1], "advance-bytes") == 0) {
  //error("advancebytes no definido");
  //advance_bytes(atoi(argv[2]));
return (TCL_OK);
}
  }
  return (Agent::command(argc, argv));
}

void STP::debug_message(char *message){
  if(debug_){
    if(this->name())
       printf("%s:",this->name());
    printf(message);
  }
}

void STP::debug_message(char *message,int p1){
  if(debug_){
    if(this->name())
       printf("%s:",this->name());
    printf(message,p1);
  }
}

void STP::debug_message(char *message,int p1,int p2){
  if(debug_){
    if(this->name())
       printf("%s:",this->name());
    printf(message,p1,p2);
  }
}
```

9.4.3 ns/monitor.h y ns/monitor.cc

Estos dos ficheros implementa la clase Monitor.

```
#include "app.h"

class Monitor;

class Monitor : public Application {
 public:
Monitor();
void recv(int nbytes);
double bytes_recibidos_;
double bytes_enviados_;
void resume();
 protected:
int command(int argc, const char*const* argv);
};

#include "monitor.h"
#include "tcp.h"

Monitor::Monitor(){
  bind("bytes_recibidos_", &bytes_recibidos_);
  bind("bytes_enviados_", &bytes_enviados_);
}

void Monitor::recv(int nbytes){
  bytes_recibidos_+=nbytes;
}
```

```
static class MonitorClass : public TclClass {
public:
  MonitorClass():TclClass("Application/Monitor"){}
  TclObject* create(int argc, const char*const* argv){
    return(new Monitor);
      }
} class_monitor;

void Monitor::resume(){
  Tcl::instance().evalf("%s resume", this->name());
}

int Monitor::command(int argc, const char*const* argv){
  return (Application::command(argc,argv));
}
```

9.4.4 definiciones.tcl

Este fichero contiene definiciones comunes a todas las simulaciones realizadas para la elaboración del capítulo 8.

```
#Cargar el modulo WWW
source httpModel.tcl

#Crear un objeto simulador
set ns [new Simulator]

##### FICHEROS #####

#El nombre base del fichero debe ser especificado
set base Ninguna
proc usar_base_fichero {base_fichero} {
    global ns base fichero_texto fichero_nam
    set base $base_fichero
    #Se aprovecha para crear ficheros de traza
    #e indicar al simulador que realice trazas sobre ellos
    #set fichero_texto [open $base.tr w]
    #set fichero_nam [open $base.nam w]
    #$ns trace-all $fichero_texto
    #$ns namtrace-all $fichero_nam
}

#Un fichero con valor 0 es un fichero no abierto
#(Se utiliza para el procedimiento cerrar)
set fichero_texto 0
set fichero_nam 0
set fichero_retardo 0
set fichero_retardo 0
set fichero_throughput_periodo 0
set fichero_throughput_acumulado 0
set fichero_goodput_periodo 0
set fichero_goodput_acumulado 0
set fichero_throughput_alambrico_periodo 0
set fichero_throughput_inalambrico_periodo 0
set fichero_throughput_alambrico_acumulado 0
set fichero_throughput_inalambrico_acumulado 0
set fichero_goodput_alambrico_periodo 0
set fichero_goodput_inalambrico_periodo 0
set fichero_goodput_alambrico_acumulado 0
set fichero_goodput_inalambrico_acumulado 0

##### TOPOLOGIA ####
#puts "Definiendo la Topologia..."

#Nodos: cliente, servidor y ras (Remote Access Server)
set cliente [$ns node]
set ras [$ns node]
set servidor [$ns node]

#Enlaces
set bandwidth_alambrico 100000; #100Kbs
set bandwidth_inalambrico 9600 ;#9.6Kbs
set retardo_alambrico 0.100 ;#100ms
set retardo_inalambrico 0.400 ;#400ms
set am [expr 4*$retardo_inalambrico] ;#Estos cuatro parametros son
set aM [expr 6*$retardo_inalambrico] ;#para la variabilidad
set t_L 50.0
set t_E 10.0
set tipo_cola DropTail
$ns duplex-link $cliente $ras $bandwidth_inalambrico $retardo_inalambrico $tipo_cola
$ns duplex-link $ras $servidor  $bandwidth_alambrico $retardo_alambrico $tipo_cola
set enlace_inalambrico1 [$ns link $cliente $ras] ;#Esto es para hacer
set enlace_inalambrico2 [$ns link $ras $cliente] ;#Esto es para hacer
set retardo_inalambrico1 [$enlace_inalambrico1 link] ;#variable el enlace
set retardo_inalambrico2 [$enlace_inalambrico2 link] ;#variable el enlace
#set retardo_constante 0
if {![info exists retardo_constante]} {
    $retardo_inalambrico1 inicializar $t_L $t_E $am $aM ;#en un sentido
    $retardo_inalambrico2 inicializar $t_L $t_E $am $aM ;#en el otro sentido
}



##### COMUNICACIONES #####
#puts "Definiendo las Comunicaciones..."
```

```
set transporte_cliente Agent/TCP/FullTcp
set transporte_servidor Agent/TCP/FullTcp
set duracion_comunicacion 100000.0
set peticiones_www 2
set peticiones_telnet 50
set tamano_fichero [expr 1*1024]
set comienzo_comunicacion 0.5
set fin_comunicacion [expr $comienzo_comunicacion + $duracion_comunicacion]
set comienzo_registro $comienzo_comunicacion
set comenzar_comunicacion "Accion nula"
set periodo 0.5
set periodo_retardo 0.5
set aplicacion_cliente 0
set aplicacion_ras 0
set aplicacion_servidor 0

proc calcular_periodo_necesario {comunicacion} {
    global periodo bandwidth_alambrico bandwidth_inalambrico retardo_inalambrico retardo_alambrico tamano_fichero peticiones_telnet peticiones_www

    if {$comunicacion=="ftp"} {
set min $bandwidth_inalambrico
if {$bandwidth_inalambrico>$bandwidth_alambrico} {
    set min $bandwidth_alambrico
}
set duracion_optima [expr ($tamano_fichero/$min) + $retardo_inalambrico + $retardo_alambrico]
set duracion_estimada [expr 1.5*$duracion_optima]
    }
    if {$comunicacion=="www"} {
set duracion_estimada [expr $peticiones_www*60]
    }
    if {$comunicacion=="telnet"} {
set duracion_estimada [expr $peticiones_telnet*2]
    }
    set periodo [expr $duracion_estimada/100.0]
}

proc inicializar_parametros_transporte {} {
    set credito_inicial 8192
    set trama_ppp 1500
    set segmento_tcp [expr $trama_ppp - [Agent/TCP set tcpip_base_hdr_size_]]
    set segmento_stp [expr $trama_ppp - [Agent/STP set stpip_base_hdr_size_]]
    Agent/TCP/FullTcp set segsize_ $segmento_tcp
    Agent/TCP/FullTcp set windowInit_ [expr $credito_inicial/[Agent/TCP/FullTcp set segsize_]]
    Agent/STP set segment_size_ $segmento_stp
    Agent/STP set credit_size_ [expr 2*[Agent/STP set credit_size_]]
    Agent/STP set initial_credits_ [expr $credito_inicial/[Agent/STP set credit_size_]]
}

inicializar_parametros_transporte

proc instalar_stp {} {
    puts "Instalando STP..."
    global transporte_cliente transporte_servidor

    set transporte_cliente Agent/STP
    WWWClient set TRANSPORTCL_ Agent/STP
}

proc instalar_proxy {} {
    puts "Instalando Proxy..."
    #Variables globales
    global ns ras transporte_cliente transporte_servidor aplicacion_ras

    $ns install-proxy $ras $transporte_cliente $transporte_servidor
    set aplicacion_ras [$ns set Proxy_]
}

proc instalar_ftp {} {
    #Variables globales
    global ns cliente servidor transporte_cliente transporte_servidor aplicacion aplicacion_cliente aplicacion_servidor comenzar_comunicacion tamano_fichero

    puts "Instalando FTP ($tamano_fichero bytes)..."
    calcular_periodo_necesario ftp
    set tcp_cliente [new $transporte_cliente]
    set tcp_servidor [new $transporte_servidor]
    set aplicacion [new Application/FTP]
    set monitor [new Application/Monitor]
    $ns attach-agent $cliente $tcp_cliente
    $ns attach-agent $servidor $tcp_servidor
    $aplicacion attach-agent $tcp_servidor
    $monitor attach-agent $tcp_cliente
    #Problemitas con mundos y listen
    $ns connect $tcp_servidor $tcp_cliente 1
    $tcp_cliente listen
    set aplicacion_cliente $monitor
    set aplicacion_servidor $aplicacion
    set comenzar_comunicacion "$aplicacion_servidor send $tamano_fichero"
}

proc instalar_telnet {} {
    #Variables globales
    global ns cliente servidor transporte_cliente transporte_servidor aplicacion aplicacion_cliente aplicacion_servidor comenzar_comunicacion peticiones_telnet

    puts "Instalando Telnet ($peticiones_telnet peticiones)..."
    calcular_periodo_necesario telnet
    set tcp_cliente [new $transporte_cliente]
    set tcp_servidor [new $transporte_servidor]
    set telnet_cliente [new Application/Telnet]
    $telnet_cliente set peticiones_ $peticiones_telnet
```

```
    set telnet_servidor [new Application/TelnetServer]
    $ns attach-agent $cliente $tcp_cliente
    $ns attach-agent $servidor $tcp_servidor
    $telnet_cliente attach-agent $tcp_cliente
    $telnet_servidor attach-agent $tcp_servidor
    $ns connect $tcp_cliente $tcp_servidor
    $tcp_servidor listen
    set aplicacion_cliente $telnet_cliente
    set aplicacion_servidor $telnet_servidor
    set comenzar_comunicacion "$aplicacion_cliente start"
}

proc instalar_www {} {
    global ns cliente servidor aplicacion_cliente aplicacion_servidor comenzar_comunicacion WWWverbose_ peticiones_www

    puts "Instalando WWW ($peticiones_www peticiones)..."
    set WWWverbose_ 0

    # Create the www objects and cdf's.
    initCdfs $ns

    calcular_periodo_necesario www
    set aplicacion_servidor [new WWWServer $servidor $ns]
    set aplicacion_cliente [new WWWClient $cliente $ns]
    $aplicacion_cliente set max_requests_ $peticiones_www

    #$wwwClient(0) statDump 20

    set comenzar_comunicacion "$aplicacion_cliente start $aplicacion_servidor"
}

##### RESULTADOS #####
#puts "Definiendo la Recogida de Resultados..."

#Variables con semantica estatica
set registrado_retardo 0
#Registrar el retardo en el enlace inalambrico
proc registrar_retardo {} {
    global ns retardo_inalambrico1 retardo_inalambrico2 fichero_retardo1 fichero_retardo2 periodo_retardo base registrado_retardo

    if {$registrado_retardo==0} {
set registrado_retardo 1
set fichero_retardo1 [open $base.retardo1 w]
set fichero_retardo2 [open $base.retardo2 w]
    }
    set ahora [$ns now]
    puts $fichero_retardo1 "$ahora [$retardo_inalambrico1 retardo]"
    puts $fichero_retardo2 "$ahora [$retardo_inalambrico2 retardo]"
    #$ns at  [expr $ahora+$periodo_retardo] "registrar_retardo"

}


#Variables globales con semantica estatica
set bytes_acumulados_throughput 0.0
set bytes_acumulados_throughput_alambrico 0.0
set bytes_acumulados_throughput_inalambrico 0.0
set tiempo_throughput 0.0
set tiempo_throughput_alambrico 0.0
set tiempo_throughput_inalambrico 0.0
set registrado_throughput 0
set registrado_throughput_alambrico 0
set registrado_throughput_inalambrico 0
set throughput 0.0

#Registrar el throughput global
proc registrar_throughput {} {
    #Variables globales
    global ns periodo aplicacion_cliente aplicacion_servidor fichero_throughput_periodo fichero_throughput_acumulado tiempo_throughput bytes_acumulados_throughput base

    #Obtener el tiempo actual
    set ahora [$ns now]
    if {$registrado_throughput==0} {
set fichero_throughput_periodo [open $base.throughput_periodo w]
set fichero_throughput_acumulado [open $base.throughput_acumulado w]
set registrado_throughput 1
puts $fichero_throughput_periodo "$ahora 0.0"
puts $fichero_throughput_acumulado "$ahora 0.0"
#$ns at [expr $ahora+$periodo] "registrar_throughput"
return
    }
    set agente1 $aplicacion_cliente
    set agente2 $aplicacion_servidor
    set fichero_periodo $fichero_throughput_periodo
    set fichero_acumulado $fichero_throughput_acumulado

    #Calcular el tiempo transcurrido hasta ahora
    set aux $tiempo_throughput
    set tiempo_throughput [expr $ahora - $comienzo_registro]
    set mi_periodo [expr $tiempo_throughput - $aux]
    #Calcular los bytes acumulados y los recibidos en este periodo
    set bytes_periodo [expr ([$agente1 set bytes_recibidos_]+[$agente2 set bytes_recibidos_]) - $bytes_acumulados_throughput]
    set bytes_acumulados_throughput [expr [$agente1 set bytes_recibidos_]+[$agente2 set bytes_recibidos_]]

    #Calcular throughput  en Kbits/Segundo
    set throughput_periodo [expr (($bytes_periodo*8)/$mi_periodo)/1000]
    set throughput_acumulado [expr (($bytes_acumulados_throughput*8)/$tiempo_throughput)/1000]
    set throughput $throughput_acumulado
    #Escribir los resultados en los ficheros
```

```
    puts $fichero_periodo "$ahora $throughput_periodo"
    puts $fichero_acumulado "$ahora $throughput_acumulado"
    #puts "A1([$agente1 set bytes_recibidos_]) A2([$agente2 set bytes_recibidos_]) PERIODO ($bytes_periodo) TP($throughput_periodo)"
    #Replanificar la ejecución del procedimiento
    $ns at [expr $ahora+$periodo] "registrar_throughput"
}

proc registrar_throughput_alambrico {} {
    #Variables globales
    global ns periodo aplicacion_ras aplicacion_servidor fichero_throughput_alambrico_periodo fichero_throughput_alambrico_acumulado tiempo_throughput_alambrico bytes_acumulados_throughput

    #Obtener el tiempo actual
    set ahora [$ns now]
    if {$registrado_throughput_alambrico==0} {
set fichero_throughput_alambrico_periodo [open $base.throughput_alambrico_periodo w]
set fichero_throughput_alambrico_acumulado [open $base.throughput_alambrico_acumulado w]
set registrado_throughput_alambrico 1
puts $fichero_throughput_alambrico_periodo "$ahora 0.0"
puts $fichero_throughput_alambrico_acumulado "$ahora 0.0"
#$ns at [expr $ahora+$periodo] "registrar_throughput_alambrico"
return
    }

    set agente1 $aplicacion_ras
    set agente2 $aplicacion_servidor
    set fichero_periodo $fichero_throughput_alambrico_periodo
    set fichero_acumulado $fichero_throughput_alambrico_acumulado

    #Calcular los bytes acumulados y los recibidos en este periodo
    set bytes_periodo [expr ([$agente1 set bytes_recibidos_mundo2_]+[$agente2 set bytes_recibidos_]) - $bytes_acumulados_throughput_alambrico]
    set bytes_acumulados_throughput_alambrico [expr [$agente1 set bytes_recibidos_mundo2_]+[$agente2 set bytes_recibidos_]]

    #Calcular el tiempo transcurrido hasta ahora
    set aux $tiempo_throughput_alambrico
    set tiempo_throughput_alambrico [expr $ahora - $tiempo_throughput_alambrico]
    set mi_periodo [expr $tiempo_throughput_alambrico - $aux]
    #Calcular throughput en Kbits/Segundo
    set throughput_periodo [expr (($bytes_periodo*8)/$mi_periodo)/1000]
    set throughput_acumulado [expr (($bytes_acumulados_throughput_alambrico*8)/$tiempo_throughput_alambrico)/1000]
    #Escribir los resultados en los ficheros
    puts $fichero_periodo "$ahora $throughput_periodo"
    puts $fichero_acumulado "$ahora $throughput_acumulado"
    #Replanificar la ejecución del procedimiento
    $ns at [expr $ahora+$periodo] "registrar_throughput_alambrico"
}

proc registrar_throughput_inalambrico {} {
    #Variables globales
    global ns periodo aplicacion_ras aplicacion_cliente fichero_throughput_inalambrico_periodo fichero_throughput_inalambrico_acumulado tiempo_throughput_inalambrico bytes_acumulados_thro

    #Obtener el tiempo actual
    set ahora [$ns now]
    if {$registrado_throughput_inalambrico==0} {
set fichero_throughput_inalambrico_periodo [open $base.throughput_inalambrico_periodo w]
set fichero_throughput_inalambrico_acumulado [open $base.throughput_inalambrico_acumulado w]
set registrado_throughput_inalambrico 1
puts $fichero_throughput_inalambrico_periodo "$ahora 0.0"
puts $fichero_throughput_inalambrico_acumulado "$ahora 0.0"
#$ns at [expr $ahora+$periodo] "registrar_throughput_inalambrico"
return
    }

    set agente1 $aplicacion_ras
    set agente2 $aplicacion_cliente
    set fichero_periodo $fichero_throughput_inalambrico_periodo
    set fichero_acumulado $fichero_throughput_inalambrico_acumulado

    #Calcular los bytes acumulados y los recibidos en este periodo
    set bytes_periodo [expr ([$agente1 set bytes_recibidos_mundo1_]+[$agente2 set bytes_recibidos_]) - $bytes_acumulados_throughput_inalambrico]
    set bytes_acumulados_throughput_inalambrico [expr [$agente1 set bytes_recibidos_mundo1_]+[$agente2 set bytes_recibidos_]]

    #Calcular el tiempo transcurrido hasta ahora
    set aux $tiempo_throughput_inalambrico
    set tiempo_throughput_inalambrico [expr $ahora-$comienzo_registro]
    set mi_periodo [expr $tiempo_throughput_inalambrico - $aux]
    #Calcular throughput  en Kbits/Segundo
    set throughput_periodo [expr (($bytes_periodo*8)/$mi_periodo)/1000]
    set throughput_acumulado [expr (($bytes_acumulados_throughput_inalambrico*8)/$tiempo_throughput_inalambrico)/1000]
    #Escribir los resultados en los ficheros
    puts $fichero_periodo "$ahora $throughput_periodo"
    puts $fichero_acumulado "$ahora $throughput_acumulado"
    #Replanificar la ejecución del procedimiento
    $ns at [expr $ahora+$periodo] "registrar_throughput_inalambrico"
}

#Variables globales con semantica estatica
set bytes_usuario_acumulados_goodput 0.0
set bytes_enlace_acumulados_goodput 0.0
set bytes_usuario_acumulados_goodput_alambrico 0.0
set bytes_enlace_acumulados_goodput_alambrico 0.0
set bytes_usuario_acumulados_goodput_inalambrico 0.0
set bytes_enlace_acumulados_goodput_inalambrico 0.0
set registrado_goodput 0.0
set registrado_goodput_alambrico 0.0
set registrado_goodput_inalambrico 0.0
set enlace_alambrico1 [$ns link $servidor $ras]
set enlace_alambrico2 [$ns link $ras $servidor]
set enlace_inalambrico1 [$ns link $cliente $ras]
set enlace_inalambrico2 [$ns link $ras $cliente]
```

```
set cola_alambrica1 [$enlace_alambrico1 queue]
set cola_alambrica2 [$enlace_alambrico2 queue]
set cola_inalambrica1 [$enlace_inalambrico1 queue]
set cola_inalambrica2 [$enlace_inalambrico2 queue]
set monitor_alambrico1 [$ns monitor-queue $servidor $ras $cola_alambrica1]
set monitor_alambrico2 [$ns monitor-queue $ras $servidor $cola_alambrica2]
set monitor_inalambrico1 [$ns monitor-queue $cliente $ras $cola_inalambrica1]
set monitor_inalambrico2 [$ns monitor-queue $ras $cliente $cola_inalambrica2]
set goodput 0.0
set goodput_alambrico 0.0
set goodput_inalambrico 0.0


#Registrar el goodput global
proc registrar_goodput {} {
    #Variables globales
    global registrado_goodput base fichero_goodput_periodo fichero_goodput_acumulado servidor cliente ras ns aplicacion_cliente aplicacion_servidor ns periodo bytes_usu

    #Obtener el tiempo actual
    set ahora [$ns now]
    if {$registrado_goodput==0} {
set fichero_goodput_periodo [open $base.goodput_periodo w]
set fichero_goodput_acumulado [open $base.goodput_acumulado w]
set registrado_goodput 1
puts $fichero_goodput_periodo "$ahora 0.0"
puts $fichero_goodput_acumulado "$ahora 0.0"
#$ns at [expr $ahora+$periodo] "registrar_goodput"
return
    }
    set fichero_periodo $fichero_goodput_periodo
    set fichero_acumulado $fichero_goodput_acumulado
    set agente1 $aplicacion_cliente
    set agente2 $aplicacion_servidor
    set enlace1 $monitor_alambrico1
    set enlace2 $monitor_inalambrico1

    #Calcular los bytes acumulados y los recibidos en este periodo
    set bytes_usuario_periodo [expr ([$agente1 set bytes_recibidos_]+[$agente2 set bytes_recibidos_]) - $bytes_usuario_acumulados_goodput]
    set bytes_enlace_periodo [expr ([$enlace1 set bdepartures_]+[$enlace2 set bdepartures_]) - $bytes_enlace_acumulados_goodput]
    set bytes_usuario_acumulados_goodput [expr [$agente1 set bytes_recibidos_]+[$agente2 set bytes_recibidos_]]
    set bytes_enlace_acumulados_goodput [expr [$enlace1 set bdepartures_]+[$enlace2 set bdepartures_]]

    #Calcular goodput
    set goodput_periodo [calcular_goodput $bytes_usuario_periodo $bytes_enlace_periodo]
    set goodput_acumulado [calcular_goodput $bytes_usuario_acumulados_goodput $bytes_enlace_acumulados_goodput]
    set goodput $goodput_acumulado
    #Escribir los resultados en los ficheros
    puts $fichero_periodo "$ahora $goodput_periodo"
    puts $fichero_acumulado "$ahora $goodput_acumulado"
    #Replanificar la ejecución del procedimiento
    $ns at [expr $ahora+$periodo] "registrar_goodput"
}

proc registrar_goodput_alambrico {} {
    #Variables globales
    global registrado_goodput_alambrico base fichero_goodput_alambrico_periodo fichero_goodput_alambrico_acumulado servidor cliente ras ns aplicacion_ras aplicacion_ser

    #Obtener el tiempo actual
    set ahora [$ns now]
    if {$registrado_goodput_alambrico==0} {
set fichero_goodput_alambrico_periodo [open $base.goodput_alambrico_periodo w]
set fichero_goodput_alambrico_acumulado [open $base.goodput_alambrico_acumulado w]
set registrado_goodput_alambrico 1
puts $fichero_goodput_alambrico_periodo "$ahora 0.0"
puts $fichero_goodput_alambrico_acumulado "$ahora 0.0"
#$ns at [expr $ahora+$periodo] "registrar_goodput_alambrico"
return
    }
    set fichero_periodo $fichero_goodput_alambrico_periodo
    set fichero_acumulado $fichero_goodput_alambrico_acumulado
    set agente1 $aplicacion_ras
    set agente2 $aplicacion_servidor
    set enlace1 $monitor_alambrico1
    set enlace2 $monitor_alambrico2

    #Calcular los bytes acumulados y los recibidos en este periodo
    set bytes_usuario_periodo [expr ([$agente1 set bytes_recibidos_mundo2_]+[$agente2 set bytes_recibidos_]) - $bytes_usuario_acumulados_goodput_alambrico]
    set bytes_enlace_periodo [expr ([$enlace1 set bdepartures_]+[$enlace2 set bdepartures_]) - $bytes_enlace_acumulados_goodput_alambrico]
    set bytes_usuario_acumulados_goodput_alambrico [expr [$agente1 set bytes_recibidos_mundo2_]+[$agente2 set bytes_recibidos_]]
    set bytes_enlace_acumulados_goodput_alambrico [expr [$enlace1 set bdepartures_]+[$enlace2 set bdepartures_]]

    #Calcular goodput
    set goodput_periodo [calcular_goodput $bytes_usuario_periodo $bytes_enlace_periodo]
    set goodput_acumulado [calcular_goodput $bytes_usuario_acumulados_goodput_alambrico $bytes_enlace_acumulados_goodput_alambrico]
    set goodput_alambrico $goodput_acumulado
    #Escribir los resultados en los ficheros
    puts $fichero_periodo "$ahora $goodput_periodo"
    puts $fichero_acumulado "$ahora $goodput_acumulado"
    #Replanificar la ejecución del procedimiento
    $ns at [expr $ahora+$periodo] "registrar_goodput_alambrico"
}

proc registrar_goodput_inalambrico {} {
    #Variables globales
    global registrado_goodput_inalambrico base fichero_goodput_inalambrico_periodo fichero_goodput_inalambrico_acumulado servidor cliente ras ns aplicacion_ras aplicaci

    #Obtener el tiempo actual
    set ahora [$ns now]
    if {$registrado_goodput_inalambrico==0} {
```

```
set fichero_goodput_inalambrico_periodo [open $base.goodput_inalambrico_periodo w]
set fichero_goodput_inalambrico_acumulado [open $base.goodput_inalambrico_acumulado w]
set registrado_goodput_inalambrico 1
puts $fichero_goodput_inalambrico_periodo "$ahora 0.0"
puts $fichero_goodput_inalambrico_acumulado "$ahora 0.0"
#$ns at [expr $ahora+$periodo] "registrar_goodput_inalambrico"
return
    }
    set fichero_periodo $fichero_goodput_inalambrico_periodo
    set fichero_acumulado $fichero_goodput_inalambrico_acumulado
    set agente1 $aplicacion_ras
    set agente2 $aplicacion_cliente
    set enlace1 $monitor_inalambrico1
    set enlace2 $monitor_inalambrico2

    #Calcular los bytes acumulados y los recibidos en este periodo
    set bytes_usuario_periodo [expr ([$agente1 set bytes_recibidos_mundo1_]+[$agente2 set bytes_recibidos_]) - $bytes_usuario_acumulados_goodput_inalambrico]
    set bytes_enlace_periodo [expr ([$enlace1 set bdepartures_]+[$enlace2 set bdepartures_]) - $bytes_enlace_acumulados_goodput_inalambrico]
    set bytes_usuario_acumulados_goodput_inalambrico [expr [$agente1 set bytes_recibidos_mundo1_]+[$agente2 set bytes_recibidos_]]
    set bytes_enlace_acumulados_goodput_inalambrico [expr [$enlace1 set bdepartures_]+[$enlace2 set bdepartures_]]

    #Calcular goodput
    set goodput_periodo [calcular_goodput $bytes_usuario_periodo $bytes_enlace_periodo]
    set goodput_acumulado [calcular_goodput $bytes_usuario_acumulados_goodput_inalambrico $bytes_enlace_acumulados_goodput_inalambrico]
    set goodput_inalambrico $goodput_acumulado
    #Escribir los resultados en los ficheros
    puts $fichero_periodo "$ahora $goodput_periodo"
    puts $fichero_acumulado "$ahora $goodput_acumulado"
    #Replanificar la ejecución del procedimiento
    $ns at [expr $ahora+$periodo] "registrar_goodput_inalambrico"
}

proc calcular_goodput {user_rate transfer_rate} {
    if {$user_rate==0 && $transfer_rate==0} {
return 0.0
    }
    if {$user_rate==0} {
return 0.0
    }
    if {$transfer_rate==0} {
return 1.0
    }
    #Lo anterior por convenio
    return [expr (1.0*$user_rate)/$transfer_rate]
}

proc ultimo_registro {} {
    global registrado_throughput
    global registrado_throughput_alambrico
    global registrado_throughput_inalambrico
    global registrado_goodput
    global registrado_goodput_alambrico
    global registrado_goodput_inalambrico

    if {$registrado_throughput} {
registrar_throughput
    }
    if {$registrado_throughput_alambrico} {
registrar_throughput_alambrico
    }
    if {$registrado_throughput_inalambrico} {
registrar_throughput_inalambrico
    }
    if {$registrado_goodput} {
registrar_goodput
    }
    if {$registrado_goodput_alambrico} {
registrar_goodput_alambrico
    }
    if {$registrado_goodput_inalambrico} {
registrar_goodput_inalambrico
    }
}

#OTROS
proc cerrar {fichero} {
    if {$fichero!=0} {
close $fichero
    }
}


proc finalizar {} {
    global ns base
    global fichero_texto
    global fichero_nam
    global fichero_throughput_periodo 0
    global fichero_throughput_acumulado
    global fichero_goodput_periodo
    global fichero_goodput_acumulado
    global fichero_throughput_alambrico_periodo
    global fichero_throughput_inalambrico_periodo
    global fichero_throughput_alambrico_acumulado
    global fichero_throughput_inalambrico_acumulado
    global fichero_goodput_alambrico_periodo
    global fichero_goodput_inalambrico_periodo
    global fichero_goodput_alambrico_acumulado
    global fichero_goodput_inalambrico_acumulado
    global registrado_throughput
```

```
    global registrado_throughput_alambrico
    global registrado_goodput
    global registrado_goodput_alambrico

    informe
    puts "finalizando..."
    #$ns flush-trace
    cerrar $fichero_texto
    cerrar $fichero_nam
    cerrar $fichero_throughput_periodo
    cerrar $fichero_throughput_acumulado
    cerrar $fichero_goodput_periodo
    cerrar $fichero_goodput_acumulado
    cerrar $fichero_throughput_alambrico_periodo
    cerrar $fichero_throughput_inalambrico_periodo
    cerrar $fichero_throughput_alambrico_acumulado
    cerrar $fichero_throughput_inalambrico_acumulado
    cerrar $fichero_goodput_alambrico_periodo
    cerrar $fichero_goodput_inalambrico_periodo
    cerrar $fichero_goodput_alambrico_acumulado
    cerrar $fichero_goodput_inalambrico_acumulado
    #exec nam $base.nam &
    #exec xgraph $base.throughput_acumulado $base.throughput_periodo &
    #if {($registrado_goodput && !$registrado_goodput_alambrico)} {
#exec xgraph $base.goodput_acumulado &
    #} elseif {$registrado_goodput_alambrico} {
#exec xgraph $base.goodput_alambrico_acumulado $base.goodput_inalambrico_acumulado &
# }
puts "============================================================="
    exit 0
}

proc informe {} {
    global aplicacion_cliente aplicacion_servidor ns comienzo_comunicacion throughput goodput goodput_alambrico goodput_inalambrico fin_comunicacion

    set tcl_precision 3
    set duracion [expr $fin_comunicacion - $comienzo_comunicacion]
    set throughput_cliente [expr (([$aplicacion_cliente set bytes_recibidos_]/1000.0)*8)/$duracion]
    set throughput_servidor [expr (([$aplicacion_servidor set bytes_recibidos_]/1000.0)*8)/$duracion]
    puts "Duracion de la comunicacion $duracion segundos"
    puts "Cliente: [expr [$aplicacion_cliente set bytes_enviados_]/1024.0]KB enviados, [expr [$aplicacion_cliente set bytes_recibidos_]/1024.0]KB recibidos, $throughput
    puts "Servidor: ([expr [$aplicacion_servidor set bytes_enviados_]/1024.0]KB enviados, [expr [$aplicacion_servidor set bytes_recibidos_]/1024.0]KB recibidos $through
    puts "Throughput total: $throughput Kbs"
    puts "Goodputs: $goodput (total), $goodput_alambrico (alambrico), $goodput_inalambrico (inalambrico)"
}


Application/Monitor instproc resume {} {
    global fin_comunicacion ns

    puts "Aplicacion terminada por fin de transmision."
    set fin_comunicacion [$ns now]
    ultimo_registro
    finalizar
}

WWWClient instproc resume {} {
    global fin_comunicacion ns

    puts "Aplicacion terminada por fin de transmision."
    set fin_comunicacion [$ns now]
    ultimo_registro
    finalizar
}

Application/Telnet instproc resume {} {
    global fin_comunicacion ns

    puts "Aplicacion terminada por fin de transmision."
    set fin_comunicacion [$ns now]
    ultimo_registro
    finalizar
}
```

9.4.5 ns/pingpong.h y ns/pingpong.cc

Estos dos ficheros contienen la implementación de la aplicación PingPong definida en el capítulo 5.

```
#include "app.h"

class PingPong : public Application {
 public:
  PingPong();
  void send(int nbytes);
  void recv(int nbytes);
 protected:
  int command(int argc, const char*const* argv);
  void start();
  void stop();
  int stop_;
};

#include "pingpong.h"
```

```
#include "tcp.h"

static class PingPongClass:public TclClass{
public:
  PingPongClass():TclClass("Application/PingPong"){}
  TclObject* create(int argc, const char*const* argv){
    return(new PingPong);
  }
} class_pingpong;


PingPong::PingPong(){
  bind("stop_", &stop_);
}

void PingPong::start(){
  stop_=0;
}

void PingPong::recv(int nbytes){
  if(!stop_){
    agent_->send(nbytes);
  }
}

void PingPong::send(int nbytes){
  agent_->send(nbytes);
}

void PingPong::stop(){
  stop_=1;
}

int PingPong::command(int argc, const char*const* argv)
{
  if(argc==2){
    if(strcmp(argv[1],"start")==0){
      start();
      return(TCL_OK);
    }
    if(strcmp(argv[1],"stop")==0){
      stop();
      return(TCL_OK);
    }
  }
  if(argc==3){
    if(strcmp(argv[1],"send")==0){
      send(atoi(argv[2]));
      return(TCL_OK);
    }
  }
  return(Application::command(argc, argv));
}


\subsection{no.tcl}


Este fichero contiene el código de la simulación presentada en el cap\'itulo~\ref{Simulaciones Sencillas} bajo el nombre de UNO.

#Obtener una instancia del simulador
set ns [new Simulator]

#Abrir un fichero para guardar las trazas y otro para NAM
set f [open uno.tr w]
set nf [open uno.nam w]

#Obligar a que se haga una traza de todo
$ns trace-all $f
$ns namtrace-all $nf

#Definir colores
$ns color 1 Blue
$ns color 2 Red

#Crear los nodos
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Definir los enlaces
$ns duplex-link $n2 $n0 128Kbs 100ms SFQ
$ns duplex-link $n1 $n2 128Kbs 25ms DropTail
$ns duplex-link $n2 $n3 512Kbs 25ms DropTail

#Crear los agentes de transporte, sumideros y generadores de trafico
set udp1 [new Agent/UDP]
set udp3 [new Agent/UDP]
set null0 [new Agent/TCPSink]
set null3 [new Agent/Null]
set cbr3 [new Application/Traffic/CBR]
set expo1 [new Traffic/Expoo]

#Asociar los agentes
$ns attach-agent $n3 $udp3
$ns attach-agent $n1 $udp1
$ns attach-agent $n0 $null0
$ns attach-agent $n3 $null3
```

```
$expo1 attach-agent $udp1
$cbr3 attach-agent $udp3

#Definir los agentes
$udp3 set fid_ 1
$udp1 set fid_ 2
$cbr3 set packetsize_ 210b
#$cbr3 set rate_ 5 Mb
#$expo1 set packet_size_ 512b
$expo1 set burst_time_ 0.5
$expo1 set idle_time_ 0.1
$expo1 set rate 92Kb


#Conectar los agentes de transporte
$ns connect $udp1 $null3
$ns connect $udp3 $null0

#Indicar visualizacion de las colas
$ns duplex-link-op $n2 $n0 queuePos 0.5

#Definir los eventos
$ns at 0.5 "$cbr3 start"
$ns at 0.75 "$expo1 start"
$ns at 2.25 "$cbr3 stop"
$ns at 2.5 "$expo1 stop"
$ns at 3.0 "finalizar"

#definir el procedimiento finalizar
proc finalizar {} {
    global ns f nf
    $ns flush-trace
    close $f
    close $nf
    exec nam uno.nam &
    exit 0
}

#Lanzar la simulacion
$ns run
```

9.4.6 dos.tcl

Este fichero contiene el código de la simulación presentada en el capítulo 5 bajo el nombre de DOS.

```
#Obtener una instancia del simulador
set ns [new Simulator]

#Abrir un fichero para guardar las trazas y otro para NAM
set f [open dos.tr w]
set nf [open dos.nam w]

#Obligar a que se haga una traza de todo
$ns trace-all $f
$ns namtrace-all $nf

#Definir colores
$ns color 1 Blue

#Establecer el routing dinamico
$ns rtproto DV

#Crear los nodos en un array
for {set i 0} {$i<5} {incr i} {
    set n($i) [$ns node]
}

#Definir los enlaces
$ns duplex-link $n(0) $n(1) 1Mbs 10ms SFQ
$ns duplex-link $n(1) $n(2) 1Mbs 10ms SFQ
$ns duplex-link $n(1) $n(3) 1Mbs 10ms SFQ
$ns duplex-link $n(2) $n(3) 1Mbs 10ms SFQ
$ns duplex-link $n(3) $n(4) 1Mbs 10ms SFQ

#Crear los agentes
set sumideroTCP [new Agent/TCPSink]
set fuenteTCP [new Agent/TCP]
set ftp [new Application/FTP]

#Asociar los agentes
$ns attach-agent $n(0) $sumideroTCP
$ns attach-agent $n(4) $fuenteTCP
$ftp attach-agent $fuenteTCP

#Conectar los agentes
$ns connect $fuenteTCP $sumideroTCP

#Definir el color del flujo
$fuenteTCP set fid_ 1

#Indicar visualizacion de las colas
$ns duplex-link-op $n(4) $n(3) queuePos 0.5

#Definir los eventos
```

```
$ns at 0.5 "$ftp start"
$ns rtmodel-at 1.0 down $n(1) $n(3)
$ns rtmodel-at 1.5 up $n(1) $n(3)
$ns at 2.0 "$ftp stop"
$ns at 2.5 "finalizar"

#definir el procedimiento finalizar
proc finalizar {} {
    global ns f nf
    $ns flush-trace
    close $f
    close $nf
    exec nam dos.nam &
    exit 0
}

#Lanzar la simulacion
$ns run
```

9.4.7 tres.tcl

Este fichero contiene el código de la simulación presentada en capítulo 5 bajo el nombre de TRES.

```
set ns [new Simulator]
Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1
set grupo [Node allocaddr]

set f [open tres.tr w]
set nf [open tres.nam w]
set g1 [open tres.entrante w]
set g2 [open tres.saliente w]
$ns trace-all $f
$ns namtrace-all $nf

$ns color 1 red

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

# Use automatic layout
$ns duplex-link $n0 $n1 1.5Mb 50ms DropTail
$ns duplex-link $n0 $n2 1.5Mb 25ms DropTail
$ns duplex-link $n0 $n3 2Mb 25ms DropTail
$ns duplex-link $n0 $n4 2Mb 10ms DropTail

set mproto DM
set mrthandle [$ns mrtproto $mproto {}]

set emisor1 [new Agent/UDP]
set aplicacion [new Application/Traffic/CBR]
set sumidero2 [new Agent/Null]
set sumidero3 [new Agent/Null]
set sumidero4 [new Agent/Null]

$ns attach-agent $n1 $emisor1
$aplicacion attach-agent $emisor1
$emisor1 set dst_ $grupo
$ns attach-agent $n2 $sumidero2
$ns attach-agent $n3 $sumidero3
$ns attach-agent $n4 $sumidero4

$emisor1 set fid_ 1

#Definir el grupo multicast
$n2 join-group $sumidero2 $grupo
$n3 join-group $sumidero3 $grupo
$n4 join-group $sumidero4 $grupo

#Definir los eventos
$ns at 0.5 "registrar"
$ns at 0.5 "$aplicacion start"
$ns at 1.0 "$n4 leave-group $sumidero4 $grupo"
$ns at 1.5 "$n3 leave-group $sumidero3 $grupo"
$ns at 2.0 "$aplicacion stop"
$ns at 2.5 "finalizar"

set enlace1 [$ns link $n1 $n0]
set enlace2 [$ns link $n0 $n2]
set enlace3 [$ns link $n0 $n3]
set enlace4 [$ns link $n0 $n4]

set cola1 [$enlace1 queue]
set cola2 [$enlace2 queue]
set cola3 [$enlace3 queue]
set cola4 [$enlace4 queue]

set monitor1 [$ns monitor-queue $n1 $n0 $cola1]
set monitor2 [$ns monitor-queue $n0 $n2 $cola2]
set monitor3 [$ns monitor-queue $n0 $n3 $cola3]
set monitor4 [$ns monitor-queue $n0 $n4 $cola4]
```

```
#definir el procedimiento registrar
proc registrar {} {
    global g1 g2 monitor1 monitor2 monitor3 monitor4
    #Obtener una instancia del simulador
    set ns [Simulator instance]
    #Definir el periodo de registro
    set periodo 0.05
    #Obtener datos sobre los bytes recibidos por cada agente
    set b1 [$monitor1 set bdepartures_]
    set b2 [$monitor2 set bdepartures_]
    set b3 [$monitor3 set bdepartures_]
    set b4 [$monitor4 set bdepartures_]
    $monitor1 set bdepartures_ 0
    $monitor2 set bdepartures_ 0
    $monitor3 set bdepartures_ 0
    $monitor4 set bdepartures_ 0
    set ahora [$ns now]
    set bwsaliente [expr (((($b2+$b3+$b4)*8)/$periodo)/1000000]
    set bwentrante [expr (($b1*8)/$periodo)/1000000]
    puts $g2 "$ahora $bwsaliente"
    puts $g1 "$ahora $bwentrante"

    $ns at [expr $ahora+$periodo] "registrar"
}

#definir el procedimiento finalizar
proc finalizar {} {
    global ns f nf
    $ns flush-trace
    close $f
    close $nf
    exec nam tres.nam &
    exec xgraph tres.entrante tres.saliente &
    exit 0
}

#Lanzar la simulacion
$ns run
```

9.4.8 cuatro.tcl

Este fichero contiene el código de la simulación presentada en el capítuloSimulaciones Sencillas bajo el nombre de CUATRO.

```
#Obtener una instancia del simulador
set ns [new Simulator]

#Abrir un fichero para guardar las trazas y otro para NAM
set f [open cuatro.tr w]
set nf [open cuatro.nam w]

#Obligar a que se haga una traza de todo
$ns trace-all $f
$ns namtrace-all $nf

#Definir colores
$ns color 1 Blue
$ns color 2 Red

#Crear los nodos
set nodoping [$ns node]
set nodopong [$ns node]

#Definir los enlaces
$ns duplex-link $nodoping $nodopong 64Kb 100ms SFQ

#Crear los agentes de transporte, sumideros y generadores de trafico
set tcpping [new Agent/TCP/FullTcp]
set tcppong [new Agent/TCP/FullTcp]
set ping [new Application/PingPong]
set pong [new Application/PingPong]

#Asociar los agentes
$ns attach-agent $nodoping $tcpping
$ns attach-agent $nodopong $tcppong
$ping attach-agent $tcpping
$pong attach-agent $tcppong

#Conectar los agentes de transporte
$ns connect $tcpping $tcppong

$tcppong listen

#Definir colores de flujo
$tcpping set fid_ 1
$tcppong set fid_ 2


#Definir los eventos

$ns at 0.5 "$ping start"
$ns at 0.5 "$pong start"
$ns at 0.6 "$ping send 512"
$ns at 2.5 "$ping stop"
```

```
$ns at 2.5 "$pong stop"
$ns at 3.0 "finalizar"

#definir el procedimiento finalizar
proc finalizar {} {
    global ns f nf
    $ns flush-trace
    close $f
    close $nf
    exec nam cuatro.nam &
    exit 0
}

#Lanzar la simulacion
$ns run
```

9.4.9 Simulaciones

Los ficheros que han servido para realizar las simulaciones cuyos resultados están han sido recogidos en el capítuloSimulaciones siguen un mismo patrón.

El nombre tiene el patrón `sistema-comunicación.tcl`, donde `sistema` representa el escenario que se desea simular y `comunicación`, el tipo de comunicación.

Los escenarios posibles son cte (comunicación directa manteniendo constante el retardo en el enlace inalámbrico), tcp (comunicación directa), tcp-tcp (comunicación indirecta) y stp-tcp (comunicación indirecta con STP sustituyendo a TCP en el enlace inalámbrico).

El tipo de comunicación puede ser ftp, telnet ó www según se desee simular comunicaciones tipo FTP, tipo Telnet o tipo WWW.

A continuación se presenta un patrón del contenido de los ficheros de simulaciones. Los parámetros de las comunicaciones (tamaño de fichero, número de peticiones Telnet o WWW) pueden ser definidos en estos ficheros o preferiblemente en el fichero `definiciones.tcl`.

```
#Retardo cte?
#set retardo_cte 1

#Cargar definiciones
source definiciones.tcl

#Abrir los ficheros necesarios
usar_base_fichero sistema-comunicación

#Instalar STP?
#instalar_stp

#Instalar proxy?
#instalar_proxy

#Se debe elegir una de las tres opciones
#Instalar aplicacion FTP?
#instalar_ftp
#Instalar aplicacion Telnet?
#instalar_telnet
#Instalar aplicacion WWW
#instalar_www


#Definir eventos
puts "Programando eventos..."
$ns at $comienzo_registro "registrar_retardo"
$ns at $comienzo_registro "registrar_throughput"
# Las dos siguientes solo en caso de utilizar
# comunicaciones indirectas
$ns at $comienzo_registro "registrar_throughput_alambrico"
$ns at $comienzo_registro "registrar_throughput_inalambrico"
$ns at $comienzo_registro "registrar_goodput"
# Las dos siguientes solo en caso de utilizar
# comunicaciones indirectas
$ns at $comienzo_registro "registrar_goodput_alambrico"
$ns at $comienzo_registro "registrar_goodput_inalambrico"
$ns at $comienzo_comunicacion $comenzar_comunicacion
$ns at $fin_comunicacion "finalizar"

$ns run
```

9.5 Miscelánea

Se incluyen en este capítulo ficheros que no contienen código fuente.

9.5.1 Makefile

Este fichero ha debido ser modificado para poder compilar el código C++ incorporado.

```
OBJ_CC = \
...
srm-topo.o \
ping.o \
pingpong.o \
proxy.o \
monitor.o \
stp.o \
alloc-address.o address.o \
...
# Se ha anyadido ping.o, pingpong.o, proxy.o, stp.o y monitor.o a la lista de objetos
```

Chapter 10

Mejoras en el Proyecto

10.1 Introducción

El presente capítulo recoge ideas de mejora en diferentes aspectos del proyecto, y que por falta de recursos (principalmente tiempo) no han podido llevarse a cabo.

Las razones por las que diversos aspectos pueden ser mejorados son varias. Primero, hay que tener en cuenta herramienta de simulación utilizada [Network Research Group, 1998] está en continuo desarrollo; se debería seguir esta evolución por si surgen cosas interesantes o que afecten a lo desarrolado en el proyecto. Cuando comenzó el proyecto los conocimientos eran menores; tareas realizadas entonces obtuvieron productos que ahora serían mejor realizados.

10.2 Mejoras

10.2.1 Modelos Analíticos

Una idea que surgió al comienzo del proyecto y que quedó como un objetivo a realizar en caso de disponer de tiempo suficiente, fue intentar avalar el estudio presentado en el capítulo 6 no sólo con resultados obtenidos en simulaciones, sino con modelos análiticos.

A lo largo de la realización del proyecto no se encontró en las fuentes bibliográficas más que un estudio basado en modelos matemáticos. En [Nguyen, 1996] se presenta un estudio que trata de hallar modelos de enlaces inalámbricos lo más cercanos posibles a la realidad. La ausencia de información encontrada en estudios similares al proyecto que se estaba realizando, sumada a la falta de tiempo, deshicieron toda esperanza de aportar una demostración formal basada en modelos analíticos.

10.2.2 Ampliación de los Manuales

Los manuales de Funcionamiento Interno (capítulo 4 y Externo (capítulo 3 no contienen todo lo que se puede contar de la herramienta de simulación de redes ns: UCB/LBNL Network Simulator. De hecho ni siquiera el manual *oficial* [Fall et al., 1998] está completo.

La realización de estos manuales tuvo como principal motivo la familiarización con la herramienta, y no tanto la obtención de manuales que sirvieran de alternativa a [Fall et al., 1998]. No obstante, si la herramienta de simulación de redes va a ser utilizada por usuarios nuevos (quizá alumnos de alguna asignatura relacionada con las redes de ordenadores), estos manuales deberían revisarse ampliando y recortando donde hiciere falta.

10.2.3 Mejor Caracterización de los Enlaces GSM

La falta de datos sobre el comportamiento de los enlaces GSM ha dado lugar a un modelo que quizá no sea muy fiable. Con la obtención de nuevas medidas reales de tales enlaces, el modelo diseñado debería ser revisado. Se sospecha que afortunadamente los únicos cambios que se deberían realizar consistirían en cambiar valores de parámetros, con lo que el impacto en la implementación sería mínimo.

10.2.4 Mejor Implementación de la Variabilidad en los Enlaces

La implementación de la variabilidad en el retardo de un enlace dista mucho de ser elegante. Los escasos conocimientos de OTcl y de los complicados mecanismos internos de ns impidieron en un primer momento hacer una implementación más elegante.

Implementar enlaces con retardo variable como una subclase de la clase SimpleLink fue algo que se intentó y no se pudo hacer. El constructor de esta clase tendría además como argumentos los parámetros del enlace y no sería necesaria la parafernalia que implica actualmente hacer que el retardo de un enlace sea variable.

10.2.5 Mejor Caracterización de Comunicaciones tipo Telnet

La aplicación Telnet diseñada tiene un punto flaco. Los tamaños de los mensajes tienen un tamaño que no ha sido basado en datos reales y además, al contrario que en la realidad, permanecen constantes.

Una posible mejora sería obtener datos más reales sobre los patrones de tráfico de comunicaciones Telnet y modificar el modelo para hacerlo más cercano a la realidad.

El problema principal de hacer que los mensajes de tamaño variables es que no se puede hacer con el pobre interfaz de aplicación implementado. Por ello el generador de tráfico HTTP hace uso de un enrevesado mecanismo de *callbacks*. Una forma de implementar una aplicación Telnet más real sería modificar el generador de tráfico HTTP para que las respuestas del servidor sean únicas y para que los tamaños de mensajes y el intervalo de peticiones se ajusten a la realidad.

10.2.6 Comunicaciones basadas en Intercambio *real* de datos

La mayoría de las aplicaciones implemetadas en ns intercambian únicamente cantidades de bytes, pues tal es el servicio que ofrecen los agentes de trans-

porte. No obstante, una ampliación en la herramienta ha sido desarrollada para permitir que las aplicaciones intercambien datos reales.

Esta ampliación no estaba suficientemente documentada cuando comenzón el proyecto, así que no fue tenida en cuenta. Sin embargo, parece que nuevas versiones de la documentación de ns [Fall et al., 1998] tratan con más profundidad esta funcionalidad de la herramienta.

Aunque en principio lo más importante en el estudio realizado, son los patrones de tráfico, utilizar el intercambio real de datos implicaría usar aplicaciones más versátiles y que no necesitarían de complicados mecanismos de *callback*.

10.2.7 Optimización de la Implementación de STP

La implementación de STP no contempla el envío de datos junto con peticiones de apertura de conexión. Si se quisiera hacer más eficiente aún este protocolo de transporte, podría pensarse en añadir esta funcionalida. En su momento no se hizo por justicia: la implementación de TCP usada no lo hace.

10.2.8 Código más y mejor comentado

En general el código cuenta con comentarios bastante pobres que pueden hacerlo difícil de descifrar a quién no sea el autor. Si hay tiempo para ello, el código introducido y modificado debería ser convenientemente comentado.

10.2.9 Pruebas con aleatoriedad *real*

En las pruebas realizadas hay un cierto factor de aleatoriedad, presente, por ejemplo, en los intervalos entre peticiones Telnet o WWW, y en la variabilidad de los retardos. Dicha aleatoriedad es fija, es decir, siempre sucede lo mismo en cada ejecución pues las semillas de los generadores de aleatoriedad son siempre las mismas.

Esto ha permitido que existiera justicia en las comparaciones realizadas. Sin embargo, sería aún más justo y adecuado inicializar las semillas con valores aleatorios (basados, por ejemplo, en la hora actual) y realizar un conjunto suficiente de pruebas obteniendo luego medias de los resultados obtenidos.

El no haber hecho esto así ha provocado que algunos casos (transmisiones cortas de ficheros) no se vieran afectados por la variabilidad del enlace. Utilizando una aleatoriedad *real* de todo el conjunto de pruebas de una misma sesión, algunas se verían afectadas y otras no, y todo se vería recogido en la media (y quizá otros estadísticos) de los resultados.

Lo dicho no invalida en absoluto el estudio realizado, puesto que se hicieron diferentes pruebas de cada tipo de comunicación, y algunas de la suficiente duración como para que los resultados obtenidos sean enagñosos.

10.2.10 Indagar sobre los Enlaces inalámbricos implementados

Por lo que se ha podido saber leyendo nuevas versiones del manual de ns y mensajes en la lista de usuarios de ns, esta herramienta cuenta con la posibilidad de simular redes locales inalámbricas. Sería interesante indagar sobre esto y observar como son los enlaces inalámbricos implementados para ello.

10.2.11 Mejoras en el Proxy

Las posibles mejoras en el proxy son varias. Por un lado, si finalmente se implementara el intercambio real de datos, podría pensarse en adecuar la implementación del Proxy a ello, de manera que este elemento podría trabajar con datos reales realizando funciones de filtro como las indicadas en capítuloiEntornos Hibridos.

El tema de las conexiones está no muy bien implementado. En principio, se hizo que éstas el uso del proxy fuera transparente a las aplicaciones, pero más tarde se tuvo que optar por introducir cierta falta de transparencia para hacer posible la distinción entre mundos y así poder recoger las estadísticas definidas. Habría que pensar en como mejorar este tema.

Una mejora apetecible en el proxy es la anticipación de aperturas de conexión, que actualmente no es posible debido al interfaz de aplicación existente. Este interfaz no notifica a las aplicaciones sobre peticiones de apertura de conexión. En el mundo real, un proxy de nivel de aplicación también permanecería ignorante a las peticiones de apertura de conexión, pero quizá un proxy de nivel de transporte podría anticiparse comenzar a abrir la petición en un extremo antes de recibir los primeros datos del otro. Todo esto no esta claro; habría que informarse mejor sobre lo que ocurre en el mundo real.

Bibliography

[Alanko et al., 1994] Timo Alanko, Markku Kojo, Heimo Laamanen, Mika Liljeberg, Marko Moilanen, Kimmo Raatikainen (1994). Measured Performance of Data Transmission Over Cellular Telephone Networks. Department of Computer Science, University of Helsinki, Report C-1994-53, November 1994. URL: http://www.cs.helsinki.fi/research/mowgli/mowgli-papers.html

[Bakshi et al., 1995] Bikram S. Bakshi, P. Krishna, N.H. Vaidya, D.K. Pradhan (1995). Improving Performance of TCP over Wireless Networks. 1st ACM Conference on Mobile Computing and Networking, Berkeley, CA, November 1995. URL: http://www.cs.berkeley.edu/ hari/papers/papers.html

[Balakrishnan et al., 1996] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Randy H. Katz (1996). A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. ACM SIGCOMM Conference, Standford CA, USA, Aug 1996. URL: http://daedalus.cs.berkeley.edu

[Balakrishnan et al., 1997] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, Elan Amir, Randy H. Katz (1997). TCP Improvements for Heterogeneous Networks: The Daedalus Approach. 35th Annual Allerton Conference on Communication, Control, and Computing, Urbana, Illinois, Octubre 1997. URL: http://daedalus.cs.berkeley.edu

[Cáceres et al., 1995] Ramón Cáceres, Liviu Iftode (1995). Improving the Performance of Reliable Transport Protocols in Mobile Computing Enviroments. IEEE Journal on Selected Areas in Communications, vol. 13, NO. 5, June 1995. URL: http://www.research.att.com/ ramon/

[Comer, 1998] Douglas Comer (1998). Internetworking with TCP/IP: principles, protocols, and architecture. Prentice Hall, 1988.

[Degemark et al., 1996] M. Degermark, M. Engan, B. Nordgen, and S. Pink (1996). Low-loss TCP/IP Header Compression for Wireless Networks. MOBICOM Š96, Rye New York, November 1996. ACM and IEEE.

[Duncan, 1996] William R. Duncan (1996). A guide to the project management body of knowledgement. Project Management Institute.

[Fall et al., 1996] Kevin Fall and S. Floyd (1996). Comparisons of Tahoe, Reno, and Sack TCP. URL ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z, December 1996.

[Fall et al., 1998] Kevin Fall, Kannan Varadhan (1998). ns Notes and Documentation. The Vint Project. A collaboration between members at UC Berkeley, LBL, USC/ISI, and Xerox PARC. 21 de Octubre de 1998. URL: http://www-mash.CS.Berkeley.EDU/ns/ns-documentation.html

[Keshav et al., 1996] S. Keshav and S. Morgan (1996). Smart retransmission: Performance with overload and random losses. URL: http://www.cs.att.com/netlib/att/cs/home/keshav/papers/smart.ps.Z, 1996. Preprint.

[Greis, 1998] Marc Greis (1998). Tutorial for the Network Simulator. URL: http://titan.cs.uni-bonn.de/ greis/ns/ns.html

[Henderson et al., 1998a] Tom Henderson, Emile Sahouria (1998a). UCB/LBNL Network Simulator: Contributed Code, HTTP/1.0 Traffic Generator. URL: http://www-mash.cs.berkeley.edu/ns/WWW.html

[Henderson et al., 1998b] Thomas R. Henderson, Emile Sahouria, Steven McCanne, Randy H. Katz (1998b). On Improving the Fairness of TCP Congestion Avoidance. IEEE Globecom Ś98, Sydney, Australia, November 1998. URL: http://daedalus.cs.berkeley.edu

[Jacobson, 1990] V. Jacobson (1990). Compressing TCP/IP Headers for Low-Speed Serial Links. Network Working Group, Request for Comments 1144, February, 1990.

[Kojo et al., 1997] Markku Kojo, Kimmo Raatikainen, Mika Liljeberg, Jani Kiiskinen, Timo Alanko (1997). An efficient Transpor Service for Slow Telephone Links. IEEE Journal on selected areas in communication, VOL. 15, NO. 7, SEPTEMBER 1997. URL: http://www.cs.helsinki.fi/research/mowgli/mowgli-papers.html

[Laamanen, 1995] Heimo Laamanen (1995). An Experiment of Dependability and Performance of GSM Access to Fixed Data Network. Department of Computer Science, University of Helsinki, Report C-1995-41, September 1995. URL: http://www.cs.helsinki.fi/research/mowgli/mowgli-papers.html

[Mah, 1997] Bruce Mah (1997). An Empirical Model of HTTP Network Traffic. INFOCOM Ś97, Kobe, Japan, April 1997. URL: http://www.ca.sandia.gov/ bmah/Papers

[Mathis et al., 1996] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow (1996). TCP Selective Acknowledgements Options, 1996. RFC-2018

[McCanne, 1998] Steven McCanne - University of California at Berkeley and Lawrence Berkeley National Laboratory, Sally Floyd -Lawrence Berkeley National Laboratory (1998). ns Manual Page. URL: http://www-mash.CS.Berkeley.EDU/ns/ns-man.html, ns/ns.1

[Network Research Group, 1998] Network Research Group (1998), Lawrence Berkeley National Laboratory, University of California at Berkeley. ns: UCB/LBNL Network Simulator. 1998. URL: http://www-mash.CS.Berkeley.EDU/ns/.

[Network Research Group, 1997] Network Research Group (1997), Lawrence Berkeley National Laboratory, University of California at Berkeley. nam Manual Page. 4 de Noviembre de 1997. URL: http://titan.cs.uni-bonn.de/ greis/ns/nam.txt, nam/nam.1

[Nguyen et al., 1996] Giao T. Nguyen, Brian Noble, Randy H. Katz, Mahadev Satyanaranyanan (1996). A Traced-Based Approach for Modeling Wireless Channel Behavior. Winter Simulation Conference, December 1996. URL: http://daedalus.cs.berkeley.edu

[Padmanabhan et al., 1998] Venkata N. Padmanabhan, Randy H. Katz.TCP Fast Start: A Technique For Speeding Up Web Transfers (1998). In Proc. IEEE Globecom Ś98 Internet Mini-Conference, Sydney, Australia, November 1998. URL: http://daedalus.cs.berkeley.edu

[Rivanedeyra et al., 1998a] J.M. Rivanedeyra Sicilia and J. Miguel-Alonso (1998a). A Communication Architechture to Access Data Services through GSM. 6th IFIP/ICCC Conference on Information Networks and Data Communications. June 15-17, 1998 Aveiro, Portugal. URL: http://www.sc.ehu.es/acwmialj/publicat.html

[Rivanederyra et al., 1998b] J.M. Rivanedeyra Sicilia, J. Miguel-Alonso (1998b). An Architechture to Access Data Services through Cellular Phone Networks. Euromicro Summer School on Mobile Computing Ś98. Oulu, Finland, Aug. 20-21, 1998. URL: http://www.sc.ehu.es/acwmialj/publicat.html

[Tanenbaum, 1989] Andrew Tanenbaum (1989). Computer Networks 2. ed.. Englewood Cliffs, NJ: Prentince Hall, 1989.

[Wang et al., 1998] Kuang-Yeh Wang, Satish K.Tripathi (1998). Mobile-End Transport Protocol: An Alternative to TCP/IP Over Wireless Links. IEEE INFOCOMŠ98, San Francisco, California, March 1998. URL: http://www.cs.umd.edu/users/kwang

[Welch, 1995] Brent Welch (1995). Practical Programming in Tcl and Tk. DRAFT, Jannuary 13, 1995. URL: ftp://ftp.scriptics.com/pub/mirror/ftp.sunlabs.com/welch/tkbook.ps.gz

[Wetheral, 1996] David Wetheral. Otcl Tutorial (Version 0.96, September 95). URL: ftp://ftptm.lcs.mit.edu/pub/otcl/doc/tutorial.html Acknowledgments