Escola Politècnica Superior
de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Adaptación y test del protocolo 802.11e al simulador ns-2.28

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Alejandro Masalias Falcon

DIRECTOR: David Pérez Díaz de Cerio

DATA: 23 de Febrer de 2006

Títol: Adaptación y test del protocolo 802.11e al simulador ns-2.28

Autor: Alejandro Masalias Falcon

Director: David Pérez Díaz de Cerio

Data: 23 de Febrer de 2006

Resumen

El uso de las redes inalámbricas 802.11 está en continua expansión en la actualidad. Ello provoca que cada vez se requiera en ellas mayor calidad de servicio (QoS) a la hora de usarse para aplicaciones en tiempo real.

A causa de estos requisitos se está desarrollando un nuevo estándar que permite proporcionar calidad de servicio a las aplicaciones que requieren ser transmitidas en tiempo real. Este estándar es el 802.11e.

Una de las herramientas mas usadas en investigación es la simulación, ello permite estimar el comportamiento de una red en un determinado escenario.

El simulador ns-2.28 (Network Simulator-2.28) es un simulador ampliamente utilizado en el campo de la investigación ya que es de código abierto y permite modificar su código, de este modo está en constante evolución.

Ns-2.28 no permite la simulación del estándar 802.11e. Existe un grupo de trabajo denominado TKN de la Universidad de telecomunicaciones de Berlín que ha realizado la adaptación de la versión anterior de NS al estándar 802.11e. El objetivo de este proyecto es adaptar este parche al simulador ns-2.28.

Para finalizar se realizarán simulaciones de escenarios haciendo uso de 802.11e y se compararán los resultados con los obtenidos en el mismo escenario sin usar 802.11e.

Title: Adaptación y test del protocolo 802.11e al simulador ns-2.28

Author: Alejandro Masalias Falcon

Director: David Pérez Díaz de Cerio

Date: 23 de Febrer de 2006

Overview

The use of the 802.11 wireless LAN is in constant expansion at this moment. For this cause, every day is needed from wireless LAN more quality of services for the transmission with need real time.

To improve QoS, in the actuality are developing a new standard that allow provide QoS to the application that requires be transmitted in real time. The standard 802.11e.

One of the methods more used in investigations are simulation models, this allow knowing approximately what can happen in a real scenario.

The simulator ns-2.28 (Network Simulator-2.28) is a simulator very used in the investigation. Ns-2.28 is open source software that allow rewrite the source which permit be in a constant evolution.

In this moment ns-2.28 don't allow the transmission with the standard 802.11e. Exist a work group called TKN form Berlin University that has realized the adaptation of this standard in ns-2.26 version. The objective of this project is adapting this patch to ns-2.28 version.

To finalize will be realized simulation of the sceneries making use of the standard 802.11e and will compare the result with the obtained in the same scenario without 802.11e.

INDICE

INTRODUCCIÓN

En los últimos años, se están extendiendo de manera significativa las redes Wireless LANs (WLANs) estandarizadas según el estándar 802.11 de la IEEE. Como consecuencia se pretende obtener de ellas las mismas prestaciones que en las redes Ethernet al estar siendo sustituidas por WLANs.

Dado el alto throughput de las redes Ethernet, éstas no han presentado deficiencias en cuanto a calidad de servicio (QoS). En cambio, en las redes inalámbricas se requiere de un estándar que garantice QoS en las aplicaciones en tiempo real para mejorar la efectividad del uso de estas redes.

En la capa MAC del estándar 802.11 se incluyen dos modos de acceso al medio: Distributed Coordination Function (DCF) y Point Coordination Function (PCF).

Actualmente existe en desarrollo el protocolo 802.11e, el cual aún no ha sido aprobado pero del que sí se han publicado borradores que permiten hacerse una idea del estándar definitivo. Se introducen dos nuevos modos de operación: Enhanced Distributed Channel Access (EDCA) y Controlled Channel Access (HCCA), los cuales definen mecanismos para permitir QoS.

Este estándar pretende proporcionar calidad de servicio a las redes 802.11 diferenciando el tráfico transmitido en cuatro categorías, dependiendo del audio, video o datos de mayor o menor prioridad.

Gran parte de este proyecto se basa en el análisis y estudio de la documentación disponible sobre redes 802.11 y sobre el protocolo 802.11e.

La finalidad de este proyecto es permitir la simulación del protocolo 802.11e en el simulador de redes ns-2.28. Este se ejecuta bajo el sistema operativo Linux por lo que para empezar se instaló este sistema operativo en mi ordenador. Posteriormente se instaló ns-2.26 y ns-2.28.

La simulación es uno de los métodos mas frecuentes en la investigación de nuevas tecnologías, ya que permite hacerse una idea de que sucedería en un escenario real mediante un ordenador y una herramienta informática (el simulador). Además permite realizar pruebas que no se pueden implementar en un escenario real, porque puede darse el caso en que el hardware no este implementado en la actualidad o requiera costes económicos imposibles de asumir para muchos investigadores.

Uno de los primeros pasos para formarme sobre este simulador fue la lectura del tutorial de Marc Greis y el NS manual, disponible en la página de descarga del simulador. A continuación, después de estudiar dicha documentación, realicé diversas simulaciones escribiendo y modificando los scripts escritos en OTcl para representar distintos escenarios.

Para permitir la simulación en ns-2.28 del protocolo 802.11e, nos basasamos en la adaptación ya realizada para el simulador ns-2.26 por el grupo de trabajo TKN de la facultad de telecomunicaciones de la Universidad de Berlín. Comparando su código con el del simulador sin adaptar y realizando los mismos cambios pero en el simulador ns-2.28. Para ello se instala la herramienta UltraCompare Professional que facilita la búsqueda de diferencias.

Posteriormente se realizan simulaciones con el fin de obtener conclusiones sobre la eficacia de este protocolo.

En la primera parte de este documento se describe el protocolo 802.11 donde se detalla su funcionamiento, modos de acceso, principales variables, etc. Posteriormente veremos que cambios implementa el protocolo 802.11e en el modo de acceso al medio para proporcionar QoS.

Una breve introducción del simulador NS es interesante para conocer el simulador que se usó. A continuación se describe el proceso de trabajo, la instalación de Linux y NS, los primeros scripts, etc. Se concluye el proyecto realizando simulaciones sobre posibles escenarios.

CAPITULO 1. EL ESTÁNDAR 802.11

IEEE 802.11 wireless LAN (WLAN) es una de las tecnologías inalámbricas mas usadas en todo el mundo. Las principales características del 802.11 WLAN son la simplicidad, flexibilidad y la eficiencia. El objetivo del estándar 802.11 es permitir la conectividad sin cables entre dispositivos, ya sean portátiles o fijos, en un determinado entorno local.

En el IEEE 802.11 WLAN, esta definida tanto la subcapa MAC (medium acces control) como la capa física (PHY) del modelo OSI. La subcapa LLC (Logical link layer) esta especificada en el IEEE 802.2.

Esta arquitectura permite una interfaz transparente para los usuarios de capas superiores, las estaciones base pueden moverse por el área de cobertura pareciendo estacionarias para el 802.2 LLC. Esto permite el uso de protocolos como el TCP/IP sobre el 802.11 WLAN.

1.1. Evolución histórica del estándar 802.11

El estándar 802.11 fue aprobado en 1997 por la IEEE y tiene un ancho de banda de 1 a 2 Mbps.

En la capa física están definidas tres interfaces las cuales no son compatibles entre ellas. Una esta basada en comunicación Infrared (IR) y las otras dos operan en la banda de 2.4 GHz. De estas, una de ellas está basada en FHSS (Frecuency Hopping Spread Spectrum) y la otra en DSSS (Direct Sequence Spectrum). Ambos métodos permiten tener muchas transmisiones al mismo tiempo.

FHSS consiste en enviar cada paquete por un canal distinto dentro de la banda, de esta manera se reducen las posibles interferencias externas.

El método DSSS opera en un canal determinado, trabaja tomando un paquete de datos y lo modula con un segundo modelo que es la secuencia de chipping.

1.1.1 Estándar 802.11b

En 1999, la IEEE aprueba el estándar 802.11b High Rate (también llamado Wi-Fi), El límite de velocidad de transmisión asciende a 11 Mbps aplicando CCK (Complementary Code Keying) y DSSS (Direct Sequence Spread Spectrum) como esquemas de transmisión.

La banda de los 2,4 GHz está muy concurrida por teclados inalámbricos, hornos microondas, etc. Ya que esta banda es libre y no se requiere disponer de licencia para transmitir en ella, esto conlleva que a veces esta banda sufra bastantes interferencias.

1.1.2 Estándar 802.11a

Es la respuesta a las necesidades de alto throughput, este estándar puede alcanzar tasas de 54 Mbps y utiliza la banda de 5 GHz, que si bien es libre en EEUU, en Europa es de uso restringido. Como consecuencia del cambio de la banda de trabajo el 802.11a esta prácticamente libre de interferencias, aunque su mayor atenuación provoca que la cobertura de las estaciones sea menor y que los dispositivos 802.11a sean incompatibles con 802.11b y 802.11g. En este estándar se hace uso de Orthogonal Frequency Division Multiplexing (OFDM).

1.1.3 Estándar 802.11g

Es el último de los estándares derivados de 802.11. Combina las ventajas del 802.11b (cobertura) y del 802.11a (alto throughput) usando el esquema de transmisión de OFDM. Trabaja a 2,4 GHz, lo que lo hace compatible con la tecnología ya ampliamente instaurada del 802.11b. La tasa es de 54 Mbps.

Tabla 1.1. Características del 802.11

Estándar	Portadora	Tasa	Modulación
802.11	2.4 GHz	2 Mbps	IR, FHSS y DSSS
802.11b	2.4 GHz	11 Mbps	DSSS
802.11g	2.4 GHz	54 Mbps	DSSS y OFDM
802.11a	5 GHz	54 Mbps	OFDM

1.1.4. Estándares 802.11 en la capa MAC

En la capa de acceso al medio (MAC) existen también algunos estándares del 802.11:

La privacidad y la seguridad son realmente importantes en las redes 802.11. 802.11i pretende solventar estos problemas. Se aplicará a los estándares físicos a, b y g de 802.11 Proporciona una alternativa a la Privacidad Equivalente Cableada (WEP) con nuevos métodos de encriptación y procedimientos de autentificación.

El objetivo del estándar 802.11f es lograr la interoperabilidad de Puntos de Acceso (AP) dentro de una red WLAN. El estándar define el registro de Puntos de Acceso (AP) dentro de una red y el intercambio de información entre dichos Puntos de Acceso cuando un usuario se traslada desde un punto de acceso a otro, es decir realiza un handover.

Un aspecto muy importante en las redes 802.11 es proporcionar soporte de QoS (Calidad de Servicio). De este aspecto se encargará el 802.11e. Aplicará a los estándares físicos a, b y g de 802.11.

Este proyecto se basa en este estándar por lo que posteriormente se detalla su funcionamiento.

1.2. Tipos de redes

1.2.1. Redes IBSS

La forma más simple posible de organizar una red 802.11 es la red ad-hoc. El conjunto de estaciones de una red ad-hoc forman una independent basic service set (IBSS), siendo necesarias como mínimo dos estaciones para poder formarla. En este modo todas las estaciones se comunican entre si directamente, por lo que todas han de estar en el radio de cobertura de las demás. En al siguiente figura tenemos dos IBSS, la primera formada por las estaciones uno, dos y tres y la segunda por las estaciones tres y cuatro.

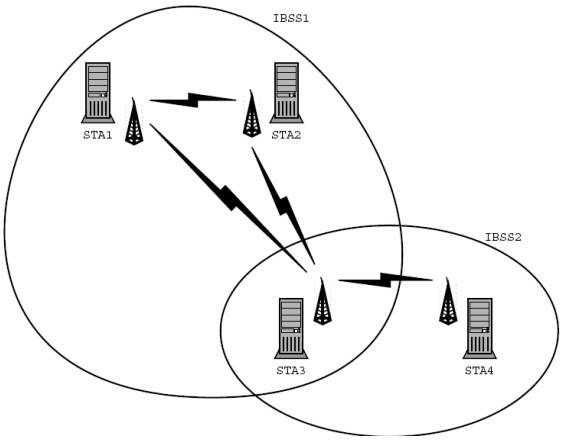

Fig. 1.1. Red IBSS

La pertenencia a una IBSS se implementa mediante una función de asociación, que puede estar controlada por un mecanismo de clave privada, de forma que se asegura que solo las estaciones que conozcan la clave podrán asociarse. Las IBSS son dinámicas, las estaciones pueden entrar y salir del área de cobertura sin mayor problema.

Diferentes IBSS pueden convivir en una misma área, y una misma estación puede pertenecer a diferentes IBSS, que utilizan la combinación de canales de transmisión y tramas de beacon.

Esta arquitectura de red es muy versátil, ya que permite la creación de enlaces a nivel MAC entre estaciones sin necesidad de infraestructura, y sin planificación previa.

Los principales problemas de las redes ad-hoc es que el área de cobertura está limitada por la estaciones con menor radio de cobertura y que las estaciones no

pueden utilizar la IBSS para comunicarse con otras redes. En el ejemplo, la estación cuatro, solo se puede comunicar con la estación tres, por lo que está fuera de la IBSS1, y ha tenido que formar otra IBSS junto con estación tres para poder comunicarse con ella.

Para solucionar estos dos problemas se han creado las redes ESS, que se describen mas adelante.

1.2.2. Redes BSS

Otro modo es el infrastructure basic service set (BSS). La característica de una BSS es que necesita contar con una estación con ciertas capacidades especiales, esto es, un AP, que provee a su BSS de distribution system services (DSS), necesarios para la interconexión con otras redes y con otras estaciones de la misma BSS.

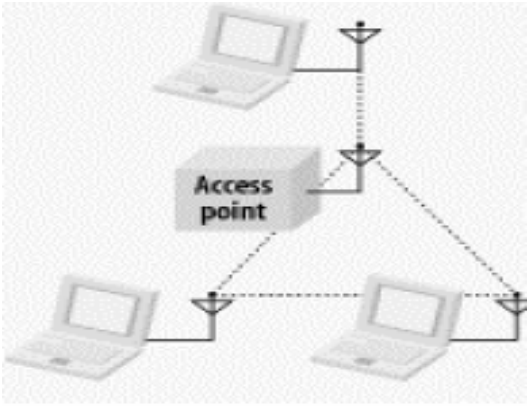

Fig. 1.2. Red BSS

El AP es usado para todas las comunicaciones dentro de la BSS, incluso entre dos estaciones dentro de la BSS. Cuando una estación desea transmitir un paquete a otra, el paquete es mandado de la estación origen al AP y este se encarga de entregarlo a la estación destino.

El área de cobertura del AP delimita la BSS.

1.2.3. Redes ESS

La estrategia para aumentar la cobertura de las BSS es interconectarlas entre si mediante un sistema de distribución (DS), un sistema de distribución es un sistema usado para interconectar diferentes BSS, y otras LANs entre si. De

esta manera varias BSS pasan a formar un conjunto de orden superior, denominado *extended service set* ESS.

Como hemos dicho anteriormente el AP provee a la BSS de los servicios del sistema de distribución (DSS), de esta manera se forma la red ESS.

En la figura se muestra una ESS, formada por varias BSS interconectadas entre si por medio de sus AP, que acceden a un sistema de distribución formado por una red Ethernet.

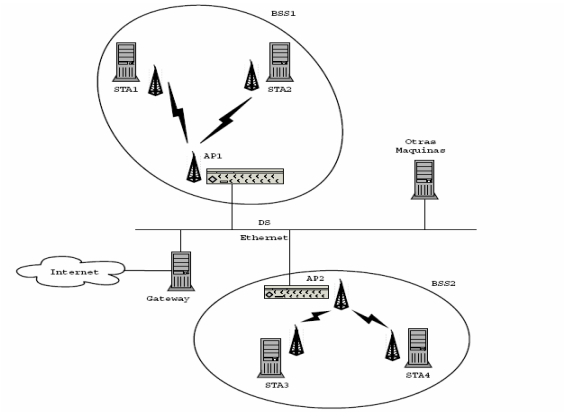

Fig. 1.3. Red ESS

CAPITULO 2. MODOS DE ACCESSO AL MEDIO DCF / PCF

2.1. El protocolo CSMA/CA

CSMA Carrier Sense Multiple Access. Este protocolo consiste en escuchar antes de transmitir. Todas las estaciones pueden acceder al medio ya que este protocolo es implementado para poder hacer uso de un acceso múltiple al canal, pero es imprescindible que comprueben que el canal esta vacío antes de transmitir. Este protocolo es una evolución del ALOHA, el cual transmite sin escuchar el medio.

En ocasiones se puede dar el caso de que dos o más estaciones transmitan en el mismo instante de tiempo y se produzca una colisión entre los paquetes transmitidos. Para reducir la probabilidad de que esto suceda, se evoluciona al estándar CSMA/CA, Carrier Sense Multiple Access with Collision el cual implementa un retardo aleatorio a la hora de transmitir una vez se ha detectado que el canal está libre.

2.2. Tiempos de acceso al medio

Cuando una estación trata de acceder al medio, su prioridad ante el resto de estaciones que también lo intentan varía en función del tiempo de acceso de esta, dependerá del tipo de tráfico a transmitir.

Es lógico, si una estación tarda menos en intentar la transmisión, esta ganara el canal antes que otra que tarde mas, ya que una vez el canal este ocupado, el resto de estaciones deberán esperar a que vuelva a estar vacío.

Como en la red Ethernet tradicional, los tiempos para acceder al medio juegan un papel importante para coordinar el acceso al medio de transmisión. En 802.11 se usan tres tiempos distintos los cuales son usados para determinar el acceso al medio.

En la siguiente figura se muestran estos tiempos:

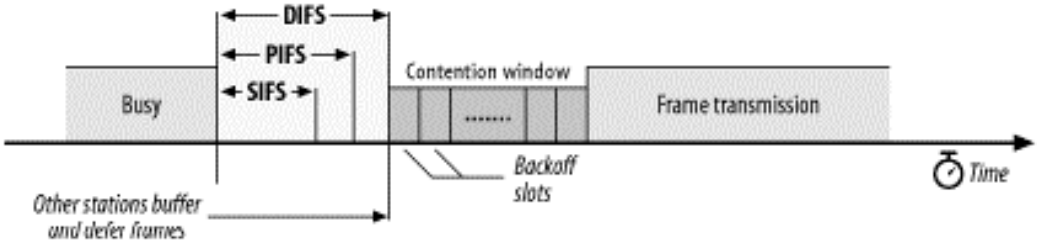

Fig. 2.1. Tiempos de acceso

2.2.1. Short interframe space (SIFS)

El tiempo SIFS es usado par transmisiones de mayor prioridad, como tramas RTS/CTS y reconocimientos positivos de entrega de paquete, ACK. Las transmisiones de mayor prioridad pueden acceder al medio una vez transcurrido un tiempo SIFS. Una vez se ocupa el medio, el resto de transmisiones de menor prioridad no podrán transmitir y deberán esperar a que el medio vuelva a estar vacío.

El valor del tiempo SIFS variará en función del estándar 802.11 que estemos usando.

2.2.2. PCF interframe space (PIFS)

El tiempo PIFS es únicamente usado por el point coordinatio function (PCF) durante el periodo libre de contención. Las estaciones con datos a transmitir durante este periodo, podrán hacerlo una vez trascurrido un PIFS.

El tiempo PIFS se puede calcular mediante la siguiente formula:

$$PIFS = SIFS + SlotTime \qquad \textbf{(2.1)}$$

Donde el valor del SlotTime dependerá del estándar que estemos usando.

2.2.3. DCF interframe space (DIFS)

Es el tiempo mínimo que debe permanecer el medio vacío para poder transmitir durante el periodo de contención. Las estaciones podrán acceder al medio directamente siempre que el medio haya estado vacío durante un tiempo DIFS. El tiempo DIFS se puede calcular mediante la siguiente formula:

$$DIFS = SIFS + 2 * SlotTime \qquad \textbf{(2.2)}$$

2.2.4. Extended Inter-Frame Space (EIFS).

Es el mas largo de los tiempos. Solo se usa cuando ha habido un error en la transmisión.

Tabla 2.1. Los valores SlotTime, SIFS, PIFS, DIFS y CW son los siguientes:

Estándar	SlotTime μs	SIFS μs	DIFS μs	PIFS μs	CW min	CWmax
802.11a	9	16	34	25	15	1023
802.11b	20	10	50	30	31	1023
802.11g	*9 o 20	10	28 o 50	19 o 30	15 o 31	1023

*En 802.11g puede optar por transmitir en Short SlotTime que sera de 9 μs

Estos valores han sido obtenidos apartir de los estándares de 802.11 citados en la bibliografía. [6]

Existen dos modos de acceso al medio radio:

-DCF (Distributed Coordination Function). El cual usa transmisión asíncrona. Este modo de acceso al medio radio, esta basado en el protocolo CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) y debe ser implementada en todas las estaciones de la red. El modo DCF puede ser usado tanto en modo ad-hoc como en modo infraestructura

-PCF (Point Coordination Function). En este caso se usa transmisión síncrona, y únicamente puede ser implementado con la presencia de un punto de coordinación que suele ser el AP (Acces Point), es decir solo puede ser usada en modo infraestructura. El AP va dando la oportunidad de transmitir a cada una de las estaciones base que forman la BSS.

2.3. DCF

El modo DCF permite interactuar a varias estaciones independientes sin un centro de control, de esta manera puede ser usado tanto en una red IBSS como BSS.

El modo DCF actúa de la siguiente manera. Cuando una estación quiere transmitir, escucha el medio, si no esta ocupado se espera un tiempo DIFS y se transmite directamente. En el caso de que este ocupado, se espera a que finalicen las transmisiones, una vez finalizadas se espera un tiempo DIFS y posteriormente con el objetivo de disminuir el número de colisiones, empieza un periodo de contención, Backoff.

La estación genera un número aleatorio que será un intervalo de tiempo denominado Backoff time, uniformemente distribuido entre cero y el tamaño de la ventana de contención.

Para reducir la probabilidad de colisión, después de cada intento de transmisión fallido, la ventana de contención doblara su tamaño hasta un máximo establecido (CWmax), de lo contrario cada vez que un paquete es

transmitido correctamente la ventana de contención pasara a tener el tamaño mínimo permitido (CW min), así se mejora el rendimiento del canal.

Cada vez que el medio deja de estar ocupado, la estación espera un tiempo DIFS y luego continúa decreciendo el tiempo backoff. Cuando el tiempo backoff llega a cero la estación accede al medio para transmitir. En la **figura 2.2** se muestra el proceso. Obviamente, cuando dos estaciones transmiten en el mismo instante de tiempo se produce una colisión. A diferencia de las redes donde el medio físico es un cable, como en una red ethernet, en la cual se usa el protocolo (CSMA /CD), cuando se produce una colisión, no se puede detectar ya que el nivel de señal transmitido es muy superior al nivel de señal recibido y este es enmascarado por el transmitido. Como no se puede detectar la colisión, lo que hacemos es esperar un paquete ACK (acknowledgement) positivo cada vez que transmitimos, de lo contrario supondremos que no ha llegado a su destino y se retransmitirá el paquete. Para la transmisión del paquete ACK, la estación receptora espera un tiempo SIFS (Short Inter Frame Space) después de la recepción del paquete. Un tiempo SIFS es menor que un tiempo DIFS, por lo cual se asegura que dos estaciones terminan su dialogo antes de que otra estación intervenga el medio.

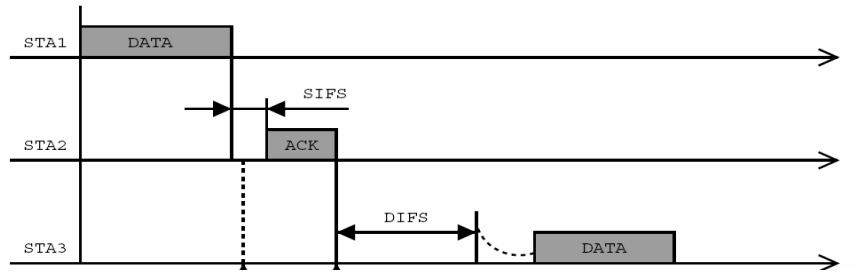

Fig. 2.2. Tiempos de espera para una transmisión en DCF.

Sin embargo el sistema hasta ahora descrito tiene un problema: estamos suponiendo que una estación es capaz de saber si el medio está ocupado o no, simplemente escuchando. Esto, en redes inalámbricas centralizadas es imposible, ya que una estación no tiene por que estar en el radio de cobertura de otra estación. A esta situación se la denomina el problema de los nodos ocultos.

Si una estación A no puede detectar si una estación B está transmitiendo o no, el mecanismo de CSMA/CA degenera en un simple ALOHA, donde los terminales transmiten cuando lo necesitan, sin importar si hay una transmisión en progreso. Para evitar esto se complementa el CSMA/CA con el envío de paquetes RTS/CTS.

El sistema de RTS/CTS consiste en, antes de enviar los datos, calcular el tiempo total que estará el canal ocupado por esta transacción y enviarlo al receptor en una trama corta, denominada RTS (Request To Send o solicitud para enviar). El receptor de un RTS lo duplica y lo manda en un CTS (Clear To

Send o permiso para enviar) a broadcast. Se puede ver un croquis del proceso en la **figura 2.3.**

De esta manera todos los nodos de la BSS susceptibles de interferir con la comunicación, están al tanto de la duración de la misma. Cuando un nodo recibe un CTS, almacena la información de cuanto va a estar el canal ocupado en un NAV (Network Allocation Vector). La función de este vector es indicar el tiempo previsto de uso del canal, de esta manera una estación puede saber cuando el canal estará libre.

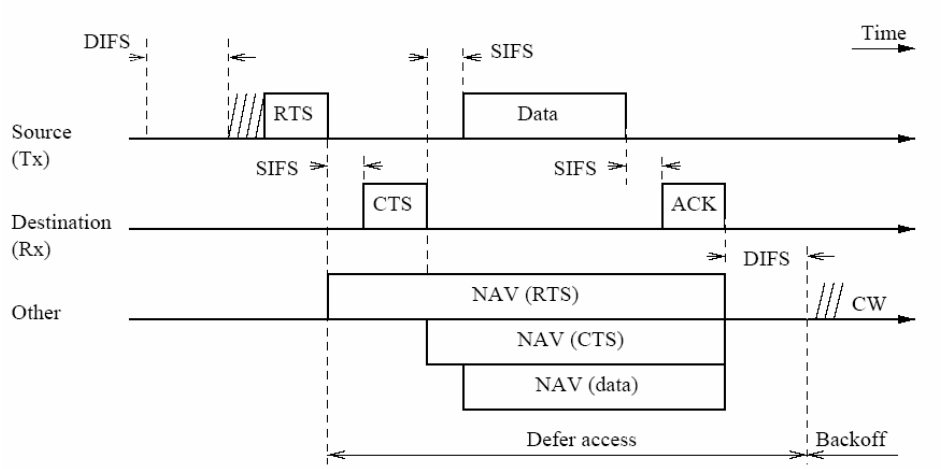

Fig. 2.3. Proceso RTS/CTS.

Mientras el NAV en el nodo receptor del CTS le indique que no debe transmitir, permanecerá callado, incluso si aparentemente, para él, el canal está libre.

De esta manera la posibilidad de colisión por nodo oculto se ve disminuida enormemente, ya que ahora solo pueden colisionar los RTS/CTS, que son tramas de duración muy corta.

2.4. PCF

Este modo de acceso al medio radio permite la transmisión síncrona de datos, lo que mejora la capacidad de transmisión de datos en tiempo real.

En esta configuración, el AP actúa de coordinador del resto de las estaciones, eligiendo quien ha de transmitir en cada momento. El AP tiene prioridad en cuanto al acceso al medio ya que utiliza Un tiempo PIFS el cual es menor que un tiempo DIFS por lo que el AP siempre ganará el acceso al canal y ninguna estación de la red que trabaje en modo DCF le será posible interrumpir el modo PCF.

De esta manera, cuando hay que hacer una asignación de tiempos de transmisión el PC (Point Coordiantion) primero escucha el canal durante un

periodo PIFS (PCF InterFrame Space) y luego empieza el CFP (Contention-free period) mandando una señal beacon a broadcast. Un componente de la trama beacon indica la máxima duración del periodo CFP. Todas las estaciones que reciben la trama beacon, guardan en su vector NAV dicha duración, de manera que no transmitirán durante este periodo, de esta manera se evitan interferencias.

Durante el modo PCF, el PC va dando el turno de transmitir a las estaciones en las cuales sus datos requieren ser transmitidos en tiempo real (video, voz, etc). Una estación no puede transmitir a no ser que el AP se lo haya autorizado. El AP va dando el turno de transmitir a partir de una lista de las estaciones denominada polling list, las estaciones solo acceden a dicha lista cuando están asociadas al AP.

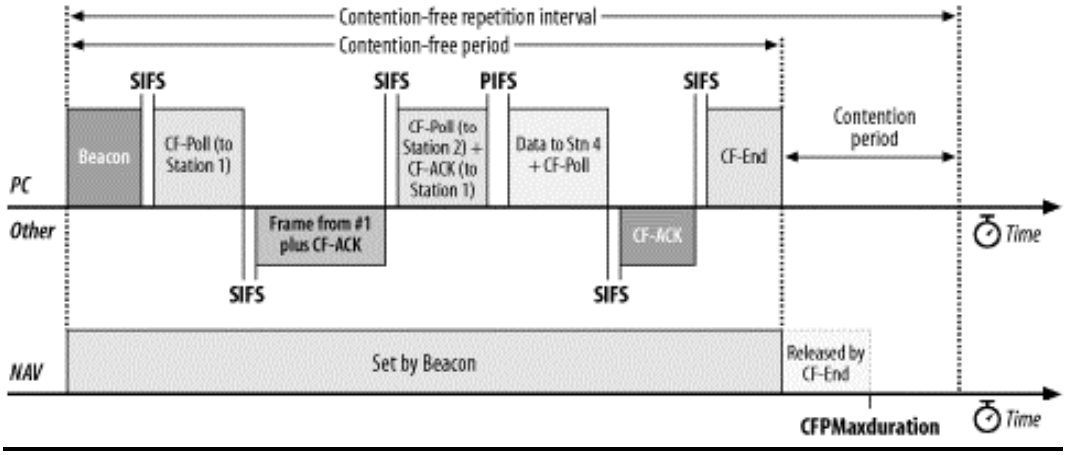

Fig. 2.4. Tiempos de espera para una transmisión en PF.

Cuando le llega el turno de transmitir a una estación, el PC le envía un paquete CF-Poll. La estación transmite en un tiempo igual a un SIFS. Los paquetes siempre van separados por un tiempo SIFS excepto para el caso en que la estación la cual tiene le turno de transmitir no responde al CF-Poll durante un periodo SIFS, entonces el PC envía el próximo paquete después de un tiempo PIFS desde el final de la última transmisión

Este método en el cual el AP domina la red y controla el acceso al medio puede ser ineficiente en redes grandes. Además, todo el tráfico debe ir a través del AP de manera que no se optimiza el ancho de banda.

Normalmente tanto el DCF como el PCF conviven a la vez en la red, el AP debe dejar el suficiente tiempo en modo DCF, para que las estaciones puedan transmitir datos por acceso distribuido. Para ello, el PC puede finalizar el modo PCF enviando una trama CF-end a broadcast devolviendo así el control de acceso al medio al modo DCF. Esto ocurre frecuentemente cuando la red esta cargada.

CAPITULO 3. EL PROTOCOLO 802.11E

El estándar 802.11e fue desarrollado para solventar los problemas que presentaba el 802.11 a la hora de ofrecer calidad de servicio. El modo de acceso al medio DCF, no aporta ninguna posibilidad de priorizar diferentes tipos de tráficos, todas las estaciones tienen la misma probabilidad de acceder al canal. Por este motivo se desarrollo, una mejora del DCF, el EDCA (Enhanced Distributed Channel Access Function). Este mecanismo ofrece la posibilidad de priorizar distintos tipos de tráfico a la hora de ganar el acceso al canal durante el periodo de contención. El mecanismo EDCA incluye, al igual que su antecesor DCF, el protocolo CSMA/CA, los tiempos backoff, etc.. También existe la evolución de PCF hacia el HCCA (Hybrid Coordination Function Controlled Channel Access).

3.1. Calidad de servicio

Calidad de servicio se define como el efecto global de la calidad de funcionamiento de un servicio que determina el grado de satisfacción de un usuario.

La calidad de servicio de una aplicación, depende de factores como el throughput, el retardo de los paquetes de dicha aplicación, la probabilidad de error, etc.

En función del tipo de aplicación, será mayor la importancia de un factor u otro. Por ejemplo, si deseamos transmitir datos, no es tan relevante el throughput ni el retardo de llegada de los paquetes a su destino, pero si que lo será si estos llegan a su destino con una tasa de error elevada. Supongamos que lo que estamos enviando es una aplicación software, si los datos contienen muchos errores no se podrá ejecutar correctamente, en cambio si llegan con retardo no tiene importancia en lo que aplica a calidad de servicio.

No serán tan relevantes los errores en los paquetes recibidos en el caso de aplicaciones en tiempo real. Para estas, el throughput y el bajo retardo de los paquetes es prioritario, ya que estos factores permiten la recepción de los datos en tiempo real. Es mas, si un paquete se recibe con errores, este se descarta en cambio de pedir la retransmisión ya que si un paquete llega con un retardo excesivo no sirve de nada. Si durante la transmisión no se puede alcanzar la tasa de generación de tráfico, no se podrán entregar todos los paquetes generados en tiempo real y esto, deteriorará la QoS de la aplicación.

El modo de acceso al medio EDCA intenta mejorar la QoS durante la transmisión.

3.2. EDCA

Para poder proporcionar calidad de servicio el 802.11e usa distintas colas en función del tipo de tráfico. Cada categoría de acceso corresponde a una prioridad, y cada una de estas colas tiene sus propios parámetros de contención y su propio mecanismo de backoff. En la siguiente figura se muestra las distintas categorías de tráfico.

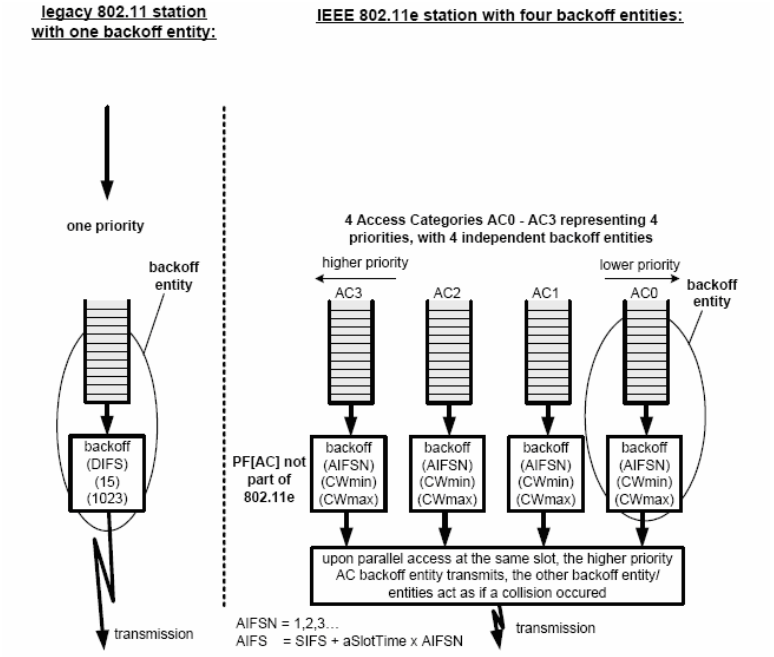

Fig. 3.1. Cada estación 802.11e tiene cuatro categorías de trafico (AC).

Se podría dar el caso de que en una misma estación, dos o más colas terminaran el proceso de backoff al mismo tiempo, en este caso para evitar el conflicto transmitiría la de mayor prioridad y el resto se comportarían como si hubiera habido una colisión en el medio.

Sabemos que cada una de las colas se corresponde con una prioridad, vamos a ver que parámetros logran conseguir dicho objetivo.

3.2.1 AIFS (Arbitrarion Interframe Space)

Cada cola de tráfico tiene un valor distinto. Es el tiempo que debe esperar antes de intentar transmitir o de empezar el periodo de contención. Su tarea es la misma que el DIFS usado en DCF. Se calcula del siguiente modo:

$$AIFS[AC] = SIFS + AIFSN[AC] * SlotTime \quad \textbf{(3.1)}$$

El AIFSN[AC] es el valor que define el tamaño de cada AIFS en cada una de las colas. Cuanto más pequeño sea este valor menor tiempo de espera para acceder al medio y por lo tanto mayor probabilidad de acceder. Lógicamente cuanto menor sea el AIFSN[AC] de mayor prioridad será la cola. Los tiempos AIFS[AC] se muestran graficametne en la **figura 3.2** .
Por definición, para la cola de mayor prioridad un AIFS será igual a un DIFS, es decir AIFSN[AC] = 2 , ya que recordemos que $DIFS = SIFS + 2 * SlotTime$

3.2.2. CW[AC]

CWmin[AC] y CWmax[AC]. Son los tamaños de la ventana de contención (backoff) mínimo y máximo que variaran también en función de la prioridad de la cola. Para tamaños menores, mayor prioridad ya que de este modo, el tiempo backoff será también menor y por lo tanto accederá al medio antes.
Como en DCF el temporizador backoff se para si se detecta que el medio esta en uso y disminuye si no es usado durante un tiempo AIFS.

3.2.3. TXOPLimit[AC]

Un TXOP (Transmisión Opportunity) es un intervalo de tiempo en el cual una estación tiene la oportunidad de iniciar una transmisión, se define desde el momento en que empieza a transmitir y para la máxima duración. Este parámetro permite aumentar el throughput del sistema mediante una asignación adecuada de la duración de este parámetro para cada Acces Category [AC]. En DCF cada vez que una estación gana el acceso al medio le es permitido enviar un único paquete.

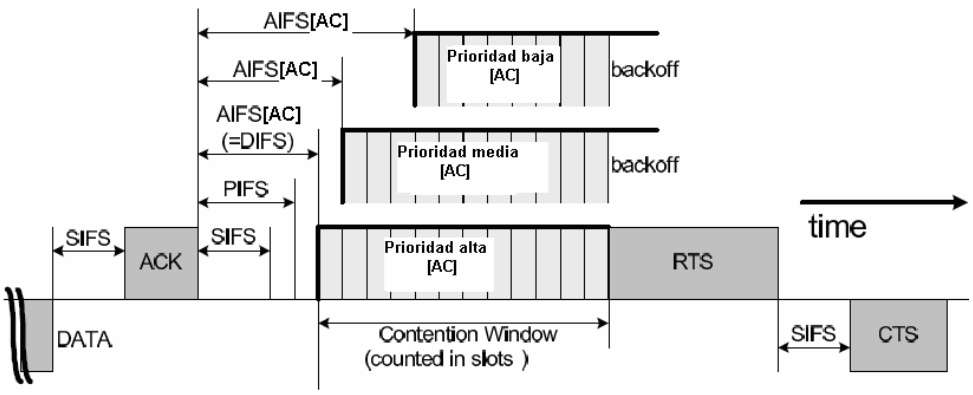

Fig. 3.2. Tiempos de acceso al medio y CW para una transmisión en EDCA.

Tabla 3.1. Los parámetros según su prioridad son los siguientes:

802.11b	AIFSN[AC]	CWmin (SlotTime)	Cwmax (SlotTime)	TXOP (μs)
Prio 0	2	2	15	3274
Prio 1	2	15	31	6016
Prio 2	3	31	1023	0*
Prio 3	7	31	1023	0*

802.11g	AIFSN[AC]	CWmin (SlotTime)	Cwmax (SlotTime)	TXOP (μs)
Prio 0	2	3	7	1504
Prio 1	2	7	15	3008
Prio 2	3	15	1023	0*
Prio 3	7	15	1023	0*

* El valor 0 indica que solo se transmite un paquetes una vez se acceda al medio.

Estos valores han sido obtenidos apartir del estándar 802.11e citado en la bibliografía. [6]

3.3. HCCA

En 802.11e el periodo libre de contención es gestionado por el modo de acceso al medio HCCA. Este es la base de EDCA y controla tanto le periodo de contención como el periodo libre de contención. Como en modo PCF, en modo HCCA una estación debe gestionar el acceso al medio del resto de la BSS. En HCCA, esta estación es denominada HC (Hybrid Coordinator) y normalmente se aloja en el AP. Esta estación puede transmitir con prioridad ante el resto de estaciones de la BSS haciendo uso del tiempo PIFS ya que AIFS ≥ DIFS > PIFS.

Existen dos TXOP en HCCA:

EDCA-TXOP: Es solo obtenida en el periodo de contención

Polled- TXOP: Puede ser obtenida por el HC en cualquier momento.

Las TXOP son asignadas por el HC durante el periodo libre de contención gracias a las tramas CF-poll enviadas por este a la estación la cual tendrá el turno de transmitir, en el CF-poll se indica el momento en que puede empezar a transmitir y la máxima duración de la transmisión. El HC enviará una trama CF-end para indicar el comienzo del periodo de contención.

CAPITULO 4. EL SIMULADOR NS

NS comenzó en 1989 como una variante al ya existente REAL network simulator y ha ido evolucionado en los últimos años. En 1995 el desarrollo de NS pasó a manos de DARPA (Defense Advanced Research Projects Agency) a través del proyecto VINT (Virtual InterNetwork Testbed). Actualmente ha quedado en manos de un grupo de investigadores y desarrolladores de la Universidad de Berkeley.

El software es de código libre, por tanto no es un producto acabado y continúa en desarrollo para mejorarlo.

NS (network simulator) es un simulador de eventos discretos centrado en la investigación sobre redes. La Web de este simulador se encuentra citada en la bibliografia [0]. NS permite simulaciones de redes para TCP, routing y multicast sobre redes cableadas o inalámbricas (locales y por satélite).

El simulador NS esta implementado en lenguaje de programación C++. Se elige este lenguaje ya que la simulación detallada de los protocolos requiere un lenguaje de programación que permita manipular eficientemente bytes, paquetes, cabeceras e implementar algoritmos. Para éstas tareas es importante la velocidad de procesado y C++ lo permite.

Para utilizarlo, el usuario puede interactuar directamente con el simulador, a través de un lenguaje de interface. El funcionamiento es el siguiente:

Se escribe en un script en OTcl una versión de Tcl orientada a objetos. Una serie de comandos donde se define los protocolos que se usaran en la simulación, la estructura del escenario a simular (número y tipo de nodos, conexiones entre ellos, propiedades de BW y de retardo, características del trafico que se va a utilizar, etc..). De esta manera se proporciona a NS diversos datos, algunos datos pueden ser también facilitados en forma de ficheros.

De este modo se permite modificar rápidamente el escenario de simulación sin tener que compilar de nuevo todo el código.

Posteriormente se llama al núcleo de NS tecleando el comando ./ns haciendo referencia al script y empieza la simulación. Conforme avanza la simulación, se generan un conjunto de datos de salida que se almacenan en un fichero de traza.

A partir de los archivos de trazas, se analizan posteriormente los resultados de la red simulada, ya sea con xgraph para ver graficas o filtrando el fichero de trazas con otros lenguajes de programación como Perl para obtener los resultados deseados.

También puede usarse herramientas como Network Animador (nam) la cual permite observar visualmente el envío y recepción de paquetes entre nodos.

CAPITULO 5. PROCESO DEL TRABAJO

Como ya he comentado anteriormente, la ejecución de una simulación en NS se realiza mediante un script escrito en lenguaje OTcl donde se detallan los parámetros del escenario. El primer paso a seguir fue documentarme sobre este lenguaje de programación y tratar de realizar un script con una simulación sencilla.

Para ello consulté el manual Marc Greis's tutorial. Citado en la bibliografía [2]. El tutorial escrito por Marc Greis es un tutorial breve y preciso el cual detalla paso a paso como generar un script.

5.1. Instalación de Linux y NS.

El primer paso a realizar fue la instalación de las herramientas informáticas necesarias para llevar a cabo el objetivo del proyecto.

En primer lugar se realizó una partición del disco duro para así poder instalar el sistema operativo Linux. La versión elegida fue SUSE LINUX Professional 9.2.

Una vez instalado Linux se procedió a la instalación del simulador NS-2. Descargamos el NS-2 de la red [0]. El formato es un tar.gz que hay que descomprimir en mediante la siguiente sentencia:

```
tar -xvzf ns-allinone-2.28.tar.gz
```

Una vez descomprimido el fichero descargado de la red, debemos ejecutar el comando de instalación en la carpeta donde se encuentra NS.

```
./install
```

Cualquier modificación realizada posteriormente en cualquiera de los archivos de NS, debe compilarse de nuevo ejecutando la sentencia:

```
./configure
```

Y a continuación

```
make
```

5.2. Primer Script Tcl

5.2.1. Dos nodos enlazados mediante una línea duplex.

Para entender la estructura de un script, se empezó por una simulación muy básica donde se definen dos nodos, unidos entre ellos mediante un enlace bidireccional por el cual se envían paquetes con un tamaño y una tasa de generación de tráfico definidos por el usuario. En este script también podemos ver los ficheros de trazas que podemos generar mediante unas sentencias ya definidas. La estructura es la siguiente:

Lo primero que se debe crear es el objeto del simulador mediante el comando:

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

Abrimos los ficheros de traza:

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

set tracefd     [open simple.tr w]
$ns trace-all $tracefd
```

Creamos los nodos

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

Creamos un enlace entre los dos nodos

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

Generamos el tráfico con los parámetros que nos interese

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

Creamos un agente que será el que recibirá todo el tráfico y lo asociamos al nodo1

```
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n1 $null0
```

Posteriormente se define el instante de tiempo en el que los agentes empiezan a generar tráfico y cuando finaliza la simulación.

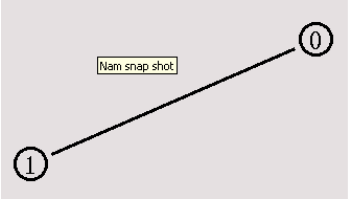

Fig. 5.1 Escenario generado mediante el Script

Este script es muy sencillo ya que no aparece ninguna infraestructura jerárquica.

A continuación veremos un script donde aparecen nodos conectados entre si mediante un enlace wireless. Este es el objetivo ya que lo que pretendemos es simular escenarios donde los nodos usen el protocolo 802.11e

El script completo aparece en el anexo [1].

5.2.2. Simulación con nodos Wireless

Para la simulación de nodos wireless debemos definir los parámetros de dichos nodos como el tipo de canal, tipo de cola o el protocolo de la capa MAC usado.

Al inicio del script declaramos ciertas variables globales:

```
set val(chan)       Channel/WirelessChannel      ;# Tipo de canal
set val(prop)       Propagation/TwoRayGround     ;#Modelo de
                                                      propagación radio
set val(ant)        Antenna/OmniAntenna          ;# Tipo de antena
set val(ll)         LL                           ;#Tipo  Link layer
set val(ifq)        Queue/DropTail/PriQueue      ;# Tipo de cola
set val(ifqlen)     50                           ;#Numero maximo de
                                                      paquetes en ifq
set val(netif)      Phy/WirelessPhy              #Tipo de network interface
set val(mac)        Mac/802_11                   ;#Tipo de MAC
set val(rp)         DSDV                         ;#Protocolo de
                                                      enrutamiento ad-
hoc
set val(nn)         [expr $n1 + $n2 + 1]         ;# numero de nodos
                                                      móviles
```

Antes de crear el nodo lo configuramos:

```
$ns_ node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
```

```
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-channel $chan \
-wiredRouting ON \
-agentTrace ON \
-routerTrace OFF\
-macTrace OFF \
-movementTrace OFF \
-eotTrace OFF
```

Aquí se define también que parámetros deseamos guardar en nuestro fichero de trazas, datos sobre el agente, de enrutamiento o sobre la capa mac. En el ejemplo anterior únicamente está activado el agentTrace por lo que se guardarán datos sobre el agente (generación y recepción de paquetes por el agente en cada uno de los nodos, etc.)

En simulaciones con redes wireless se debe definir siempre el god, donde se especifica el número total de nodos wireless.

```
create-god $numero de nodos
```

En una simulación, puede haber nodos cuya conexión es cableada o bien vía wireless. Si deseamos tener ambos tipos de conexión en nuestra simulación debemos definir el enrutamiento jerárquico. Donde se especifica el número de dominios, numero de cluster por dominio y numero de nodos en cada cluster. Se define del siguiente modo:

```
$ns_ node-config -addressType hierarchical

AddrParams set domain_num_ 3          # numero de dominios
lappend cluster_num 2 1 1             # numero de cluster en cada dominio
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 1 2 1           # numero de nodos en cada cluster de cada
                                        dominio
AddrParams set nodes_num_ $eilastlevel
```

Hasta aquí una breve introducción sobre como crear un script para simular una red IBSS. A continuación veremos que modificaciones se deben realizar durante la creación de los nodos para asignar a uno de ellos la función de AP. Una vez exista un AP podremos realizar simulaciones de una red BSS. Los cambios a la hora de definir los nodos son los siguientes:

El AP se crea del siguiente modo:

```
set AP(0) [$ns_ node [lindex $temp 0]]          ;#direccion del AP
```

Configuramos su mac:

```
set macAP [$AP(0) getMac 0]
set direccionAP [$macAP id]
```

```
$macAP bss_id $direccionAP
```

La creación de los nodos:

```
set movil($i) [$ns_ node [lindex $temp $i]]
$movil($i) base-station [AddrParams addr2id [$AP(0) node-addr]] ;#Indicamos cual es el AP
```

Configuramos su mac:

```
set mac [$movil($i) getMac 0]
$mac bss_id $direccionAP        ;# Indicamos cual es la dirección mac del AP
```

Ahora ya sabemos generar un script donde se simula una red wireless en modo infraestructura.

El script completo aparece en los anexo [1]

Podemos elegir que estándar usar (802.11b, 802.11g o 802.11a) en nuestro simulación definiendo los parámetros de ventana de contención (CW), difs, pifs, Slot time, etc, especificados para cada uno de los estándares. (En la **Tabla 2.1** aparecen dichos valores).

La modificación del valor de los parámetros la podemos realizar en el fichero tcl/lan/ns-mac.tcl o bien en nuestro script TCl según los comandos siguientes:

```
Mac/802_11 set delay_ 64us
Mac/802_11 set ifs_ 16us
Mac/802_11 set slotTime_ 20us
Mac/802_11 set cwmin_ 31
Mac/802_11 set cwmax_ 1023
Mac/802_11 set rtxLimit_ 16
Mac/802_11 set bssId_ -1
Mac/802_11 set sifs_ 10us
Mac/802_11 set pifs_ 30us
Mac/802_11 set difs_ 50us
Mac/802_11 set rtxAckLimit_ 1
Mac/802_11 set rtxRtsLimit_ 3
Mac/802_11 set basicRate_ 11Mb
Mac/802_11 set dataRate_ 11Mb
```

Y en la 802.11e:

```
Mac/802_11e set delay_ 64us
…
…
```

5.2.3. Ficheros de traza

Como se ha comentado anteriormente, se pueden generar de manera automática ficheros de trazas donde se guardan los datos de la simulación.

Fichero donde se guardan parámetros de la transmisión de paquetes entre otros. Las trazas guardadas las definimos en el script al configurar el nodo. Este fichero es el más interesante puesto que nos proporciona resultados del tráfico.

En cada línea de este fichero se definen datos como si el paquete es enviado, recibido o si se pierde (drop), el instante de la acción, el nodo destino y origen, el número y tamaño del paquete, etc.

Creación del fichero:

```
set tracefd     [open simple.tr w]
$ns trace-all $tracefd
```

En el fichero de traza nam se guardan las trazas para posteriormente poder visualizar gráficamente el envío de paquetes mediante la herramienta nam (Network Animator).

Creación del fichero:

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

Para mayor detalle se puede consultar el CAPÍTULO 24. Trace and Monitoring Support del manual ns manual referenciado en la bibliografía. **[1]**

5.2.4. Procedimiento record

A parte de guardar datos de la simulación mediante los ficheros de trazas, también existen otros métodos de generar ficheros de salida. En las simulaciones realizadas en el capitulo 6, se hace uso del procedimiento record que nos sirve para generar un fichero de salida donde se almacenan los datos que nos interesan. En este caso los datos de throughput para cada una de las fuentes de tráfico.

La estructura de la función es la siguiente:

Generamos un fichero de salida:

```
set f0 [open out.tr w]
```

Procedimiento record

```
proc record {} {
        global sink0 f0

  #Instancia del simulador
        set ns [Simulator instance]
```

```
 #Frecuencia de ejecución del procedimiento record
        set time 1.0

#Bytes recibidos en el agente sink
        set bw0 [$sink0 set bytes_]

#Tiempo actual
        set now [$ns now]

#Se calculan los Mbps recibidos y se guardan en el fichero de salida
        puts $f0 "$now [expr $bw0/$time*8/1000000]"

#Reseteamos el valor de los bytes recibidos en el agente sink
        $sink0 set bytes_ 0

#Relanzamos el procedimiento.
        $ns at [expr $now+$time] "record"
}
```

De este modo en el fichero de salida open out.tr, tendremos los Mbps recibidos cada segundo. A partir de este fichero calcularé el throughput y lo representaré gráficamente.

5.2.5. Calculo del retardo

Para el cálculo del retardo se hace uso del fichero de traza. Como he comentado posteriormente, en este fichero aparecen todos los datos de cuando ha sido transmitido el paquete, cuando ha sido recibido, los nodos de origen y destino, etc.

Sabiendo que tipo de tráfico genera cada nodo y que tamaño tienen los paquetes en cada tráfico, se filtran los paquetes según el tipo de tráfico. Posteriormente se calcula la diferencia entre el instante en que es enviado y el instante en que es recibido. De esta manera se determinar el retardo de la transmisión. Se puede calcular el retardo desde que el paquete es enviado desde el agente origen hasta que es recibido en el agente destino o bien entre que es enviado desde la capa MAC origen hasta que es recibido en la capa MAC destino.

Para calcular el retardo he usado Microsoft Excel. Existen múltiples modos de analizar este fichero, como implementando una pequeña aplicación mediante herramientas de programación.

5.3. Adaptación del simulador ns-2.26 al protocolo 802.11e

Una vez aprendido como crear una simulación mediante un script, el siguiente paso fue buscar la adaptación al 802.11e para ns-2.26.

Tras buscar por Internet encontré un grupo de trabajo denominado TKN (Telecommunication Netwrk Group) de la facultad de telecomunicaciones de la Universidad de Berlín, los cuales han desarrollado un parche para permitir el uso del protocolo 802.11e en el simulador ns-2. 26. Este docuemento se encuentra en la Bibliografía, [5].

El siguiente paso ha sido analizar el trabajo realizado por este grupo a la hora de adaptar el protocolo y que modificaciones se han realizado en el código original de ns-2.26 para realizar dicha adaptación.

5.3.1. Ficheros modificados y ficheros añadidos para adaptar el protocolo 802.11e

El protocolo 802.11e es una modificación del modo de acceso al medio sobre el 802.11 para dar calidad de servicio. Esto se realiza en la capa mac por lo que todas las modificaciones del código original del simulador NS se realizarán en la capa mac.

Para adaptar el simulador ns-2.26 al protocolo 802.11e mediante el parche desarrollado por el grupo TKN, se debe descomprimir en el directorio ns-x.y/mac/802.11e, los siguientes archivos:

-mac-802_11e.cc / mac-802_11e.h

En este archivo se encuentra el modelo de simulación 802.11e, es la base de la adaptación de NS. Se definen las características de acceso al medio para el protocolo 802.11e. Su estructura es muy parecida a mac-802_11.cc pero con ciertas modificaciones.

-mac-timers_802_11e.cc / mac-timers_802_11e.h

En la capa mac existen varios temporizadores:

-defer timer
-backoff timer
-interface timer
-send timer
-nav timer

Los métodos que permiten inicializar, detener y pausar estos temporizadores son implementados en esta función. En el simulador NS ya existe un fichero llamado **mac-timers_802_11.cc,** en el cual se gestiona estos temporizadores. **mac-timers_802_11e.cc** es una modificación del fichero original. Un fichero u otro va a ser llamado en función de si usamos tráfico con prioridad o no.

En los siguientes ficheros se encuentran los métodos que se encargan de la gestión de las colas:

-priq.cc / priq.h

El modelo 802.11e define cuatro colas según el tipo de tráfico. La prioridad de cada paquete de tráfico viene definido en la cabecera del paquete. El método que se encarga de decidir en cual de las cuatro colas se inserta el paquete esta implementado en este fichero. El tipo de cola DTail es una mejora de las usadas en 802.11 drop-tail. Ahora habrá cuatro colas drop-tail, una para cada prioridad.

-priority.tcl

Los parámetros (AIFS, CW y TXOP) para cada prioridad están relacionados a su correspondiente cola drop-tail. Los valores de estos parámetros están especificados en el archivo priority.tcl

Este procedimiento debe ser llamado cuando son creadas las colas.

-d-tail.cc / d-tail.h

Se encuentran los métodos que se encargan de asignar a cada una de las colas los parámetros fijados en priority.tcl

- ns-mobilenode_802_11e.tcl

Este fichero es modificado para contemplar la posibilidad de que el tipo de mac sea 802.11e y que el tipo de cola sea DTail. Los cambios son mínimos. Se excluye ns-mobilenode.tcl.

5.3.2. Adaptación de los scripts a la simulación en 802.11e

En los scripts escritos anteriormente no ser tenia en cuenta la prioridad entre distintos tráficos ya que estaban escritos para usare en el protocolo 802.11.
Para añadir prioridad a cada uno de los tráficos añadiremos la prioridad después de la declaración del agente. Por ejemplo para añadir prioridad 0 al agente udp:

```
set udp_(1) [new Agent/UDP]
$udp_(1) set prio_ 0
```

En el simulador 0 es la máxima prioridad y 3 la mínima.

Al inicio del script, al declarar las variables se debe cambiar la mac y el tipo de cola utilizada.

```
set val(ifq)        Queue/DTail/PriQ        ;# Tipo de cola
set val(mac)        Mac/802_11e             ;#Tipo de MAC
```

También podemos cambiar los parámetros (CW_MIN, CW_MAX, AIFS, TXOPLimit) en función del Standard que queramos usar. Los valores de estos parámetros se encuentran en ns-x/mac/802_11e/priority.tcl.

En al **Tabla 3.1** aparecen los valores para cada uno de los estándares y para cada una de las prioridades.

Hay que recordar que cada vez que modifiquemos dichos parámetros en el archivo priority.tcl, debemos ejecutar el comando ./make en el directorio de NS.

*En los scripts usados en las simulaciones, los tráficos contienen prioridad tal y como he descrito. Los podemos encontrar en el anexo [2].

5.3.3. Adaptación del parche 802.11e a la versión ns-2.28

Para adaptar el simulador ns-2.28 al protocolo 802.11e, los pasos a seguir fueron los siguientes:

Entre los ficheros añadidos en ns-x.y/mac/802.11e, hay que diferenciar aquellos que son modificados y que ya existían en la versión original del simulador y aquellos añadidos:

-Ficheros añadidos:

-priq.cc
-priq.h
-priority.tcl
-d-tail.cc
-d-tail.h

Estos no hará falta que sean modificados ya que en la versión original de NS, no se hace uso de ellos y por lo tanto todo su código esta relacionado con la adaptación que se realiza al resto de ficheros.

-Ficheros modificados sobre la versión normal de NS:

-mac-802_11e.cc
-mac-802_11e.h
-mac-timers_802_11e.cc
-mac-timers_802_11e.h
-ns-mobilenode_802_11e.tcl

5.3.4. Diferencias entre el código de ns-2.26 y ns-2.26e

Para adaptar, lo que realizó el grupo TKN fue modificar algunos de los ficheros originales en el NS, los que implican cambios en el acceso al medio.

Una vez detectados que ficheros son, se compara el código del fichero sin adaptar con el nuevo fichero adaptado. El objetivo es realizar las mismas modificaciones en los mismos ficheros de la versión ns-2.28. Los ficheros son los siguientes:

Fichero original	Fichero modificado
mac-802_11.cc	mac-802_11e.cc
mac-timers.cc	mac-timers_802_11e.cc
ns-mobilenode_802_11.tcl	ns-mobilenode_802_11e.tcl

Para realizar estas comparaciones, fue necesario el uso de una herramienta que facilitara la comparación de dos archivos de texto. Por ello instale UltraCompare Professional.

La adaptación del código no implica la eliminación de los ficheros originales ya que ellos siguen siendo validos si deseamos realizar una simulación sin las características de tráficos con prioridad de 802.11e. En el caso de ns-mobilenode_802_11e.tcl si que este sustituye al original, posteriormente se comenta el motivo.

Las principales modificaciones realizadas entre estos ficheros son las siguientes:

mac-timers_802_11e.cc

-Se modifica el nombre de las clases añadiendo _802_11e.

-Se añaden dos clases nuevas, class SIFSTimer_802_11e : public MacTimer_802_11e y class AkaroaTimer : public MacTimer_802_11e.

-Se modifican las clases class BackoffTimer_802_11e : public MacTimer_802_11e y class DeferTimer_802_11e : public MacTimer_802_11e. El código añadido sirve para coordinar los tráficos en función de la prioridad, por ejemplo en BackoffTimer_802_11e se añade:

```
for (int pri = 0; pri < MAX_PRI; pri++){
                        AIFSwait_[pri] = 0.0;
                        offset_[pri] = 0;
                        stime_[pri] = rtime_[pri] = 0;
                        backoff_[pri] = 0;
```

donde podemos observar como se tratan las distintas variables según su prioridad.

mac-802_11e.cc

-Se crea una nueva clase llamada EDCA_PHY_ MIB, esta clase es igual que la anterior PHY_MIB pero cambiando el nombre de sus variables. Como ejemplo la variable DSSS_SlotTime pasa a llamarse DSSS_EDCA_SlotTime.

-Desaparece la declaración de la clase MAC_MIB ya que el código de esta clase no necesita ser modificado. El puntero de esta función se crea a partir de

la clase MAC_MIB, en cambio el puntero de phymib es de la clase EDCA_PHY_MIB.
Creación de los punteros:

```
EDCA_PHY_MIB    *phymib_;
MAC_MIB         *macmib_;
```

-En la clase Mac802_11e: public Mac, se gestionan las variables del tráfico según su prioridad.

-La variable AIFS es gestionada en este fichero.

ns-mobilenode_802_11.tcl

Las modificaciones realizadas en este fichero son prácticamente inexistentes. Contempla la posibilidad de que el tipo de cola sea DTail para dar prioridad:

```
if {$qtype == "Queue/DTail/PriQ" } {
        priority ifq
}
```

Y la posibilidad de que estemos usando el protocolo 802.11e, como se puede observar no hay diferencias si usa uno u otro protocolo:

```
if {$mactype == "Mac/802_11"} {
        $mac nodes [$god_ num_nodes]
}
if {$mactype == "Mac/802_11e"} {
      $mac nodes [$god_ num_nodes]
}
```

El resto de fichero es exactamente igual, por este motivo queda reemplazado por el fichero sin modificar ns-mobilenode_802_11.tcl.

5.3.5. Diferencias entre el código de ns-2.26 y ns-2.28

A parte del código añadido en la versión ns-2.28, la diferencia mas significativa es el modo de declarar las clases PHY_MIB y MAC_MIB. En consecuencia el modo de consultar las variables definidas en estas clases varía en función de la versión ns-2.26 o ns-2.28. Por ejemplo si queremos consultar el valor de SlotTime lo consultaremos de la siguiente manera:

ns-2.26:

```
mac->phymib_->SlotTime;
```

ns-2.28:

```
mac->phymib_.getSlotTime();
```

5.3.6. Problemas encontrados durante la adaptación

Una vez realizados en los ficheros de ns-2.28 los mismos cambios que los que se realizaron en la versión ns-2.26 y teniendo en cuenta las posibles modificaciones en el código entre ns-2.26 y ns-2.28, se procedió a ejecutar de nuevo ./configure, y make en el directorio de NS.

Lógicamente aparecieron errores y warining como variables no declaradas, falta de punto y coma en algún lugar del código, clases mal declaradas, etc. Solventar estos errores fue cuestión de tiempo y dedicación hasta que el programa aparentemente quedó bien construido.

Posteriormente se realizo el primer script de prueba y al lanzar la simulación apareció el error Violación de Segmento. Dicho error puede aparecer por multitud de motivos entre todo el código. Ante la inminente fecha de entrega del proyecto y el poco tiempo que quedaba decidimos comparar mi código con el código realizado por mi tutor para así poderlo depurar, realizar mis simulaciones y comprobar la eficacia del protocolo 802.11e.

5.3.7. Diferencias entre mi código y el código definitivo.

Ahora, se plantea comparar el código adaptado por mí con el código adaptado por mi tutor y así encontrar las diferencias entre ambos.

De nuevo se vuelve a usar la herramienta UltraCompare Professional para encontrar las diferencias.

Las diferencias en los ficheros fueron las siguientes:

-mac-timers_802_11e.cc / h

En este fichero no existe ninguna diferencia entre mi código y el código definitivo.

-ns-mobilenode_802_11e.tcl

En este fichero no existen diferencias entre el código definitivo y el de la versión ns-2.26.

En mi fichero, agregué algunas modificaciones que aparecían en la versión ns-2.28 como los siguientes comandos:

```
# let the routing agent know about the port dmux
        $agent port-dmux $dmux_

$ns mac-type $mactype

# change wrt Mike's code
      if { [Simulator set EotTrace_] == "ON" } {
```

```
if {$imepflag != ""} {
        set eotT [$self mobility-trace EOT "MAC"]
} else {
        set eoT [cmu-trace EOT "MAC" $self]
}
$mac eot-target $eotT
```

-mac-802_11e.cc

Las diferencias básicas en este fichero es el modo de declarar las clases PHY_MIB y MAC_MIB y sus variables públicas y privadas.

Al declarar estas clases, a parte de modificar el nombre de la clase añadiendo EDCA, también modificaba las variables internas como por ejemplo: EDCA_SlotTime y EDCA_SIFS.

A parte del modo del de declarar las clases PHY_MIB y MAC_MIB existe alguna diferencia más en el código como la siguiente:

Código en mi fichero:

```
if ((u_int32_t) ch->size() < macmib_.getRTSThreshold() ||
                        (u_int32_t) ETHER_ADDR(mh->dh_ra) ==
MAC_BROADCAST) {

if((u_int32_t) ETHER_ADDR(mh->dh_ra) == MAC_BROADCAST)

rTime = (Random::random() % (getCW(pri) + 1)) * phymib_.getSlotTime();

else rTime = 0;

mhDefer_.start(pri, getAIFS(pri) + rTime);

                } else {

                    mhSifs_.start(pri, phymib_.getSIFS());

                }

            }
```

Versión definitiva:

```
if ((u_int32_t) ch->size() < macmib_.getRTSThreshold()||
(u_int32_t) ETHER_ADDR(mh->dh_ra) ==
MAC_BROADCAST) {

rTime = (Random::random() % (getCW(pri) + 1)) * phymib_.getSlotTime();

mhDefer_.start(pri, getAIFS(pri) + rTime);

        } else {

                mhSifs_.start(pri, phymib_.getSIFS());

                }

        }
```

Las diferencias entre los dos códigos aparecen en negrita.

La sentencia

rTime = (Random::random() % (getCW(pri) + 1)) * phymib_.getSlotTime();

genera un tiempo random a partir del valor de Slottime y teniendo en cuenta el tamaño de la ventana de contención (CW).

En la versión definitiva se debe cumplir una de las dos condiciones para acceder a esta sentencia:

if ((u_int32_t) ch->size() < macmib_.getRTSThreshold() ||
(u_int32_t) ETHER_ADDR(mh->dh_ra) ==
MAC_BROADCAST) {

- El tamaño de los paquetes retransmitidos a de ser menor que el valor RTSThreshold. Se cumple mediante la sentencia:

if ((u_int32_t) ch->size() < macmib_.getRTSThreshold()

ch es un puntero a la estructura hdr_cmn
struct hdr_cmn *ch;

- El paquete a retransmitir debe ser a Broadcast

if((u_int32_t) ETHER_ADDR(mh->dh_ra) == MAC_BROADCAST)

mh es un puntero a la estructura hdr_mac802_11

struct hdr_mac802_11 *mh;

En el otro fichero solo se ejecuta la sentencia

rTime = (Random::random() % (getCW(pri) + 1)) * phymib_.getSlotTime();

cuando

if((u_int32_t) ETHER_ADDR(mh->dh_ra) == MAC_BROADCAST)

de este modo el valor rTime será distinto en función del código que usemos.

6. SIMULACIONES Y COMPROBACIÓN DEL PROTOCOLO 802.11E

Para validar las simulaciones que se realizarán posteriormente se elige simular un escenario igual al realizado en el documento 'Control de Admisión en IEEE 802.11e EDCA' citado en la bibliografía [7], y comparar los resultados. Este documento ha sido elegido ya que usa un simulador distinto a ns-2.28 desarrollado con la herramienta OPNET, de esta manera se garantiza que los resultados obtenidos con ns-2.28, son similares a los realizados con OPNET.

El escenario que se simula sirve además para comprobar la eficacia del protocolo 802.11e.

En las siguientes simulaciones, se usa siempre el estándar 802.11b.

6.1. Escenario 1

Se trata de cuatro estaciones generando tráfico de audio, dos de video y cinco de datos. En total 11 estaciones enviando tráfico hacia el AP.

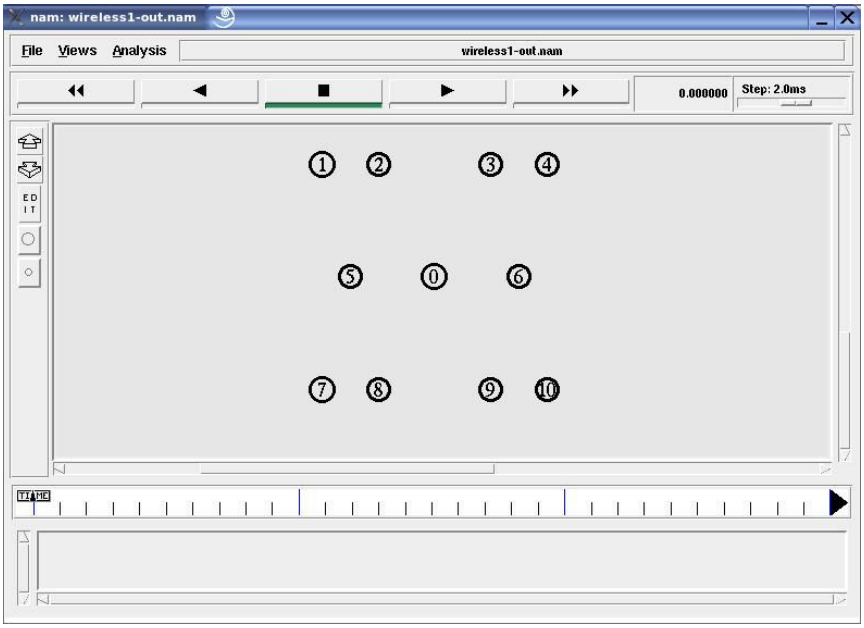

Fig. 6.1. Escenario 1

Tabla 6.1. Datos del tráfico

	Prioridad	Tasa de generación (kbps)	Tamaño del paquete (Bytes)	Numero de nodos transmitiendo
Audio	0	38	80	4
Video	1	1400	1500	2
Datos	2	800	1500	5

El tráfico total generado será de:

-Audio:

38 kbps/estación x 4 estaciones = 152 kbps

-Video:

1400 kbps/estación x 2 estaciones = 2800 kbps

-Datos:

800 kbps/estación x 5 estaciones = 4000 kbps

Se realizar la simulación de este escenario primero usando el simulador con el protocolo 802.11 para poder comprobar que throughput obtiene cada uno de los tráficos. Posteriormente se realizará la misma simulación haciendo uso de 802.11e y se observará que diferencias se producen.

Este escenario, incluido en el Anexo [2] ha sido implementado en el script Escenario_1. tcl.

Para lanzar la simulación ejecutamos:

./ns Escenari_1.tcl 4 2 5 0

Los resultados son los siguientes:

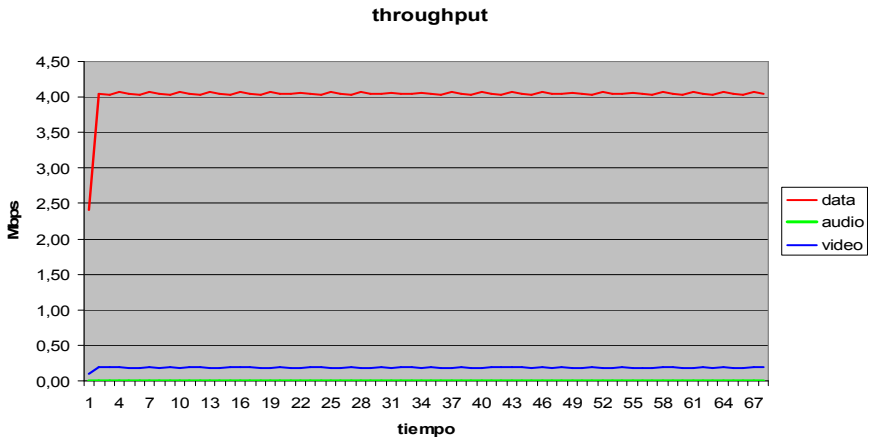

Fig.6.2. Throughput obtenido para cada uno de los tràficos:

Valor medio:

- Audio: 0 kbps
-Video: 178 kbps
-Datos: 4,02 Mbps

Podemos observar como el throughput obtenido para el audio y el video es insignificante. Frente a los 152 kbps de audio y los 2,80 Mbps de video requeridos solamente hemos obtenido 178 kbps para el video y 0 kbps para el audio. El tráfico de datos consigue ser transmitido en su totalidad. En estas condiciones se hace imposible proporcionar QoS en aplicaciones en tiempo real (como vídeo llamadas, difusión de video, etc.) ya que estas hacen uso de tráfico de tipo audio y video.

Vamos a realizar la simulación del mismo escenario pero esta vez haciendo uso del protocolo 802.11e.

Los resultados obtenidos son los siguientes:

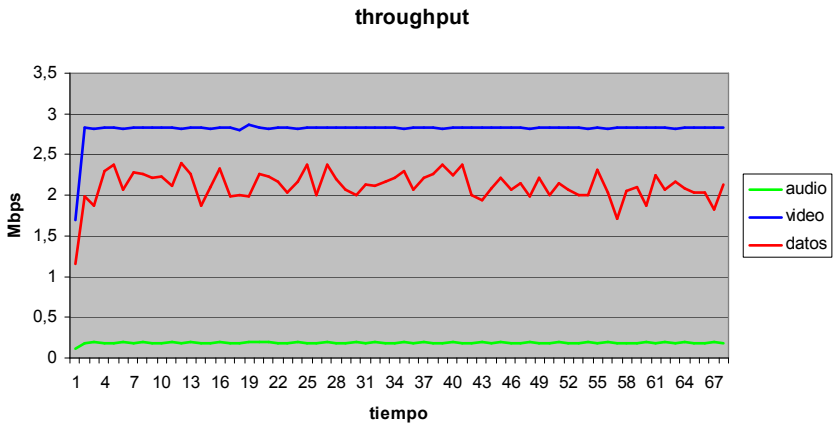

Fig.6.3. Throughput obtenido para cada uno de los tràficos:

Valor medio:

- Audio: 188,85 kbps
-Video: 2,820 Mbps
-Datos: 2,15 Mbps

El tráfico recibido es mayor que el tráfico generado ya que se contabilizan el total de bytes recibidos incluidos los bytes de cabecea de los paquetes.

En los resultados de esta simulación, se puede observar como el tráfico con menos prioridad, el de datos, es el más perjudicado, con una tasa de generación de 4 Mbps solo consigue transmitir 2,2 Mbps. Las aplicaciones con mayor prioridad, Audio y Video, consiguen transmitir todo el tráfico generado.

Ahora si que podemos asegurar calidad de servicio a aplicaciones en tiempo real ya que todo el tráfico generado en audio y video es transmitido correctamente. Pero para proporcionar QoS hace falta también asegurar que el retardo de los paquetes no es excesivo.

El retardo se calcula a partir del fichero de trazas generado por el simulador, al ejecutar $ns trace-all $tracefd en nuestro script. Como se comenta en el apartado **5.1.2.**, al configurar el nodo debe estar activado el comando -macTrace ON \. Ahora en el fichero de trazas se guarda información sobre en que instante ha sido enviado el paquete y en que instante ha sido recibido. Posteriormente mediante Excel se procesa este fichero y se obtienen los siguientes resultados:

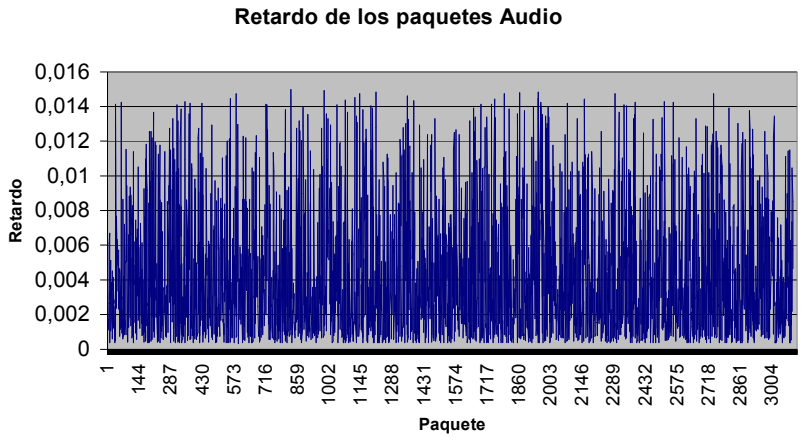

Fig. 6.4. Retardo para el tráfico de audio

Retardo medio de los paquetes de Audio: 2,2336 ms.

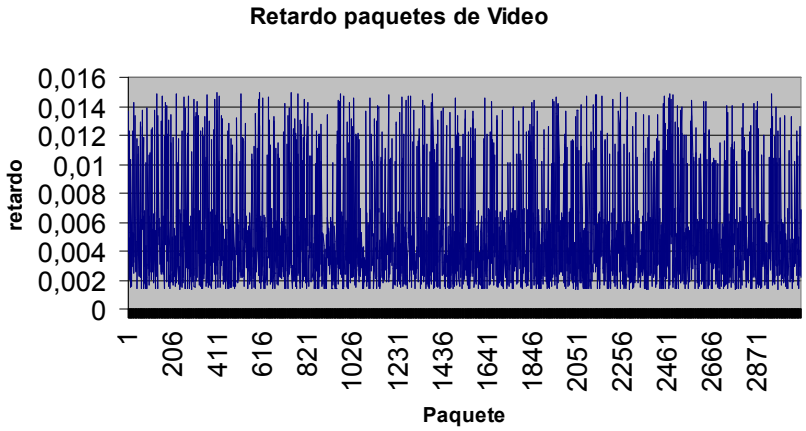

Fig. 6.5. Retardo para el tráfico de video

Retardo medio de los paquetes de video: 2,7456058 ms

El retardo mínimo en aplicaciones de audio y video, es de 100 ms, por lo tanto no vamos a tener ningún problema a la hora de realizar video llamadas o cualquier aplicación en tiempo real.

Tabla 6.2. Throughput generado y recibido según el estándar.

	Throughput audio kbps	Throughput video kbps	Throughput datos kbps
Tráfico generado	152	2800	4000
802.11	0	178	4020
802.11e	188,85	2,820	2,15

Vamos ahora a comparar los resultados obtenidos con ns-2.28 con los resultados obtenidos con OPNET.

- Los resultados obtenidos con OPNET son los siguientes:

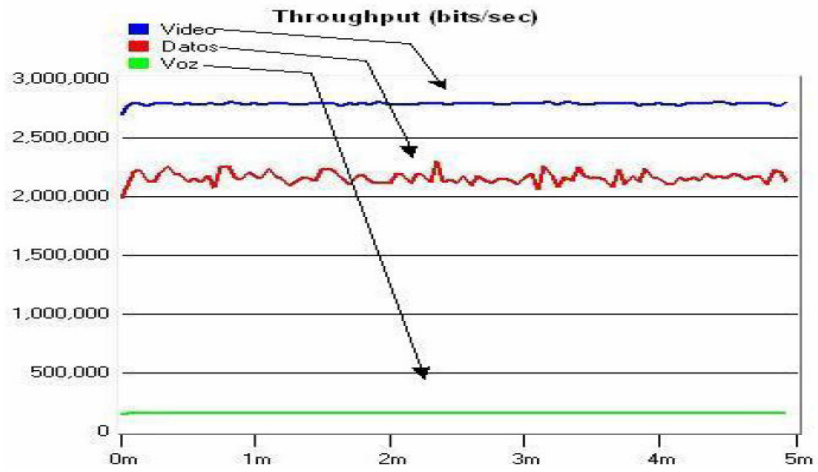

Fig. 6.6 Throughput según la prioridad

Valor medio:

- Audio: 150 kbps
-Video: 2,8 Mbps
-Datos: 2,2 Mbps

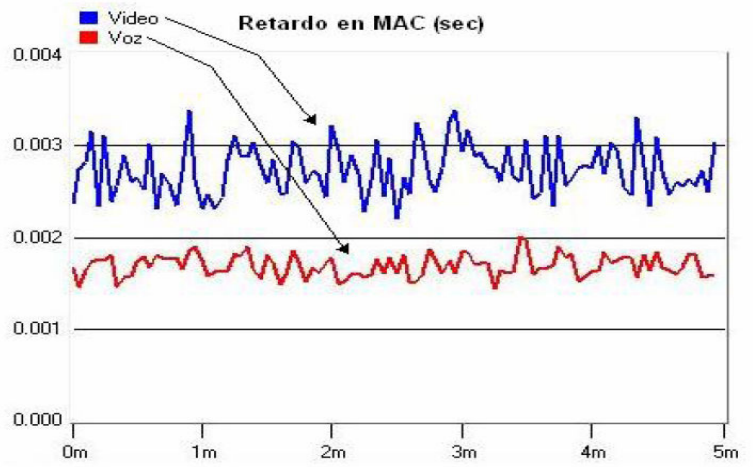

Fig. 6.7 Retardo según la prioridad

Retardo medio:

-Video: 2,5 ms

-Audio: 1,9 ms

Tabla 6.3. Resultados según el simulador.

	Throughput audio kbps	Throughput video kbps	Throughput datos kbps	Retardo audio ms	Retardo video ms
Tráfico generado	152	2800	4000	-	-
OPNET	150	2800	2200	1,9 ms	2,5 ms
ns-2.28	188,85	2820	2150	2,234	2,745

Los resultados obtenidos con ns-2.28 y OPNET son prácticamente los mismos. La pequeña variación del throughput es debido a que en ns-2.28 se cuantifica todos los bytes recibidos, incluidos los de las cabeceras. La variación del retardo es insignificante.

6.2. Escenario 2

En este escenario se añade una fuente de video al escenario 1, generando tráfico a 2 Mbps, con esto, se consigue simular una situación de saturación y se puede observar con mayor claridad como actúa el mecanismo EDCA a la hora de priorizar el tráfico en función de su prioridad.

El tráfico total generado será de:

-Audio:

38 kbps/estación x 4 estaciones = 152 kbps.

-Video:

1400 kbps/estación x 2 estaciones + 2000 kbps/estación x 1 estación = 4800 kbps.

-Datos:

800 kbps/estación x 5 estaciones = 4000 kbps.

Para lanzar la simulación ejecutamos:

./ns Escenario_2.tcl 4 3 5 0

Este script lo podemos consultar en el Anexo [2].

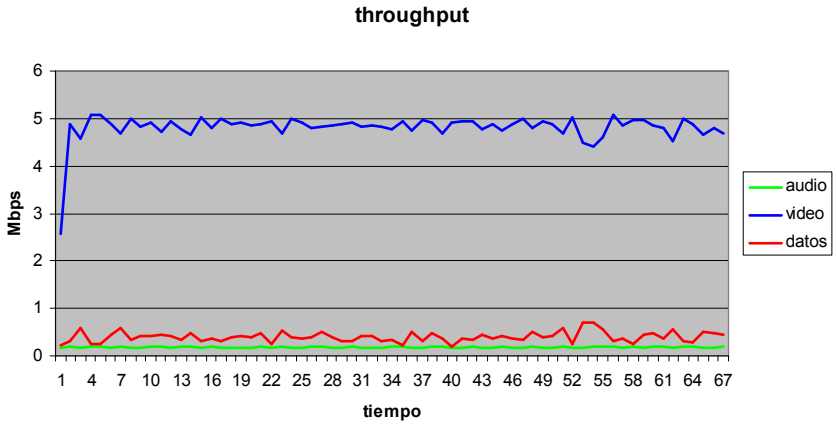

Fig. 6.8 Throughput obtenido para cada uno de los tráficos.

Valor medio:

- Audio: 188 kbps
-Video: 4,834 Mbps
-Datos: 482 kbps.

Retardo obtenido para cada uno de los tráficos:

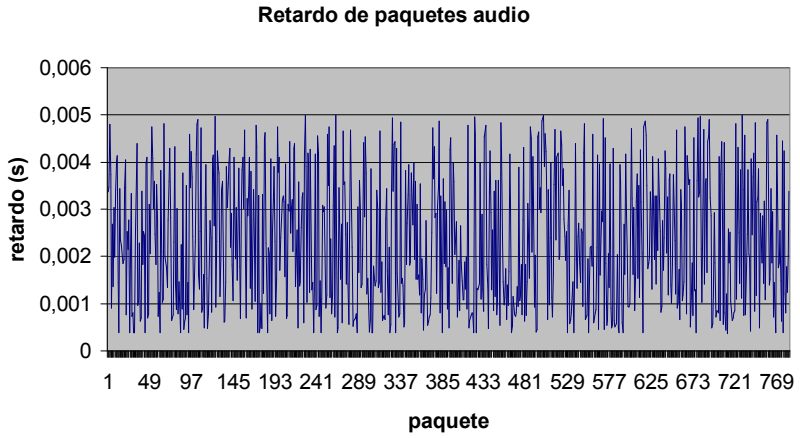

Fig. 6.9 Retardo para el tráfico de audio

Retardo medio de los paquetes de Audio: 2,1 ms.

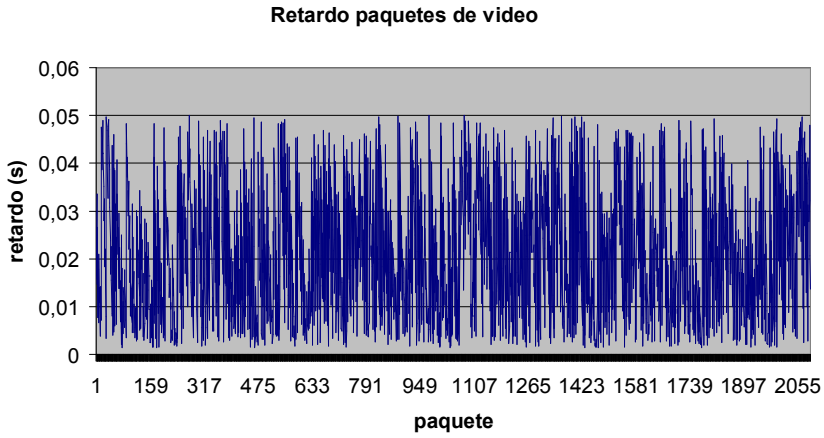

Fig. 6.10 Retardo para el tráfico de video

Retardo medio de los paquetes de Video: 29,52 ms.

-Los resultados obtenidos con OPNET son los siguientes:

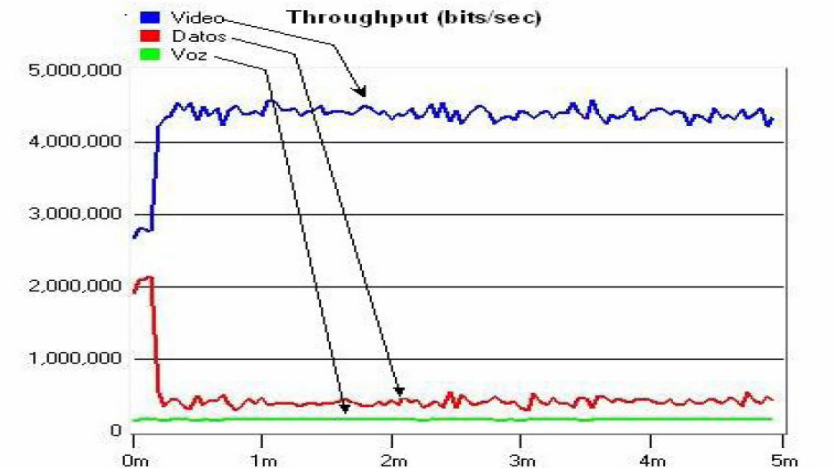

Fig. 6.11 Throughput según la prioridad

Throughput:

-Audio: 152 kbps
-Video: 4,6 Mbps
-Datos: 450 kbps

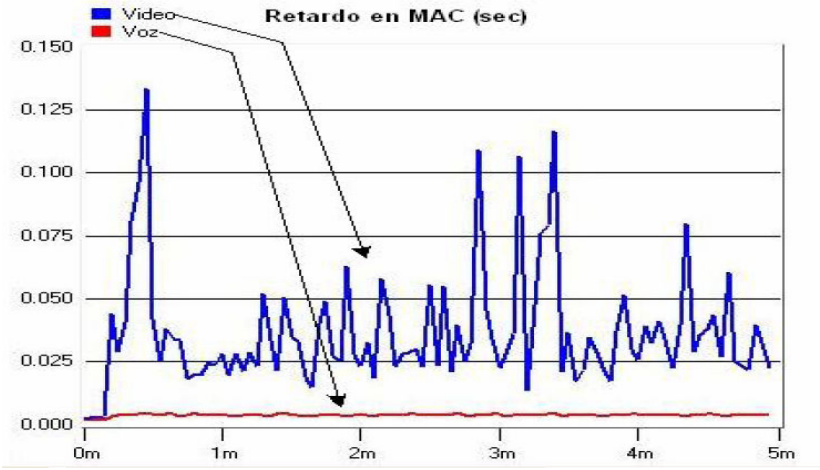

Fig. 6.12 Retardo según la prioridad

Retardo:

-Audio: 1,9 ms
-Video: 27 ms

Tabla 6.4. Resultados según el simulador.

	Throughput audio kbps	Throughput video kbps	Throughput datos kbps	Retardo audio ms	Retardo video ms
Tráfico generado	152	4800	4000	-	-
OPNET	152	4600	450	1,9	27
ns-2.28	188	4834	482	2,1	29,51

En esta situación de congestión, se puede observar mejor la utilidad del protocolo 802.11e. El tráfico de datos, que es el de menor prioridad, resulta mayormente perjudicado. En este caso, generando la misma tasa que en el escenario 1 (4 Mbps) solo consigue transmitir 482 kbps frente los 2,11 Mbps que transmitía en el escenario 1. Ello es debido a que el tráfico generado en video (de mayor prioridad que el de datos) aumenta 2 Mbps, y este, es transmitido en toda su totalidad. Por lo tanto a cambio de los 2Mbps añadidos al video, es reducida la tasa del tráfico de datos en 2 Mbps.

Por último el tráfico de audio (el de máxima prioridad) consigue transmitir los 152 kbps generados.

Los retardos aumentan tanto en el tráfico de audio como en el de video.

6.3. Escenario 3

En el escenario 3 se trata de simular una red de tres estaciones, en 2 de ellas se transmite a una tasa elevada de datos (supongamos que están haciendo uso de un programa P2P). En la tercera estación, se está realizando difusión de audio y video, está aplicación requiere garantías de QoS, vamos a ver que sucede si usamos el protocolo 802.11 y el protocolo 802.11e.

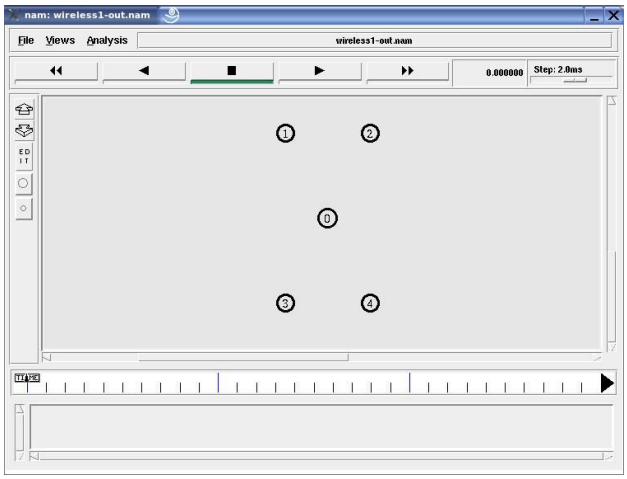

Fig. 6.13. Escenario 2

Tabla 6.5. Datos del tráfico generado:

	Prioridad	Tasa de generación (kbps)	Tamaño del paquete (Bytes)	Numero de tráficos.
Audio	0	1000	80	1
Video	1	2000	1500	1
Datos	2	3000	1500	2

El tráfico total generado será de:

-Audio:
1000 kbps/estación x 1 fuente de tráfico = 1000 kbps

-Video:
2000 kbps/estación x 1 fuente de tráfico = 2000 kbps

-Datos:
3000 kbps/estación x 2 fuentes de tráficos = 6000 kbps

Para lanzar la simulación ejecutamos:

./ns Escenario_3.tcl

Este script lo podemos consultar en el Anexo [2].

Los resultados obtenidos con 802.11 son los siguientes:

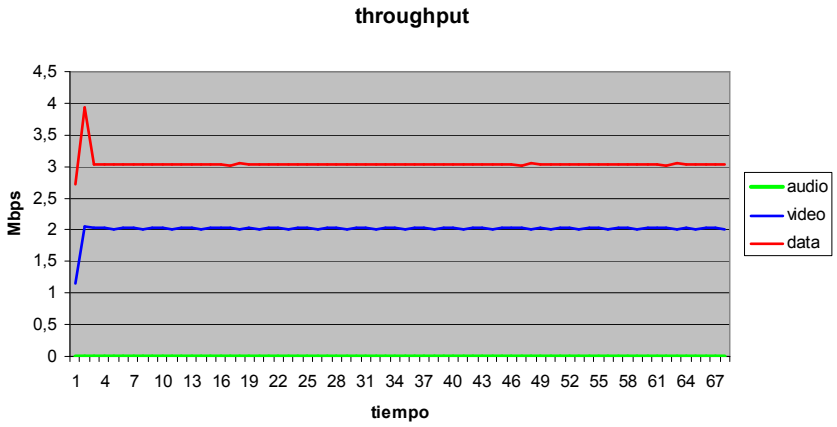

Fig. 6.14 Throughput según la prioridad

Throughput:
-Audio: 0 kbps
-Video: 2,014 Mbps
-Datos: 3,04 Mbps

Los resultados obtenidos con 802.11e son los siguientes:

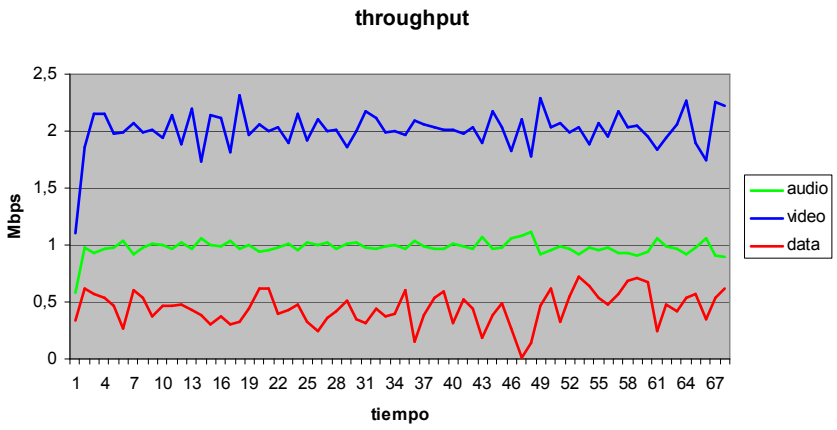

Fig. 6.15 Throughput según la prioridad

Throughput:
-Audio: 0,9799 Mbps
-Video: 2,014 Mbps
-Datos: 473 kbps

Tabla 6.6. Throughput generado y recibido según el estándar.

	Throughput audio kbps	Throughput video kbps	Throughput datos kbps
Tráfico generado	1000	2000	6000
802.11	0	2014	304
802.11e	979,9	2014	473

En este escenario podemos comprobar como la aplicación de difusión de audio y video carece de calidad de servicio si no usamos el estándar 802.11e. En cambio usando el acceso al medio mejorado EDCA, se consigue que el tráfico de audio y video pueda transmitir sin problemas a costa de reducir el throughput en el tráfico de datos, de este modo aseguramos QoS.

CONCLUSIONES

Como el título del proyecto indica, el objetivo de este proyecto es realizar la adaptación y test del protocolo 802.11e al simulador ns-2.28. Este simulador se ejecuta bajo el sistema operativo Linux.

Durante los primeros meses, el tiempo se dedicó a la selección de la versión de Linux más apropiada, el estudio de las redes 802.11, el protocolo 802.11e, el modo de implementar un script para simular un escenario en NS, y finalmente la instalación de Linux y NS-2.

El estudio de las redes 802.11 se baso en la lectura del tutorial: '802.11 Wireless Networks. The Definitive Guide'. La formación de como escribir los scripts que posteriormente sirven para implementar escenarios de simulación fue mediante el Tutorial Marc Greis.

Una vez logrado, fue básico también documentarse sobre el nuevo estándar 802.11e, y cuales son los cambios implementados sobre el mecanismo de acceso al medio respecto al estándar 802.11 para lograr proporcionar de calidad de servicio a las redes 802.11.

Hay que tener en cuenta que este estándar todavía no estaba aprobado por la IEEE en caunto empezé a realizar este proyecto. Por este motivo la documentación en que he basado mi aprendizaje (de la cual se hace referencia en la bibliografía) ha sido escrita por investigadores. Básicamente ha sido documentación del grupo de trabajo TKN.

Gran parte de este proyecto se ha basado en entender los estándares 802.11, 802.11e y la creación de scripts.

Para alcanzar el objetivo planteado se ha llevado a cabo un estudio de ficheros en OTcl con la finalidad de entender como implementar un escenario en 802.11, el cual contenga nodos, nodos móviles, estaciones base, enlaces entre nodos, y como diferenciar la prioridad de los tráficos según especifica 802.11e. Posteriormente se usarán estos ficheros para lanzar las simulaciones en NS.

El trabajo se ha basado en comprender las diferencias del código fuente del simulador entre la versión ns-2.26, escrito en lenguaje C++, sin adaptar y la versión ns-2.26 adaptada al 802.11e por el grupo de investigadores TKN. Posteriormente se han aplicado los mismos cambios al simulador ns-2.28. Esta parte ha consumido gran cantidad de tiempo ya que los ficheros son extensos e implementan muchas clases y funciones.

Una vez adaptado el simulador ns-2.28 para permitir la simulación de redes 802.11 en que se haga uso del estándar 802.11e, se han implementado scripts de prueba que simulan escenarios donde transmiten distintos nodos generando fuentes de tráfico de distinta prioridad (audio, video y datos de mayor o menor prioridad).

Los resultados de estas simulaciones permite comparar cual es el throughput logrado en cada uno de los tráficos en función de su prioridad usando el futuro estándar 802.11e y sin usarlo.

En condiciones en que el throughput generado por las fuentes no supere la capacidad de transmisión de la red, no se aprecian diferencias entre el uso o no de 802.11e. En cambio cuando la red empieza a estar congestionada, las fuentes de tráfico requeridas de calidad de servicio mantienen el throughput generado siempre i cuando no se supere la capacidad de la red, mientras que las fuentes de datos no consiguen transmitir todo el throughput generado. De este modo se comprueba la eficacia de este estándar para proporcionar QoS a las aplicaciones en tiempo real.

Por falta de tiempo no se han podido realizar simulaciones donde modificar los parámetros (AIFSN[AC], CWmin, CWmax y TXOP) especificados en 802.11e. Hubiera sido interesante realizar estas pruebas ya que de este modo se podría comprobar como influye el valor de cada uno de estos parámetros a la hora de ajustar la prioridad que obtiene un tráfico respecto al resto.

Durante el desarrollo de este proyecto hemos hablado de redes inalámbricas. Como ya es sabido, la ausencia de cables en las comunicaciones implica el uso de ondas electromagnéticas contribuyendo de este modo a la denominada contaminación electromagnética.

En los últimos años se ha producido un espectacular aumento de la contaminación electromagnética, originado por antenas emisoras de telefonía, radio y televisión, radares, aparatos eléctricos, teléfonos móviles, teléfonos inalámbricos, etc. La tecnología WLAN es un añadido a esta lista.

Según algunos estudios los efectos que la contaminación electromagnética puede provocar sobre el cuerpo humano son las siguientes: Insomnio, estrés, angustia, ansiedad, pérdida de memoria, patologías cardiovasculares entre otras.

Así pues la tecnología WLAN contribuye a la contaminación del medioambiente de manera no visible directamente ya que no somos capaces de percibir las ondas electromagnéticas usadas para transmitir en sistemas WLAN, pero si lo hace provocando efectos sobre seres vivos.

BIBLIOGRAFIA

1.1 Documentación:

[0] Web de descarga NS. http://www.isi.edu/nsnam/ns/

[1] Ns Manual. http://www.isi.edu/nsnam/ns/ns-documentation.html

[2] Ns for beginners. de Altman and Jiménez. http://www.isi.edu/nsnam/ns/

[3] Marc Greis`s Tutorial. http://www.isi.edu/nsnam/ns/tutorial/index.html

[4] 802.11 Wireless Networks. The Definitive Guide.
Matthew Gast. (O'Reilly)

1.2 Referencias:

[5] Grupo de trabajo TKN: http://www.tkn.tu-berlin.de/research/802.11e_ns2/

[6] Estándares oficiales 802.11. Descargados de la Web http://www.ieee802.org/11/

IEEE Std 802.11a-1999(Supplement to IEEE Std 802.11-1999). Tabla 93

IEEE Std 802.11b-1999. IEEE Std 802.11b-1999 (R2003). Tabla101

IEEE Std 802.11g™-2003. Tabla 123G

IEEE Std 802.11e – 2005.

[7] Fuente de las pruebas de simulación: Control de Admisión en IEEE 802.11e EDCA.
Autores: Jakub Majkowski, Ferran Casadevall, Ferran Adelantado Freixer

[8] Análisis of IEEE 802.11e and Appliation of Game Models for Support of Quality-of-Service in Coexisting Wireless Networks.
Autor: Stefan Mangold.

Anexo [1]. PRIMEROS SCRIPT TCL

En este anexo se adjunta el código de los primeros scripts usados.

1.1 Dos nodos enlazados mediante una línea duplex.

```
#
# ======================================================================
# Define options
#
# ======================================================================

set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)        Mac/802_11
set val(ifq)        Queue/DropTail/PriQueue
set val(ll)         LL
set val(ant)        Antenna/OmniAntenna
set val(x)            670   ;# X dimension of the topography
set val(y)            670   ;# Y dimension of the topography
set val(ifqlen)        50           ;# max packet in ifq
set val(seed)          0.0
set val(adhocRouting)   DSR
set val(nn)            3            ;# how many nodes are simulated
set val(cp)           "../mobility/scene/cbr-3-test"
set val(sc)           "../mobility/scene/scen-3-test"
set val(stop)          400.0          ;# simulation time

#
# ======================================================================
# Main Program
#
# ======================================================================

#
# Initialize Global Variables
#

# create simulator instance

set ns_                [new Simulator]

# setup topography object
```

```
set topo       [new Topography]

# create trace object for ns and nam

set tracefd    [open wireless1-out.tr w]
set namtrace   [open wireless1-out.nam w]

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
#
set god_ [create-god $val(nn)]

#
# define how node should be created
#

#global node setting

$ns_ node-config -adhocRouting $val(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan) \
                     -topoInstance $topo \
                     -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF

#
#  Create the specified number of nodes [$val(nn)] and "attach" them
#  to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0              ;# disable random motion
}


#
```

```
# Define node movement model
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

    # 20 defines the node size in nam, must adjust it according to your scenario
    # The function must be called after mobility model is defined

    $ns_ initial_node_pos $node_($i) 20
}


#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
}

$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x $val(x) y $val(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"

puts "Starting Simulation..."
$ns_ run
```

1.2 Red de nodos wireless en sistema jerárquico

```
#===================================================================
# Define options
#
#===================================================================

set opt(chan)   Channel/WirelessChannel        ;# channel type
set opt(prop)   Propagation/TwoRayGround       ;# radio-propagation model
set opt(netif)  Phy/WirelessPhy                ;# network interface type
set opt(mac)    Mac/802_11                     ;# MAC type
set opt(ifq)    Queue/DropTail/PriQueue        ;# interface queue type
set opt(ll)     LL                             ;# link layer type
set opt(ant)    Antenna/OmniAntenna            ;# antenna model
set opt(ifqlen)         50                     ;# max packet in ifq
set opt(nn)             1                      ;# number of mobilenodes
set opt(adhocRouting)   DSDV                   ;# routing protocol

set opt(cp)     ""                             ;# cp file not used
set opt(sc)     ""                             ;# node movement file.

set opt(x)      670                            ;# x coordinate of topology
set opt(y)      670                            ;# y coordinate of topology
set opt(seed)   0.0                            ;# random seed
set opt(stop)   250                            ;# time to stop simulation

set opt(ftp1-start)      100.0

set num_wired_nodes      2
#set num_bs_nodes       2  ; this is not really used here.

#===================================================================

# check for boundary parameters and random seed
if { $opt(x) == 0 || $opt(y) == 0 } {
        puts "No X-Y boundary values given for wireless topology\n"
}
if {$opt(seed) > 0} {
        puts "Seeding Random number generator with $opt(seed)\n"
        ns-random $opt(seed)
}

# create simulator instance
set ns_   [new Simulator]
```

```
# set up for hierarchical routing
$ns_ node-config -addressType hierarchical

AddrParams set domain_num_ 3          ;# number of domains
lappend cluster_num 2 1 1             ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 1 1 2 1           ;# number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel ;# of each domain

set tracefd  [open wireless3-out.tr w]
set namtrace [open wireless3-out.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# Create topography object
set topo   [new Topography]

# define topology
$topo load_flatgrid $opt(x) $opt(y)

# create God
#   2 for HA and FA
create-god [expr $opt(nn) + 2]

#create wired nodes
set temp {0.0.0 0.1.0}          ;# hierarchical addresses
for {set i 0} {$i < $num_wired_nodes} {incr i} {
   set W($i) [$ns_ node [lindex $temp $i]]
}

# Configure for ForeignAgent and HomeAgent nodes
$ns_ node-config -mobileIP ON \
                 -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channelType $opt(chan) \
                     -topoInstance $topo \
                 -wiredRouting ON \
                     -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF

# Create HA and FA
set HA [$ns_ node 1.0.0]
```

```
set FA [$ns_ node 2.0.0]
$HA random-motion 0
$FA random-motion 0

# Position (fixed) for base-station nodes (HA & FA).
$HA set X_ 1.000000000000
$HA set Y_ 2.000000000000
$HA set Z_ 0.000000000000

$FA set X_ 650.000000000000
$FA set Y_ 600.000000000000
$FA set Z_ 0.000000000000

# create a mobilenode that would be moving between HA and FA.
# note address of MH indicates its in the same domain as HA.
$ns_ node-config -wiredRouting OFF

set MH [$ns_ node 1.0.1]
set node_(0) $MH
set HAaddress [AddrParams addr2id [$HA node-addr]]
[$MH set regagent_] set home_agent_ $HAaddress

# movement of the MH
$MH set Z_ 0.000000000000
$MH set Y_ 2.000000000000
$MH set X_ 2.000000000000

# MH starts to move towards FA
$ns_ at 100.000000000000 "$MH setdest 640.000000000000
610.000000000000 20.000000000000"
# goes back to HA
$ns_ at 200.000000000000 "$MH setdest 2.000000000000 2.000000000000
20.000000000000"

# create links between wired and BaseStation nodes
$ns_ duplex-link $W(0) $W(1) 5Mb 2ms DropTail
$ns_ duplex-link $W(1) $HA 5Mb 2ms DropTail
$ns_ duplex-link $W(1) $FA 5Mb 2ms DropTail

$ns_ duplex-link-op $W(0) $W(1) orient down
$ns_ duplex-link-op $W(1) $HA orient left-down
$ns_ duplex-link-op $W(1) $FA orient right-down

# setup TCP connections between a wired node and the MobileHost

set tcp1 [new Agent/TCP]
$tcp1 set class_ 2
set sink1 [new Agent/TCPSink]
$ns_ attach-agent $W(0) $tcp1
$ns_ attach-agent $MH $sink1
```

```
$ns_ connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns_ at $opt(ftp1-start) "$ftp1 start"

# source connection-pattern and node-movement scripts
if { $opt(cp) == "" } {
        puts "*** NOTE: no connection pattern specified."
       set opt(cp) "none"
} else {
        puts "Loading connection pattern..."
        source $opt(cp)
}
if { $opt(sc) == "" } {
        puts "*** NOTE: no scenario file specified."
       set opt(sc) "none"
} else {
        puts "Loading scenario file..."
        source $opt(sc)
        puts "Load complete..."
}

# Define initial node position in nam

for {set i 0} {$i < $opt(nn)} {incr i} {

   # 20 defines the node size in nam, must adjust it according to your
   # scenario
   # The function must be called after mobility model is defined

   $ns_ initial_node_pos $node_($i) 20
}

# Tell all nodes when the siulation ends
for {set i 0} {$i < $opt(nn) } {incr i} {
   $ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).0 "$HA reset";
$ns_ at $opt(stop).0 "$FA reset";

$ns_ at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
proc stop {} {
   global ns_ tracefd namtrace
   close $tracefd
   close $namtrace
}

# some useful headers for tracefile
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp \
```

```
        $opt(adhocRouting)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"

puts "Starting Simulation..."
$ns_ run
```

Anexo [2]. SCRIPTS DE SIMULACIONES

Scripts usados para crear los escenarios de las simulaciones en el capítulo 6.

2.1 Escenario_1.tcl

```
#
# ======================================================================
# Parametros de la linea de comandos
#
# ======================================================================
global argv arg0

set naudio  [lindex $argv 0];#Numero d'estacions que trx audio
set nvideo  [lindex $argv 1];# Numero d'estacions que trx video
set ndata1  [lindex $argv 2];# Numero d'estacions que trx data1
set ndata2  [lindex $argv 3];# Numero d'estacions que trx data2
set ran     [lindex $argv 4]; # Seed para el generador de numeros
aleatorios

#
# ======================================================================
# Define options
#
# ======================================================================
set val(chan)         Channel/WirelessChannel  ;# channel type
set val(prop)         Propagation/TwoRayGround ;# radio-propagation
model
set val(ant)          Antenna/OmniAntenna      ;# Antenna type
set val(ll)           LL                       ;# Link layer type
set val(ifq)          Queue/DTail/PriQ         ;# Interface queue type
set val(ifqlen)       50                       ;# max packet in ifq
set val(netif)        Phy/WirelessPhy          ;# network interface
type
set val(mac)          Mac/802_11e               ;# MAC type
set val(rp)           DSDV                     ;# ad-hoc routing
protocol
set val(nn)           [expr $nvideo + $naudio + $ndata1 + $ndata2 + 1]
;# number of mobilenodes

# including AP.
set val(seed)         $ran;              # Seed para el generador de

#
# ======================================================================
# Parametros de nodos 802.11
#
# ======================================================================
Mac/802_11 set delay_ 64us          ;#MAC Processing delay. En el b 0
(not applicable) en el g <2micros
Mac/802_11 set ifs_ 16us            ;# No especificaciones, no se
utiliza en el codigo
Mac/802_11 set slotTime_ 20us       ;# 20 en el b 20 o 9 en el g
Mac/802_11 set cwmin_ 31            ;# 31 en el b 31 o 15 en el g
Mac/802_11 set cwmax_ 1023          ;#1024 en ambos
Mac/802_11 set rtxLimit_ 16         ;#No especificaciones
Mac/802_11 set bssId_ -1
Mac/802_11 set sifs_ 10us           ;# 10 en ambos
Mac/802_11 set pifs_ 30us           ;#SIFS+SlotTime
```

```
Mac/802_11 set difs_ 50us          ;#SIFS+2SlotTime
Mac/802_11 set rtxAckLimit_ 1
Mac/802_11 set rtxRtsLimit_ 3
Mac/802_11 set basicRate_ 11Mb  ;# set this to 0 if want to use
bandwidth_ for
Mac/802_11 set dataRate_ 11Mb   ;# both control and data pkts


Mac/802_11 set CWMin_         31         ;#
Mac/802_11 set CWMax_         1023       ;#
Mac/802_11 set SlotTime_      0.000020        ;# 20us
Mac/802_11 set SIFS_          0.000010        ;# 10us
Mac/802_11 set PreambleLength_        144             ;# 144 bit
Mac/802_11 set PLCPHeaderLength_      48              ;# 48 bits
Mac/802_11 set PLCPDataRate_  1.0e6           ;# 1Mbps
Mac/802_11 set RTSThreshold_  0               ;# bytes
Mac/802_11 set ShortRetryLPLCPimit_       7               ;#
retransmittions
Mac/802_11 set LongRetryLimit_        4               ;#
retransmissions


#
# Parametros de nodos 802.11e Per 802.11b
#
Mac/802_11e set delay_ 64us ;#a MAC processing delay, antes ponia 64us
Mac/802_11e set ifs_ 16us  ;#antes 16us, pero no se utiliza en el
codigo
Mac/802_11e set slotTime_ 20us ;#antes 16us
Mac/802_11e set cwmin_ 31
Mac/802_11e set cwmax_ 1023
Mac/802_11e set rtxLimit_ 16
Mac/802_11e set bssId_ -1
Mac/802_11e set sifs_ 10us ;#8us
Mac/802_11e set pifs_ 30us ;#SIFS+SlotTime
Mac/802_11e set difs_ 50us ;#SIFS+2SlotTime
Mac/802_11e set rtxAckLimit_ 1
Mac/802_11e set rtxRtsLimit_ 3
Mac/802_11e set basicRate_ 11Mb  ;# set this to 0 if want to use
bandwidth_ for
Mac/802_11e set dataRate_ 11Mb   ;# both control and data pkts
Mac/802_11e set cfb_ 1 ; # 0 disables CFB

Mac/802_11e set SlotTime_      0.000020        ;# 20us
Mac/802_11e set SIFS_          0.000010        ;# 10us
Mac/802_11e set PreambleLength_        144             ;# antes 144
bit
Mac/802_11e set PLCPHeaderLength_      48              ;# antes 48
bits
Mac/802_11e set PLCPDataRate_ 1.0e6           ;# 1Mbps
Mac/802_11e set RTSThreshold_  0               ;# bytes
Mac/802_11e set ShortRetryLimit_       7               ;#
retransmittions
Mac/802_11e set LongRetryLimit_        4               ;#
retransmissions

ns-random $val(seed)

Agent/TCP set packetSize_ 1500
```

```
Agent/TCPSink/DelAck set interval_ 200ms

Phy/WirelessPhy set RXThresh_ 3.652e-10

set ns_    [new Simulator]
set tracefd     [open ns.tr w]
$ns_ trace-all $tracefd
set graphFiles ""
set graphLostFiles ""

set audio [open audio.tr w]
set video [open video.tr w]
set data1 [open data1.tr w]
set data2 [open data2.tr w]


set topo    [new Topography]
$topo load_flatgrid 500 500

create-god $val(nn);#Hay que poner el numero total de nodos moviles

set chan [new $val(chan)]
#configuracion del enrutamiento jerarquico

$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2           ;# number of domains

lappend cluster_num 1 1               ;# number of clusters in each
domain
AddrParams set cluster_num_ $cluster_num

lappend eilastlevel 1 [expr $val(nn)-1]                ;# number of
nodes in each cluster
AddrParams set nodes_num_ $eilastlevel ;# of each domain

#Creaci?n del nodo wired
set temp {0.0.0 1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8
1.0.9 1.0.10 1.0.11 1.0.12 1.0.13 1.0.14 1.0.15 1.0.16 1.0.17 1.0.18
1.0.19 1.0.20 1.0.21 1.0.22 1.0.23 1.0.24 1.0.25 1.0.26 1.0.27 1.0.28
1.0.29 1.0.30 1.0.31 1.0.32 1.0.33 1.0.34 1.0.35 1.0.36 .0.37 1.0.38
1.0.39 1.0.40 1.0.41 1.0.42 1.0.43 1.0.44}

# Configurar AP
$ns_ node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-channel $chan \
-wiredRouting ON \
-agentTrace ON \
-routerTrace OFF\
-macTrace ON \
-movementTrace OFF \
-eotTrace OFF

set AP(0) [$ns_ node [lindex $temp 0]] ;#direccion del AP
```

```
$AP(0) random-motion 0  ;#deshabilitar movimiento del AP


#Configuracion de su MAC
set macAP [$AP(0) getMac 0]
set direccionAP [$macAP id]
puts "Direccion AP0: $direccionAP"
$macAP bss_id $direccionAP
$macAP set Short11bPreamble_ true
$macAP set RTSThreshold_ 2500
#$macAP set FrameErrorRate_ 5.0
$macAP set basicRate_ 11Mb
$macAP set dataRate_ 11Mb

#
===========================================================================
# Prioridades del AP
#
===========================================================================

#Situacion del AP0
$AP(0) set X_ 0.0
$AP(0) set Y_ 0.0
$AP(0) set Z_ 0.0

#Creacion de los nodos moviles
$ns_ node-config -wiredRouting OFF

#creo els nodes
for {set i 1} {$i < $val(nn) } {incr i} {

      set movil($i) [$ns_ node [lindex $temp $i]]
      $movil($i) base-station [AddrParams addr2id [$AP(0) node-addr]]
      $movil($i) random-motion 0;                    #deshabilitar
movimiento

      #Configuracion de su MAC
      set mac [$movil($i) getMac 0]
      $mac bss_id $direccionAP
      $mac set proto_on 0

      $mac set Short11bPreamble_ true
      $mac set RTSThreshold_ 2500
      #$mac set FrameErrorRate_ 5.0
      #$mac set basicRate_ 11Mb                     ;#S'especifica
segun especificaciones
      #$mac set dataRate_  11Mb                     ;#$tas_($i)


      $movil($i) set X_ 70.0
      $movil($i) set Y_ 0.0
      $movil($i) set Z_ 0.0
}

#creo els trafics
set j 1
for {set i 1} {$i <= $naudio } {incr i} {

            #Extraccion de datos movil
                  set graphFd($j) [open "simple$j.gra" w]
                  puts $graphFd($j) "0 0"
```

```
                    set graphFiles "$graphFiles simple$j.gra"
                    set graphlost($j) [open "lost$j.gra" w]
                    puts $graphlost($j) "0 0"
                    set graphLostFiles "$graphLostFiles lost$j.gra"
                    set total($j) 0

                    set udp_($j) [new Agent/UDP]
                    set sink_($j) [new Agent/LossMonitor]
                    set cbr_($j) [new Application/Traffic/CBR]

                    $udp_($j) set packetSize_ 1500
                    $udp_($j) set fid_ 0
                    $udp_($j) set prio_ 0
                    $udp_($j) set class_ 0
                    $cbr_($j) set rate_ 38Kb
                    $cbr_($j) set packetSize_ 80

                    $ns_ attach-agent $movil(1) $udp_($j)

                    $ns_ attach-agent $AP(0) $sink_($j)
                    $ns_ connect $udp_($j) $sink_($j)
                    $cbr_($j) attach-agent $udp_($j)
                    set pru [expr 12.4+$j*48678/100000000.0];
                    $ns_ at $pru "$cbr_($j) start"

                    puts "Creat el trafic naudio[expr $j]"
                    set j [expr $j + 1]

                    #set totalTotal [expr $totalTotal + $total($i)]

}

for {set i 1} {$i <= $nvideo } {incr i} {

            #Extraccion de datos movil
                    set graphFd($j) [open "simple$j.gra" w]
                    puts $graphFd($j) "0 0"
                    set graphFiles "$graphFiles simple$j.gra"
                    set graphlost($j) [open "lost$j.gra" w]
                    puts $graphlost($j) "0 0"
                    set graphLostFiles "$graphLostFiles lost$j.gra"
                    set total($j) 0

                    set udp_($j) [new Agent/UDP]
                    set sink_($j) [new Agent/LossMonitor]
                    set cbr_($j) [new Application/Traffic/CBR]


                    $udp_($j) set packetSize_ 1500
                    $udp_($j) set fid_ 1
                    $udp_($j) set prio_ 1
                    $udp_($j) set class_ 1
                    $cbr_($j) set rate_ 1400Kb
                    $cbr_($j) set packetSize_ 1500

                    $ns_ attach-agent $movil(1) $udp_($j)

                    $ns_ attach-agent $AP(0) $sink_($j)
                    $ns_ connect $udp_($j) $sink_($j)
                    $cbr_($j) attach-agent $udp_($j)
                    set pru [expr 12.4+$j*48678/100000000.0];
```

```
                $ns_ at $pru "$cbr_($j) start"

                puts "Creat el trafic nvideo[expr $j]"
                set j [expr $j + 1]
}

for {set i 1} {$i <= $ndata1 } {incr i} {

            #Extraccion de datos movil
                set graphFd($j) [open "simple$j.gra" w]
                puts $graphFd($j) "0 0"
                set graphFiles "$graphFiles simple$j.gra"
                set graphlost($j) [open "lost$j.gra" w]
                puts $graphlost($j) "0 0"
                set graphLostFiles "$graphLostFiles lost$j.gra"
                set total($j) 0

                set udp_($j) [new Agent/UDP]
                set sink_($j) [new Agent/LossMonitor]
                set cbr_($j) [new Application/Traffic/CBR]

                $udp_($j) set packetSize_ 1500
                $udp_($j) set fid_ 2
                $udp_($j) set prio_ 2
                $udp_($j) set class_ 2
                $cbr_($j) set rate_ 800Kb
                $cbr_($j) set packetSize_ 1500

                $ns_ attach-agent $movil(1) $udp_($j)

                $ns_ attach-agent $AP(0) $sink_($j)
                $ns_ connect $udp_($j) $sink_($j)
                $cbr_($j) attach-agent $udp_($j)
                set pru [expr 12.4+$j*48678/100000000.0];
                $ns_ at $pru "$cbr_($j) start"

                puts "Creat el trafic ndata1[expr $j]"
                set j [expr $j + 1]

}

for {set i 1} {$i <= $ndata2 } {incr i} {

            #Extraccion de datos movil
                set graphFd($j) [open "simple$j.gra" w]
                puts $graphFd($j) "0 0"
                set graphFiles "$graphFiles simple$j.gra"
                set graphlost($j) [open "lost$j.gra" w]
                puts $graphlost($j) "0 0"
                set graphLostFiles "$graphLostFiles lost$j.gra"
                set total($j) 0

                set udp_($j) [new Agent/UDP]
                set sink_($j) [new Agent/LossMonitor]
                set cbr_($j) [new Application/Traffic/CBR]

                $udp_($j) set packetSize_ 1500
                $udp_($j) set fid_ 3
                $udp_($j) set prio_ 3
                $udp_($j) set class_ 3
                $cbr_($j) set rate_ 11Mb
```

```
                $cbr_($j) set packetSize_ 1500

                $ns_ attach-agent $movil(1) $udp_($j)

                $ns_ attach-agent $AP(0) $sink_($j)
                $ns_ connect $udp_($j) $sink_($j)
                $cbr_($j) attach-agent $udp_($j)
                set pru [expr 12.4+$j*48678/100000000.0];
                $ns_ at $pru "$cbr_($j) start"

                puts "Creat el trafic ndata1[expr $j]"
                set j [expr $j + 1]

}


set time 1.0
#$ns_ at $time "record"
$ns_ at 13 "record"

set numMeasurements 0

set totalnaudio 0.0
      set totalnvideo 0.0
      set totalndata1 0.0
      set totalndata2 0.0

      set parcialaudio 0.0
      set parcialvideo 0.0
      set parcialdata1 0.0
      set parcialdata2 0.0

      set a           [expr $naudio]
      set b           [expr $nvideo + $naudio]
      set c           [expr $nvideo + $naudio + $ndata1]
      set d           [expr $nvideo + $naudio + $ndata1 + $ndata2]

#set grapherror [open "error.gra" w]
#puts $grapherror "0 0"
#set netiface [$AP(0) getNetif 0]

proc record {} {
      global graphFd total udp tcp numMeasurements val time graphlost
netiface grapherror prio_ifq traf val totalnaudio totalnvideo
totalndata1 totalndata2 a b c d data1 data2 audio video u parcialaudio
parcialvideo parcialdata1 parcialdata2



            global sink_
            set ns [Simulator instance]
            set now [$ns now]

            for {set i 1} {$i < $val(nn) } {incr i} {

                  set bw [$sink_($i) set bytes_]
                  puts "$now [expr $bw/$time*8/1000000]"
                  puts $graphFd($i) "$now [expr $bw/$time*8/1000000]"
                  set total($i) [expr $bw/$time*8/1000000 + $total($i)]
                  $sink_($i) set bytes_ 0
```

```
            if {$i<=$a} {

                  set totalnaudio [expr $totalnaudio +
$bw/$time*8/1000000]
                  set parcialaudio [expr $parcialaudio +
$bw/$time*8/1000000]

                  if {$i==$a} {
                        puts $audio "$now [expr $parcialaudio]"
                        set parcialaudio 0.0
                        }


            }
            if {$i<=$b} {
                  if {$i>$a}  {

                  set totalnvideo [expr $totalnvideo +
$bw/$time*8/1000000]
                  set parcialvideo [expr $parcialvideo +
$bw/$time*8/1000000]

                  if {$i==$b} {
                        puts $video "$now [expr $parcialvideo]"
                        set parcialvideo 0.0
                        }
                  }
            }

            if {$i<=$c} {
                  if {$i>$b}  {

                  set totalndata1 [expr $totalndata1 +
$bw/$time*8/1000000]
                  set parcialdata1 [expr $parcialdata1 +
$bw/$time*8/1000000]

                  if {$i==$c} {
                        puts $data1 "$now [expr $parcialdata1]"
                        set parcialdata1 0.0
                        }
                  }
            }
            if {$i<=$d} {
                  if {$i>$c}  {

                  set totalndata2 [expr $totalndata2 +
$bw/$time*8/1000000]
                  set parcialdata2 [expr $parcialdata2 +
$bw/$time*8/1000000]

                  if {$i==$d} {
                        puts $data2 "$now [expr $parcialdata2]"
                        set parcialdata2 0.0
                  }
                  }
            }
}
      set numMeasurements [expr $numMeasurements + 1]
```

```
      $ns at [expr $now+$time] "record"

}

$ns_ at 80.0001 "stop"
$ns_ at 80.0002 "puts \"NS EXITING...\" ; $ns_ halt"

proc stop {} {
      global ns_ tracefd graphFd total numMeasurements val udp tcp
graphFiles graphlost traf a totalnaudio totalnvideo totalndata1
totalndata2 data1 data2 audio video
      close $tracefd

      set numClients [expr $val(nn) - 1]
      set totalTotal 0.0

      for {set i 1} {$i < $val(nn) } {incr i} {
            puts "AverageBandwidth([expr $i]) = [expr $total($i) /
$numMeasurements] Mbps"
            set totalTotal [expr $totalTotal + $total($i)]
            close $graphFd($i)
            close $graphlost($i)
      }

      puts "Average Total totalnaudio = [expr $totalnaudio /
$numMeasurements] Mbps"
      puts "Average Total totalnvideo = [expr $totalnvideo /
$numMeasurements] Mbps"
      puts "Average Total totalndata1 = [expr $totalndata1 /
$numMeasurements] Mbps"
      puts "Average Total totalndata2 = [expr $totalndata2 /
$numMeasurements] Mbps"

      puts "Average Total bandwidth1 = [expr $totalTotal /
$numMeasurements] Mbps"
      # call xgraph to display the results

      exec rm -f simple.gra
      exec rm -f lost.gra

      for {set i 1} {$i < $val(nn) } {incr i} {
            set graphAllFd [open "simple.gra" a]
            puts $graphAllFd ""
            set graphAllLostFd [open "lost.gra" a]
            puts $graphAllLostFd ""
            puts $graphAllLostFd "\"Nodo $i\""

                  puts $graphAllFd "\"UDP $i\""

            close $graphAllFd
            close $graphAllLostFd
            close $audio
            close $video
            close $data1
            close $data2

            exec cat simple$i.gra >> simple.gra
            exec cat lost$i.gra >> lost.gra
      }
```

```
      #exec xgraph simple.gra -geometry 800x400 &
}

puts "Starting Simulation..."
$ns_ run
```

2.2 Escenario_2.tcl

Se añade una fuente de video al escenario 1, generando tráfico a 2 Mbps.

2.3 Escenario_3.tcl

```
#
# ======================================================================
# Parametros de la linea de comandos
#
# ======================================================================
global argv arg0


set ran     [lindex $argv 4]; # Seed para el generador de numeros
aleatorios

#
# ======================================================================
# Define options
#
# ======================================================================
set val(chan)         Channel/WirelessChannel  ;# channel type
set val(prop)         Propagation/TwoRayGround ;# radio-propagation
model
set val(ant)          Antenna/OmniAntenna      ;# Antenna type
set val(ll)           LL                       ;# Link layer type
set val(ifq)          Queue/DTail/PriQ         ;# Interface queue type
set val(ifqlen)       50                       ;# max packet in ifq
set val(netif)        Phy/WirelessPhy          ;# network interface
type
set val(mac)          Mac/802_11e               ;# MAC type
set val(rp)           DSDV                     ;# ad-hoc routing
protocol
set val(nn)           3      ;     #[expr $nvideo + $naudio + $ndata1
+ $ndata2 + 1]      number of mobilenodes

# including AP.
set val(seed)         $ran;              # Seed para el generador de

#
# ======================================================================
# Parametros de nodos 802.11
#
# ======================================================================
Mac/802_11 set delay_ 64us          ;#MAC Processing delay. En el b 0
(not applicable) en el g <2micros
```

```
Mac/802_11 set ifs_ 16us            ;# No especificaciones, no se
utiliza en el codigo
Mac/802_11 set slotTime_ 20us       ;# 20 en el b 20 o 9 en el g
Mac/802_11 set cwmin_ 31            ;# 31 en el b 31 o 15 en el g
Mac/802_11 set cwmax_ 1023          ;#1024 en ambos
Mac/802_11 set rtxLimit_ 16         ;#No especificaciones
Mac/802_11 set bssId_ -1
Mac/802_11 set sifs_ 10us           ;# 10 en ambos
Mac/802_11 set pifs_ 30us           ;#SIFS+SlotTime
Mac/802_11 set difs_ 50us           ;#SIFS+2SlotTime
Mac/802_11 set rtxAckLimit_ 1
Mac/802_11 set rtxRtsLimit_ 3
Mac/802_11 set basicRate_ 11Mb  ;# set this to 0 if want to use
bandwidth_ for
Mac/802_11 set dataRate_ 11Mb   ;# both control and data pkts


Mac/802_11 set CWMin_          31          ;#
Mac/802_11 set CWMax_          1024        ;#
Mac/802_11 set SlotTime_       0.000020         ;# 20us
Mac/802_11 set SIFS_           0.000010         ;# 10us
Mac/802_11 set PreambleLength_         144              ;# 144 bit
Mac/802_11 set PLCPHeaderLength_       48               ;# 48 bits
Mac/802_11 set PLCPDataRate_  1.0e6           ;# 1Mbps
Mac/802_11 set RTSThreshold_  0               ;# bytes
Mac/802_11 set ShortRetryLPLCPimit_       7                ;#
retransmittions
Mac/802_11 set LongRetryLimit_         4               ;#
retransmissions


#
# ============================================================================
# Parametros de nodos 802.11e Per 802.11b
#
# ============================================================================
Mac/802_11e set delay_ 64us ;#a MAC processing delay, antes ponia 64us
Mac/802_11e set ifs_ 16us  ;#antes 16us, pero no se utiliza en el
codigo
Mac/802_11e set slotTime_ 20us ;#antes 16us
Mac/802_11e set cwmin_ 31
Mac/802_11e set cwmax_ 1023
Mac/802_11e set rtxLimit_ 16
Mac/802_11e set bssId_ -1
Mac/802_11e set sifs_ 10us ;#8us
Mac/802_11e set pifs_ 30us ;#SIFS+SlotTime
Mac/802_11e set difs_ 50us ;#SIFS+2SlotTime
Mac/802_11e set rtxAckLimit_ 1
Mac/802_11e set rtxRtsLimit_ 3
Mac/802_11e set basicRate_ 11Mb  ;# set this to 0 if want to use
bandwidth_ for
Mac/802_11e set dataRate_ 11Mb   ;# both control and data pkts
Mac/802_11e set cfb_ 1 ; # 0 disables CFB

Mac/802_11e set SlotTime_      0.000020        ;# 20us
Mac/802_11e set SIFS_          0.000010        ;# 10us
Mac/802_11e set PreambleLength_        144             ;# antes 144
bit
Mac/802_11e set PLCPHeaderLength_      48              ;# antes 48
bits
Mac/802_11e set PLCPDataRate_ 1.0e6           ;# 1Mbps
```

```
Mac/802_11e set RTSThreshold_  0               ;# bytes
Mac/802_11e set ShortRetryLimit_       7               ;#
retransmittions
Mac/802_11e set LongRetryLimit_        4               ;#
retransmissions

ns-random $val(seed)

Agent/TCP set packetSize_ 1500
Agent/TCPSink/DelAck set interval_ 200ms

Phy/WirelessPhy set RXThresh_ 3.652e-10

set ns_    [new Simulator]
set tracefd     [open ns.tr w]
$ns_ trace-all $tracefd
set graphFiles ""
set graphLostFiles ""

set audio [open audio.tr w]
set video [open video.tr w]
set data1 [open data1.tr w]
set data2 [open data2.tr w]


set topo    [new Topography]
$topo load_flatgrid 500 500

create-god $val(nn);#Hay que poner el numero total de nodos moviles

set chan [new $val(chan)]
#configuracion del enrutamiento jerarquico

$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2           ;# number of domains

lappend cluster_num 1 1               ;# number of clusters in each
domain
AddrParams set cluster_num_ $cluster_num

lappend eilastlevel 1 [expr $val(nn)-1]               ;# number of
nodes in each cluster
AddrParams set nodes_num_ $eilastlevel ;# of each domain

#Creaci?n del nodo wired
set temp {0.0.0 1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8
1.0.9 1.0.10 1.0.11 1.0.12 1.0.13 1.0.14 1.0.15 1.0.16 1.0.17 1.0.18
1.0.19 1.0.20 1.0.21 1.0.22 1.0.23 1.0.24 1.0.25 1.0.26 1.0.27 1.0.28
1.0.29 1.0.30 1.0.31 1.0.32 1.0.33 1.0.34 1.0.35 1.0.36 .0.37 1.0.38
1.0.39 1.0.40 1.0.41 1.0.42 1.0.43 1.0.44}

# Configurar AP
$ns_ node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
```

```
-channel $chan \
-wiredRouting ON \
-agentTrace ON \
-routerTrace OFF\
-macTrace ON \
-movementTrace OFF \
-eotTrace OFF

set AP(0) [$ns_ node [lindex $temp 0]] ;#direccion del AP
$AP(0) random-motion 0  ;#deshabilitar movimiento del AP


#Configuracion de su MAC
set macAP [$AP(0) getMac 0]
set direccionAP [$macAP id]
puts "Direccion AP0: $direccionAP"
$macAP bss_id $direccionAP
$macAP set Short11bPreamble_ true
$macAP set RTSThreshold_ 2500
#$macAP set FrameErrorRate_ 5.0
$macAP set basicRate_ 11Mb
$macAP set dataRate_ 11Mb

#
# Prioridades del AP
#

#Situacion del AP0
$AP(0) set X_ 0.0
$AP(0) set Y_ 0.0
$AP(0) set Z_ 0.0

#Creacion de los nodos moviles
$ns_ node-config -wiredRouting OFF

#creo els nodes
for {set i 1} {$i <= $val(nn) } {incr i} {

      set movil($i) [$ns_ node [lindex $temp $i]]
      $movil($i) base-station [AddrParams addr2id [$AP(0) node-addr]]
      $movil($i) random-motion 0;  #deshabilitar movimiento

      #Configuracion de su MAC
      set mac [$movil($i) getMac 0]
      $mac bss_id $direccionAP
      $mac set proto_on 0

      $mac set Short11bPreamble_ true
      $mac set RTSThreshold_ 2500
      #$mac set FrameErrorRate_ 5.0
      #$mac set basicRate_ 11Mb                    S'especifica
segun especificaciones
      #$mac set dataRate_  11Mb    ;#$tas_($i)


      $movil($i) set X_ 70.0
      $movil($i) set Y_ 0.0
      $movil($i) set Z_ 0.0
}
```

```
#creo el trafic node a node definint en cada cas quin trafic hi ha

                  set total(1) 0
                  set graphFd(1) [open "simple1.gra" w]
                  puts $graphFd(1) "0 0"

                  set udp_(1) [new Agent/UDP]
                  set sink_(1) [new Agent/LossMonitor]
                  set cbr_(1) [new Application/Traffic/CBR]

                  $udp_(1) set packetSize_ 1500
                  $udp_(1) set fid_ 2
                  $udp_(1) set prio_ 2
                  $udp_(1) set class_ 2
                  $cbr_(1) set rate_ 3000Kb
                  $cbr_(1) set packetSize_ 1500

                  $ns_ attach-agent $movil(1) $udp_(1)

                  $ns_ attach-agent $AP(0) $sink_(1)
                  $ns_ connect $udp_(1) $sink_(1)
                  $cbr_(1) attach-agent $udp_(1)
                  set pru [expr 12.4+1*48678/100000000.0];
                  $ns_ at $pru "$cbr_(1) start"

                  puts "Creat el trafic estaciÃ³n"

set total(2) 0
set graphFd(2) [open "simple2.gra" w]
puts $graphFd(2) "0 0"
set udp_(2) [new Agent/UDP]
                  set sink_(2) [new Agent/LossMonitor]
                  set cbr_(2) [new Application/Traffic/CBR]

                  $udp_(2) set packetSize_ 1500
                  $udp_(2) set fid_ 2
                  $udp_(2) set prio_ 2
                  $udp_(2) set class_ 2
                  $cbr_(2) set rate_ 3000Kb
                  $cbr_(2) set packetSize_ 1500

                  $ns_ attach-agent $movil(2) $udp_(2)

                  $ns_ attach-agent $AP(0) $sink_(2)
                  $ns_ connect $udp_(2) $sink_(2)
                  $cbr_(2) attach-agent $udp_(2)
                  set pru [expr 12.4+2*48678/100000000.0];
                  $ns_ at $pru "$cbr_(2) start"

                  puts "Creat el trafic estaciÃ³n2"

set total(3) 0
set graphFd(3) [open "simple3.gra" w]
puts $graphFd(3) "0 0"
set udp_(3) [new Agent/UDP]                                    ;#trafico
de video en el nodo3
                  set sink_(3) [new Agent/LossMonitor]
                  set cbr_(3) [new Application/Traffic/CBR]

                  $udp_(3) set packetSize_ 1500
                  $udp_(3) set fid_ 1
```

```
                $udp_(3) set prio_ 1
                $udp_(3) set class_ 1
                $cbr_(3) set rate_ 2000Kb
                $cbr_(3) set packetSize_ 1500

                $ns_ attach-agent $movil(3) $udp_(3)

                $ns_ attach-agent $AP(0) $sink_(3)
                $ns_ connect $udp_(3) $sink_(3)
                $cbr_(3) attach-agent $udp_(3)
                set pru [expr 12.4+3*48678/100000000.0];
                $ns_ at $pru "$cbr_(3) start"

set total(4) 0
            set graphFd(4) [open "simple4.gra" w]
            puts $graphFd(4) "0 0"
            set udp_(4) [new Agent/UDP]                    ;#trafico
de audio en el nodo3
                set sink_(4) [new Agent/LossMonitor]
                set cbr_(4) [new Application/Traffic/CBR]

                $udp_(4) set packetSize_ 80
                $udp_(4) set fid_ 0
                $udp_(4) set prio_ 0
                $udp_(4) set class_ 0
                $cbr_(4) set rate_ 1000Kb
                $cbr_(4) set packetSize_ 80

                $ns_ attach-agent $movil(3) $udp_(4)

                $ns_ attach-agent $AP(0) $sink_(4)
                $ns_ connect $udp_(4) $sink_(4)
                $cbr_(4) attach-agent $udp_(4)
                set pru [expr 12.4+4*48678/100000000.0];
                $ns_ at $pru "$cbr_(4) start"


set time 1.0
#$ns_ at $time "record"
$ns_ at 13 "record"

set numMeasurements 0

set totalnaudio 0.0
      set totalnvideo 0.0
      set totalndata1 0.0
      set totalndata2 0.0

      set parcialaudio 0.0
      set parcialvideo 0.0
      set parcialdata1 0.0
      set parcialdata2 0.0



#set grapherror [open "error.gra" w]
#puts $grapherror "0 0"
#set netiface [$AP(0) getNetif 0]

proc record {} {
```

```
      global graphFd total udp tcp numMeasurements val time graphlost
netiface grapherror prio_ifq traf val totalnaudio totalnvideo
totalndata1 totalndata2 a b c d data1 data2 audio video u parcialaudio
parcialvideo parcialdata1 parcialdata2



            global sink_
            set ns [Simulator instance]
            set now [$ns now]

            for {set i 1} {$i <= 4 } {incr i} {

                  set bw [$sink_($i) set bytes_]
                  puts "$now [expr $bw/$time*8/1000000]"
                  puts $graphFd($i) "$now [expr $bw/$time*8/1000000]"
                  set total($i) [expr $bw/$time*8/1000000 + $total($i)]
                  $sink_($i) set bytes_ 0



            if {$i==4}  {

                  set totalnaudio [expr $totalnaudio +
$bw/$time*8/1000000]
                  set parcialaudio [expr $parcialaudio +
$bw/$time*8/1000000]


                        puts $audio "$now [expr $parcialaudio]"
                        set parcialaudio 0.0



            }
            if {$i==3}  {


                  set totalnvideo [expr $totalnvideo +
$bw/$time*8/1000000]
                  set parcialvideo [expr $parcialvideo +
$bw/$time*8/1000000]


                        puts $video "$now [expr $parcialvideo]"
                        set parcialvideo 0.0


            }

            if {$i<3}   {


                  set totalndata1 [expr $totalndata1 +
$bw/$time*8/1000000]
                  set parcialdata1 [expr $parcialdata1 +
$bw/$time*8/1000000]

                  if {$i==2}  {
                        puts $data1 "$now [expr $parcialdata1]"
                        set parcialdata1 0.0
```

```
                        }

            }

}
      set numMeasurements [expr $numMeasurements + 1]
      $ns at [expr $now+$time] "record"

}

$ns_ at 80.0001 "stop"
$ns_ at 80.0002 "puts \"NS EXITING...\" ; $ns_ halt"

proc stop {} {
      global ns_ tracefd graphFd total numMeasurements val udp tcp
graphFiles graphlost traf a totalnaudio totalnvideo totalndata1
totalndata2 data1 data2 audio video
      close $tracefd

      set numClients [expr $val(nn) - 1]
      set totalTotal 0.0

      for {set i 1} {$i <= 4 } {incr i} {
            puts "AverageBandwidth([expr $i]) = [expr $total($i) /
$numMeasurements] Mbps"
            set totalTotal [expr $totalTotal + $total($i)]
            close $graphFd($i)

      }

      puts "Average Total totalnaudio = [expr $totalnaudio /
$numMeasurements] Mbps"
      puts "Average Total totalnvideo = [expr $totalnvideo /
$numMeasurements] Mbps"
      puts "Average Total totalndata1 = [expr $totalndata1 /
$numMeasurements] Mbps"


      puts "Average Total bandwidth1 = [expr $totalTotal /
$numMeasurements] Mbps"
      # call xgraph to display the results



      for {set i 1} {$i < $val(nn) } {incr i} {
            set graphAllFd [open "simple.gra" a]
            puts $graphAllFd ""
            puts $graphAllFd "\"UDP $i\""

            close $graphAllFd
            close $audio
            close $video
            close $data1
            close $data2

            exec cat simple$i.gra >> simple.gra


      }
```

```
      #exec xgraph simple.gra -geometry 800x400 &
}

puts "Starting Simulation..."
$ns_ run
```