Máster Universitario en Tecnologías de la Información y Comunicaciones en Redes Móviles

Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Caracterización y planificación del tráfico en una red Digital Signage

TESIS DE FIN DE MÁSTER

Autor: Jorge David de Hoz Diego

Director: José Ruiz Mas

Marzo 2013

RESUMEN

Este trabajo ofrece una visión general del funcionamiento de una red de *Digital Signage* actual y de la problemática que ha de hacer frente. Se describen los aspectos de funcionamiento que más repercusión tienen en el sistema, agilidad del mismo y su fiabilidad. Se definen qué aspectos son más importantes en la calidad de servicio experimentada por el usuario final o por el gestor de la red y se ofrecen propuestas para intentar mejorar estos aspectos desde el ámbito de la telemática con la tecnología existente actualmente.

En primer lugar se aborda una breve descripción de la red de *Digital Signage* que es objeto de estudio y de los elementos involucrados en la misma. Se expone una descripción del cometido de cada uno de los dispositivos y del esquema de funcionamiento general.

A continuación se detectan algunos de los problemas más relevantes existentes en dos ámbitos fundamentales. El primero hace referencia a los aspectos que están relacionados con la calidad de servicio en las comunicaciones entre los dispositivos de *Digital Signage (players),* su servidor central y el terminal del usuario o gestor de red. El segundo se centra en analizar exclusivamente el comportamiento *players* en entornos *wireless*. En ambos ámbitos se analizan las causas de los problemas detectados y se proponen soluciones. Algunas de ellas se implementan y se valoran los resultados obtenidos.

Finalmente se plantean las conclusiones y se definen futuras líneas de acción.

Índices

ÍNDICE DE CONTENIDOS

INDICE DE CONTENIDOS **3**

INDICE DE FIGURAS **5**

INDICE DE TABLAS **7**

1. INTRODUCCIÓN **15**

1.1. DESCRIPCIÓN DEL PROBLEMA 15

1.2. OBJETIVOS 16

1.3. ESTRUCTURA DE LA MEMORIA 16

2. ESTRUCTURA BÁSICA DE UNA RED DS Y SU FUNCIONAMIENTO **19**

2.1. COMUNICACIÓN ENTRE EL SERVIDOR Y LOS PLAYERS 21

2.1.1. Conectividad con Internet 21

2.1.2. Establecimiento de las conexiones tuneladas 22

2.1.3. Servidor de *Digital Signage*: Gestión de conexiones 23

2.1.3.1. Monitorización de los players *24*

2.1.3.2. Gestión de los players *24*

2.1.3.3. Distribución de contenidos *26*

3. QOS EN LA GESTIÓN DE LA RED **29**

3.1. ASPECTOS MEJORABLES EN LA RED 29

3.1.1. Minimizar el tráfico de control 29

3.1.2. Sistema de distribución de contenidos en diferido 30

3.1.3. Diferenciación del tráfico 32

3.1.4. Optimización del protocolo SSH en comunicaciones tuneladas 33

3.2. MEJORA DE LA LATENCIA MEDIANTE EL CONTROL DE CONGESTIÓN 35

3.2.1. Herramientas para medir el *Round-Trip delay Time* (RTT) 35

3.2.1.1. Dtrace 36

3.2.1.2. Hping3 37

3.2.1.3. Tcpdump y tcptrace *37*

3.2.1.4. Medida directa del RTT sobre mensajes echo tunelados *37*

3.2.2. Impacto en la latencia del tráfico debido a los túneles que comparten sesión por multiplexación. 38

3.2.2.1. Metodología *38*

3.2.2.2. Creación de los túneles *40*

3.2.2.3. Ejecución de los clientes echo para obtener RTTs *40*

3.2.2.4. Comparativa del RTT entre un túnel de sesión única y otro multiplexado con una conexión SFTP *42*

3.2.2.5. Ajuste fino del tamaño de buffer de SFTP *44*

3.2.2.6. Control de congestión con TCP Vegas *45*

4. QOS CON ENLACES 802.11 **51**

4.1. DESCRIPCIÓN DE WIRELESS MULTIMEDIA (WMM) 51

4.2. WMM Y EDCA: CLASIFICACIÓN DE LAS COLAS 52

4.3. MAPEADO QOS EN WMM 54
4.3.1. Mapeado QoS desde el nivel de enlace 54
4.3.2. Mapeado QoS desde el nivel de red 55
4.4. MEDIDAS EXPERIMENTALES 59
4.4.1. Primer escenario 59
4.4.1.1. Uso de clientes 802.11 *63*
4.4.2. Segundo escenario 65
5. CONCLUSIONES **71**
6. ANEXO: DISPOSITIVOS HARDWARE **75**
6.1. PEPWAVE SURF ON-THE-GO 75
6.2. ESD2: DIGITAL SIGNAGE PLAYER 76
6.3. RASPBERRY PI COMO DIGITAL SIGNAGE PLAYER 78
7. ACRÓNIMOS **83**
8. BIBLIOGRAFÍA **87**

ÍNDICE DE FIGURAS

Figura 1-1: Red DS en distintos puntos geográficos 15
Figura 2-1: Ejemplo de contenido auto-gestionable. Plantilla basada en flash que confecciona la oferta con una foto, un texto, un precio y una descripción. 19
Figura 2-2: Comunicación del *player* con el servidor de red 20
Figura 2-3: Gestión de la red a través del servidor 20
Figura 2-4: Etapas en el establecimiento de los túneles del *player* con el servidor 22
Figura 2-6: Comunicación del terminal de control con un *player* concreto a través del servidor 23
Figura 2-7: Ejemplo de monitorización de los parámetros básicos del *player* de forma remota 24
Figura 2-8: Ejemplo de gestión del *player* vía interfaz web de forma remota 25
Figura 2-9: Captura de la interfaz web del servidor y de la interfaz de gestión de un player concreto 25
Figura 2-10: Ejemplo de distribución de contenidos de forma remota a través del servidor 26
Figura 3-1: Diagrama de flujo para la comprobación de los túneles de comunicación 30
Figura 3-2: Etapas en la actualización de los contenidos 31
Figura 3-3: Problemática típica en enlaces encadenados tras los que ofrecer QoS E2E 32
Figura 3-4: Modificaciones propuestas a OpenSSH para gestionar los túneles mediante sockets UDP 33
Figura 3-5: Resultado tras emular pérdidas de paquetes ACKs [3] 33
Figura 3-6: Establecimiento tradicional de los tres túneles en sesión única mediante multiplexación. Se plantea cambiar este esquema por sesiones independientes. 35
Figura 3-7: Arquitectura general de Dtrace [5] 36
Figura 3-8: Escenario de funcionamiento cliente/servidor *echo* para medir RTTs a través de túneles SSH 37
Figura 3-9: Escenario de pruebas propuesto para el análisis del impacto de tráfico pesado en túneles SSH 39
Figura 3-10: RTTs de un túnel SSH sin tráfico extra en otros túneles que comparten sesión 42
Figura 3-11: Comparativa de RTT's entre túneles SSH de sesión única y con sesión compartida. 43
Figura 3-12: Comparativa de *RTT* entre túneles SSH con de sesión única y compartida, ambos con el tamaño de *buffer* ajustado 44
Figura 3-13: Modificación del *kernel* del *player* para dar soporte a TCP Vegas 46
Figura 3-14: Comparativa de RTTs de túneles SSH con sesión única y con sesión compartida funcionando con TCP Vegas 47
Figura 4-1: Relaciones existentes en el espacio entre trama (IFS)[12] 52
Figura 4-2: Temporizaciones en WMM [13] 52
Figura 4-3: Colas de transmisión en un cliente WMM [13] 53
Figura 4-4: Modificación en la trama Ethernet para incluir el marcaje 802.1Q 54
Figura 4-5: Activación de *SKB editing* en *Networking support -> Networking options -> QoS and/or*

fair queuing 55
Figura 4-6: Ubicación del campo ToS en IPV4 y sus partes principales..... 56
Figura 4-7: Escenario de pruebas para comprobar el funcionamiento de EDCA en caso de flujos de tráfico múltiples en el propio dispositivo..... 59
Figura 4-8: Mensajes *debug* enviados por el módulo Ralink WI-FI del *Kernel* captados con *dmesg*..... 60
Figura 4-9: Estado del dispositivo USB 802.11 utilizado durante las pruebas. 60
Figura 4-10: Resultados de las medidas del primer escenario: Flujos con diversas clasificaciones *ToS* modelados automáticamente por EDCA 62
Figura 4-11: Alternativa al uso de hardware USB 802.11: Cliente Wi-Fi. 63
Figura 4-12: Medidas tomadas con un portátil en modo monitor a nivel 802.11..... 64
Figura 4-13: Escenario de pruebas para comprobar el funcionamiento de *EDCA* en caso de que exista tráfico de fondo sustancial de otro dispositivo de la red. 65
Figura 4-14: Estado de la interfaz de red 802.11 del ordenador utilizado durante las pruebas 66
Figura 4-15: Resultados de las medidas del segundo escenario: Tráfico del *player* con distintas prioridades compaginado con tráfico de un ordenador de la misma red sin ningún tipo de prioridad. 67
Figura 6-1: Unidad *PEPWAVE Surf on-the-go* para integración 75
Figura 6-2: Interfaz gráfica de control del *Pepwave Surf on-the-Go* dando cobertura a un *player* DS en funcionamiento..... 76
Figura 6-3: Esd2: Player de Digital Signage desarrollado en Ateire Tecnología y Comunicación 76
Figura 6-4: Esd2: Interior del dispositivo..... 77
Figura 6-5: Raspberry PI modelo B 78
Figura 6-6: Arquitectura interna de Raspberry PI 78

ÍNDICE DE TABLAS

Tabla 3-1: Descripción de los túneles empleados en cada *player* 23

Tabla 3-1: Configuración de los túneles empleados en las pruebas experimentales 39

Tabla 3-2: Valores de referencia de latencias obtenidos como promedio de 16 medidas espaciadas a lo largo de las pruebas experimentales. 41

Tabla 3-3: Datos estadísticos de RTT de un túnel SSH en ausencia de carga en la conexión SFTP 42

Tabla 3-4: Datos estadísticos de RTT en túneles SSH con tráfico en túneles que comparten sesión..... 43

Tabla 3-5: Datos estadísticos de RTT en túneles SSH con tráfico en túneles que comparten sesión con el tamaño del buffer SFTP ajustado 44

Tabla 3-6: Datos estadísticos de RTT en túneles SSH SSH con sesión única y con sesión compartida funcionando con TCP Vegas..... 47

Tabla 4-1: Listado de las categorías de acceso en WMM y su correspondencia con las prioridades 802.1d de Ethernet [11] 51

Tabla 4-2: Parámetros por defecto de *EDCA* para 802.11 y WMM [11] 53

Tabla 4-3: Nomenclatura de los bits del campo *ToS* en *IPv4*[16] 56

Tabla 4-4: Correspondencia de los niveles de prioridad *DSCP* con los *tags* 802.1D y los AC de 802.1e y WMM[13]..... 57

Tabla 4-5: Valores del ToS reconocidos por *DiffServ* y su correspondencia con *DSCP codepoints* y *PHB's*..... 58

Tabla 4-6: Correspondencias *QoS* en los *AP's* de Cisco[20]..... 58

Tabla 4-7: Correspondencias empíricas entre *Class of Service* (CoS) y *Differenciated Services Code Point* (DSCP) en el primer escenario 62

Tabla 4-8: Correspondencias empíricas entre *CoS* y *DSCP* en el segundo escenario 67

Introducción

1. INTRODUCCIÓN

Las redes *Digital Signage* (DS) están formadas por elementos que reproducen contenidos audiovisuales con capacidad opcional de interactividad dependiendo del escenario. Estos dispositivos suelen estar gobernados por micro-computadores que coordinan la reproducción de los contenidos audiovisuales en pantallas, proyectores, etc. y que gestionan la interactividad. Se les denomina comúnmente *players*.

El tipo de red DS a tratar se despliega tomando *Internet* como red troncal y un bucle de abonado compartido como punto de acceso. La conexión establecida con el punto de acceso es generalmente Wi-Fi 802.11. Todos los dispositivos se gestionan de forma centralizada por medio de uno o varios servidores dedicados en *Internet.*

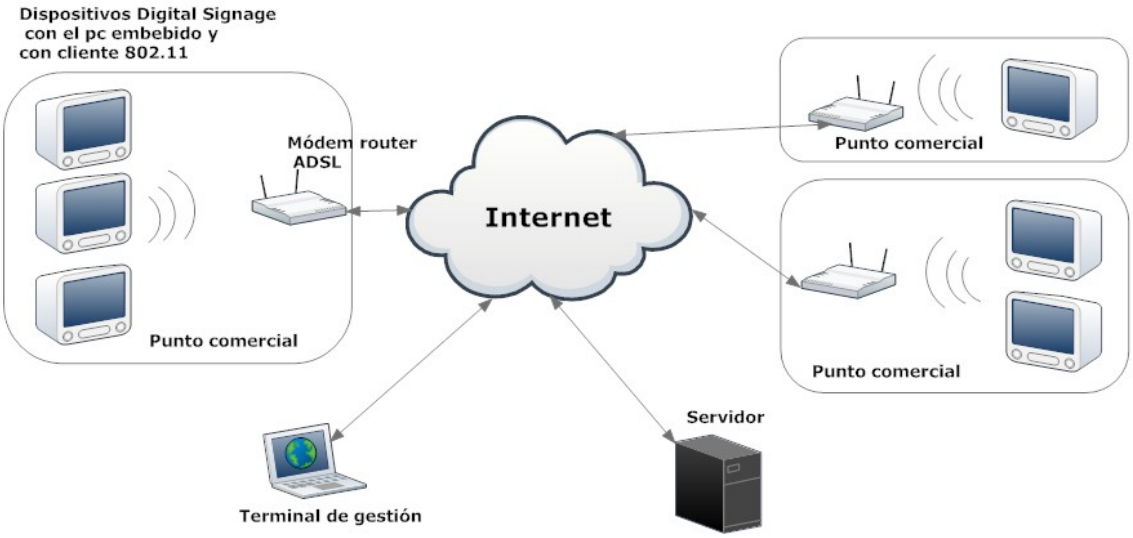

Figura 1-1: Red DS en distintos puntos geográficos

1.1. DESCRIPCIÓN DEL PROBLEMA

Este tipo de red se constituye con una topología muy variada que siempre incluye a Internet como red troncal. Además, la ubicación geográfica del servidor no está predefinida. En este documento se analiza una red DS real en la que el servidor central se encuentra en Montreal, Canadá, mientras que la totalidad de los *players* se encuentran en España.

Dado que el control y gestión se realizan de forma centralizada a través del servidor, ocurren diversos problemas que influyen en la fluidez del funcionamiento de las aplicaciones de gestión y en los contenidos audiovisuales. Este efecto implica una merma de la calidad de servicio percibido por el gestor de la red o usuarios de la misma que se deben minimizar. Estos problemas tendrán principalmente dos causas. La primera es la congestión en las comunicaciones *End-to-End* (E2E) tuneladas con el servidor, que introducen latencias elevadas. La segunda se centra en los escenarios en los que se emplean entornos *wireles* masificados, como ferias, centros comerciales y *hotspots* en general. En estos casos el cuello de botella suele estar en el enlace *wireless* del *player* con la red.

1.2. OBJETIVOS

El primer objetivo de este estudio busca minimizar la latencia E2E en la red DS mediante modificaciones y complementos a los algoritmos de comunicaciones y protocolos utilizados que consigan minimizar la congestión y por tanto la latencia; un aspecto muy negativo en la interactividad tanto de los contenidos audiovisuales como en la gestión de los dispositivos. Para ello se estudian todos los elementos principales que intervienen en las comunicaciones entre el *player* y el servidor y se proponen posibles soluciones. A lo largo de la memoria se proyecta implementar alguna de ellas y constatar la mejora esperada con medidas experimentales.

En el ámbito local *wireless* el objetivo consiste en aprovechar las prestaciones que esta tecnología puede ofrecer para el tráfico diferenciado. Se pretenden priorizar las comunicaciones de control y gestión sobre el resto de tráfico y para ello se analizan las alternativas posibles y se plantean soluciones que se valoran en base a los resultados obtenidos de las medidas experimentales realizadas.

1.3. ESTRUCTURA DE LA MEMORIA

Este documento se divide en cuatro capítulos. El primero describe los problemas existentes y los objetivos de trabajo planteados. El segundo capítulo consiste en una breve introducción a la red Digital Signage y a su modo de operación. Se centra en la forma en la que se establecen las conexiones entre los dispositivos para poder disponer de una perspectiva más completa a la hora de abordar los problemas.

En el tercer capítulo se analizan los protocolos de comunicación utilizados y la diferenciación del tráfico empleado en estos sistemas. También se estudia la congestión y el efecto del tráfico entre los flujos diferenciados dentro de un mismo túnel. Se describen las herramientas usadas para medir latencias y se proponen soluciones para cada problema planteado.

En el cuarto capítulo se aborda el entorno *wireless* local y se profundiza en la tecnología de calidad de servicio (QoS) de 802.11. Dado que los escenarios en los que se trabaja normalmente no permiten configurar los *Access Points* (APs) y se desconoce qué tecnologías implementa cada modelo de AP a priori, se centra el análisis en los procedimientos de gestión de acceso al medio definidos por *Enhanced Distributed Carrier Access* (EDCA) y cómo se pueden utilizar para priorizar el tráfico de control y gestión de los *players*. Finalmente se analizan dos dispositivos 802.11 y se realizan medidas para comprobar el funcionamiento de EDCA.

Finalmente, en la sección 5 se presentan las conclusiones y líneas futuras de trabajo.

Estructura básica de una red DS y su funcionamiento

2. ESTRUCTURA BÁSICA DE UNA RED DS Y SU FUNCIONAMIENTO

La tecnología *Digital Signage* no se encuentra estandarizada y cada fabricante de dispositivos propone su propia solución. Este hecho se debe en parte a que falta una definición clara de las funcionalidades que debe disponer una red de este tipo. En Ateire Tecnología y Comunicación, siendo conscientes de este problema, hemos optado por desarrollar nuestra solución basándonos en tecnología web principalmente. Este aspecto dota a nuestro sistema de la flexibilidad necesaria para ser capaz de adaptarse a las necesidades de cada proyecto concreto. Además, al basarse en tecnología abierta facilita la integración de la red DS en otros sistemas de información de terceros. En esta red encontramos dos elementos fundamentales, los players y el servidor central.

Los *players* constituyen el elemento básico de una red de *Digital Signage*. Pueden ubicarse en cualquier punto donde el propietario de los mismos desee realizar comunicación audiovisual dirigida a trabajadores, potenciales clientes, etc. Estos *players* muestran por una pantalla contenidos que se pueden tener alojados local ó remotamente. Los contenidos pueden estar basados en fotografía, video ó aplicaciones web más complejas que estén dotadas de cierto grado de interactividad y se conecten a sistemas de información ajenos a la red DS. Un ejemplo de uso generalizado se encuentra en el ámbito publicitario con las plantillas del tipo "oferta de productos" que disponen de conexión a bases de datos de donde obtienen la información de la oferta para generarla *on-the-fly*.

Figura 2-1: Ejemplo de contenido auto-gestionable. Plantilla basada en flash que confecciona la oferta con una foto, un texto, un precio y una descripción.

El funcionamiento de estos dispositivos es autónomo una vez quedan programados. Incluso en casos de falta de conectividad, continúan funcionando reproduciendo únicamente contenidos locales. La figura del servidor central se reserva para operaciones de monitorización, control y distribución de contenidos para reproducción en diferido.

El gestor de la red DS se conecta a este servidor por interfaz web facilitando así su accesibilidad desde la mayoría de dispositivos con navegadores web. En este portal, el gestor de la red puede listar todos los *players* que conforman la misma y monitorizar su estado actual (conectado/desconectado, último sondeo realizado, parrilla de contenidos actual, contenido actualmente en reproducción, instantáneas en miniatura actualizadas del mismo, etc.). El gestor de la red también puede controlar cualquier *player* en concreto. El sistema de gestión de los *players* se realiza también vía interfaz web y el servidor es capaz de

establecer conexión directa a los servidores web de los *players* para que el gestor de red pueda controlarlos de forma directa si fuese necesario. Antes de exponer las restantes funciones del servidor central conviene definir algunos conceptos:

- **Canal de contenidos**: Consiste en un conjunto de contenidos que tienen ciertas características en común y que se agrupan en un canal para simplificar su distribución.
- **Grupo de *players***: Se define como una agregación de estos dispositivos con alguna característica en común: mismo cliente, misma área geográfica, etc.
- **Parrilla de contenidos**: se define como la forma de organizar los contenidos de un grupo o de un único *player* para su reproducción. Existen varios factores a determinar: los contenidos que van a formar parte de la parrilla, el orden de los mismos y su duración en caso de no tener la asignada intrínsecamente; como ocurre con las fotos, aplicaciones web ó las animaciones cíclicas. En el caso de los vídeos no sería necesario.

El servidor central permite la creación de grupos de *players*, canales de contenidos y la asignación de estos canales a los grupos creados. También gestiona el proceso de sincronización de los canales a los grupos. Las comunicaciones entre los *players* y el servidor central se realizan mediante túneles SSH (*Secure SHell*) para dotar a las comunicaciones de seguridad dado que se utiliza Internet como red troncal.

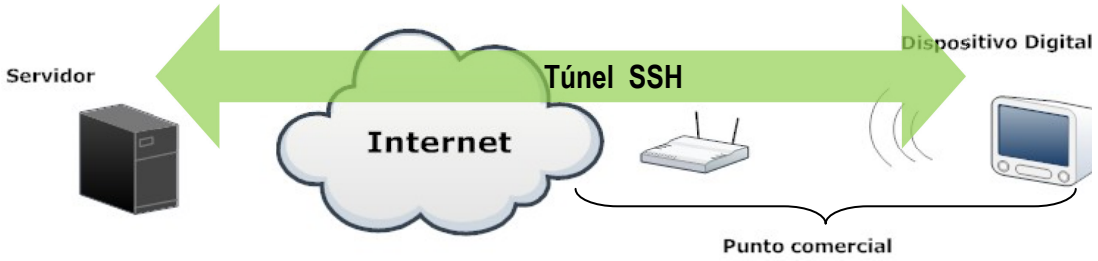

Figura 2-2: Comunicación del *player* con el servidor de red

En base a los servicios definidos y la forma de conexión elegida se pueden exponer algunas consideraciones generales sobre las características de las comunicaciones entre servidor central, player y gestor de red:

- Los contenidos que obtiene el *player* pueden proceder del servidor o de otro punto de *Internet*: Una página web, video en *streaming*, *rss feeds*, etc. Los contenidos del servidor siempre son en diferido y se precargan en el *player*, mientras que los que se pueden obtener por *Internet* pueden ser en tiempo real.

- Por otro lado, el gestor de red puede acceder a los servicios que ofrece la red DS a través del servidor central. Este supone un punto neurálgico de la red donde se concentra la información y el estado de todos los *players* de la red. El servidor ofrece servicios de gestión de los contenidos y la programación de los *players*.

Figura 2-3: Gestión de la red a través del servidor

- A su vez, el servidor puede compartir túneles SHH con el gestor de red si éste requiere comunicación directa con el *player*. Las comunicaciones del gestor de red con el servidor son vía web en usando HTTP/HTTPS. Las comunicaciones de gestión sobre un *player* concreto de la red circulan del terminal del gestor de red al *player* a través del servidor central por medio de un túnel *ssh* que el servidor tiene reservado con ese cometido para cada *player*.

Tras esta breve explicación, se pueden identificar 4 tipos de tráfico que cursará esta red:

- **Tráfico de control**: Generado por las tareas de control y monitorización entre el *player* y el servidor.
- **Trafico de gestión**: Generado por las tareas que el administrador de la red realiza en el servidor mediante su terminal de gestión.
- **Tráfico de contenidos**: Puede ser de dos tipos:
    - En tiempo real: Contenidos solicitados por *player* que se reproducirán en el momento de su solicitud.
    - En diferido: Contenidos que se descargan en el *player* y que se reproducirán posteriormente según la programación efectuada.

2.1. COMUNICACIÓN ENTRE EL SERVIDOR Y LOS PLAYERS

2.1.1. Conectividad con Internet

La red DS expuesta en el apartado anterior está diseñada para ser escalable y flexible. Los *players* deben ser capaces de registrarse en la red DS de la manera más transparente posible para simplificar las tareas del gestor de red y del despliegue de los players. Para ello la red DS, cuando debe registrar un nuevo *player*, opera según el siguiente procedimiento:

El *player* toma la iniciativa de buscar acceso a Internet. Existen modelos de *players* con interfaz de red *Ethernet* 802.3 y otros modelos con interfaz Wi-Fi 802.11 y 3G. En este último caso, el dispositivo puede contar con diversas *Service Set IDentifier* (SSID) de 802.11 en memoria y busca aquella que esté disponible y cuyo AP sea detectado con mayor potencia. En caso de que la conexión Wi-Fi no sea posible y no exista ninguna red memorizada disponible, procede a utilizar la red 3G preconfigurada como conexión de *backup* ó de *fail-over.* En caso de que al cabo de un tiempo predeterminado se detecte alguna de las redes Wi-Fi configuradas y la conexión a la misma se establece con éxito, el dispositivo modifica el enrutamiento de las comunicaciones hacia Internet por la interfaz Wi-Fi desconectando la conexión de datos de la interfaz 3G.

Este comportamiento es común en las tareas de configuración remota de los dispositivos recién adquiridos por un cliente. Los *players* con capacidades Wi-Fi/3G se envían al lugar de instalación y se configuran remotamente a través de 3G. No obstante este sistema también se emplea en dispositivos móviles como vehículos: la conectividad 3G es vital cuando el vehículo se encuentra en movimiento y la conectividad Wi-Fi se emplea cuando el vehículo se encuentra estacionado para actualizar contenidos pesados.

2.1.2. Establecimiento de las conexiones tuneladas

Una vez que el *player* dispone de acceso a *Internet,* éste intenta registrarse en la red DS que tiene como punto neurálgico el servidor central. Cuando un *player* se registra en la red DS establece una primera conexión SSH con dicho servidor central. Para ello, los *players* vienen configurados de fábrica para que dispongan de un par de llaves público/privada únicas. De ese modo pueden realizar una conexión al servidor central con la que sólo tienen derecho a solicitar una terna de valores que corresponden con los puertos reservados en el servidor para las comunicaciones del tráfico con ese *player* determinado en ese momento concreto.

El servidor, en esa comunicación, elige y reserva una terna de puertos de forma aleatoria entre 1024 y 65535 que estén disponibles: sin usar y sin reservar. Una vez que el *player* dispone de estos puertos finaliza la conexión establecida y realiza una nueva en la que configura túneles reversos desde el servidor central hacia sí mismo tal y como se aprecia en la siguiente figura:

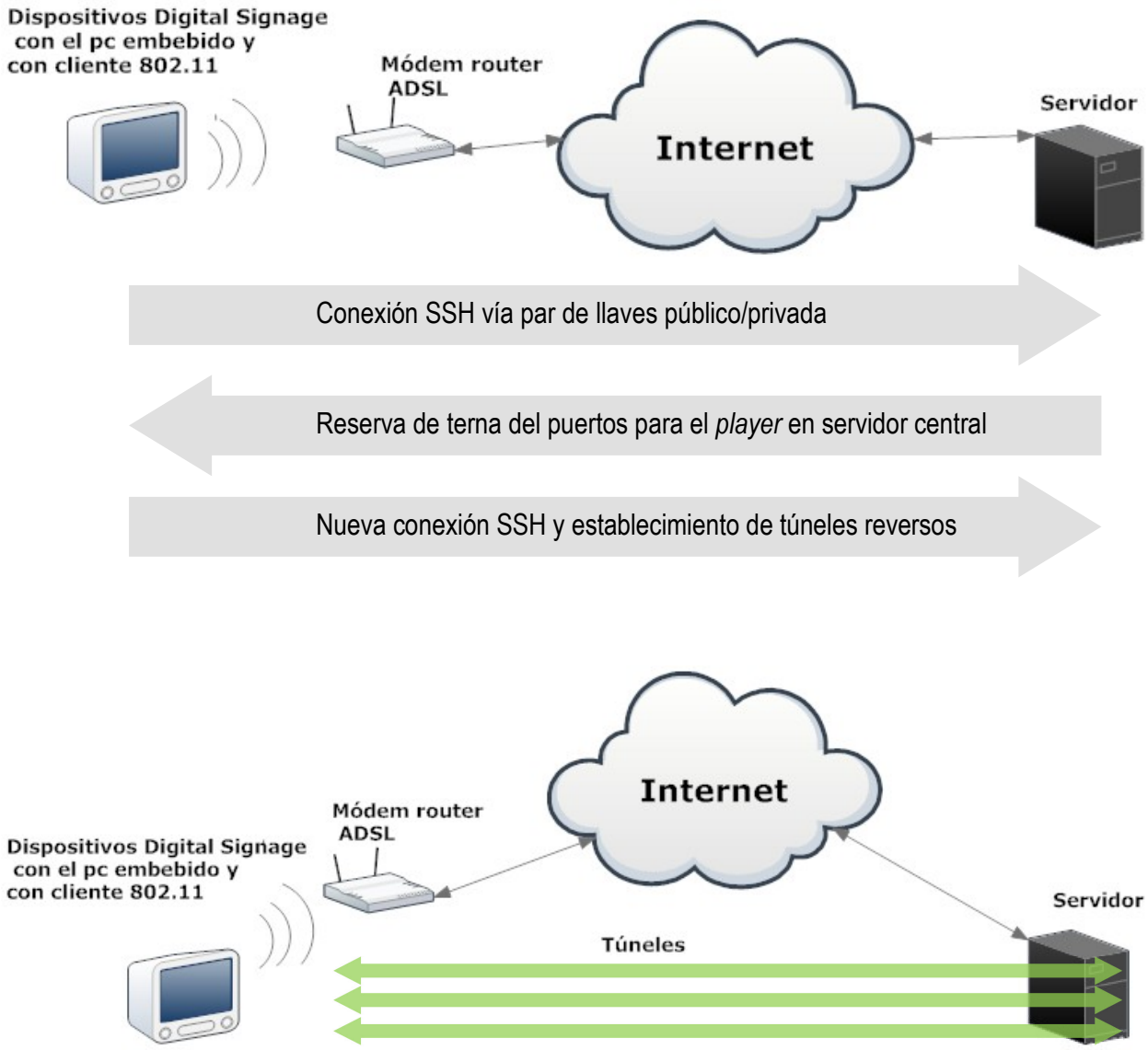

Figura 2-4: Etapas en el establecimiento de los túneles del *player* con el servidor

Tabla 3-1: Descripción de los túneles empleados en cada *player*

Número de túnel	Puerto destino en *player*	Puerto origen en Servidor	Función
1	22	x	Tráfico de control y monitorización
2	80	y	Tráfico web de gestión
3	2222	z	Tráfico de contenidos

Los puertos origen en el servidor se han denominado x, y, z, por ser valores enteros variables entre 1024 y 65535. Los túneles formados se emplean para transmitir la información relativa a la administración del *player* vía interfaz web, la monitorización del mismo por parte del servidor y la transmisión de contenidos diferidos. La asignación de estos puertos se guarda en una base de datos del servidor a la que tienen acceso las aplicaciones web de gestión de la red, los scripts del servidor relativos a la monitorización periódica de los *players* y las solicitudes de sincronización de contenidos por parte de las aplicaciones web de gestión.

2.1.3. Servidor de *Digital Signage*: Gestión de conexiones

En las redes DS pueden existir numerosos gestores de red con acceso a la monitorización de los parámetros de los *players*. Es común el utilizar consolas de monitorización de forma constante en pantallas para visualizar el estado de la red. Esta cantidad de información puede suponer un uso no despreciable del ancho de banda disponible para cada *player* y por tanto se ha optimizado su distribución a través del servidor.

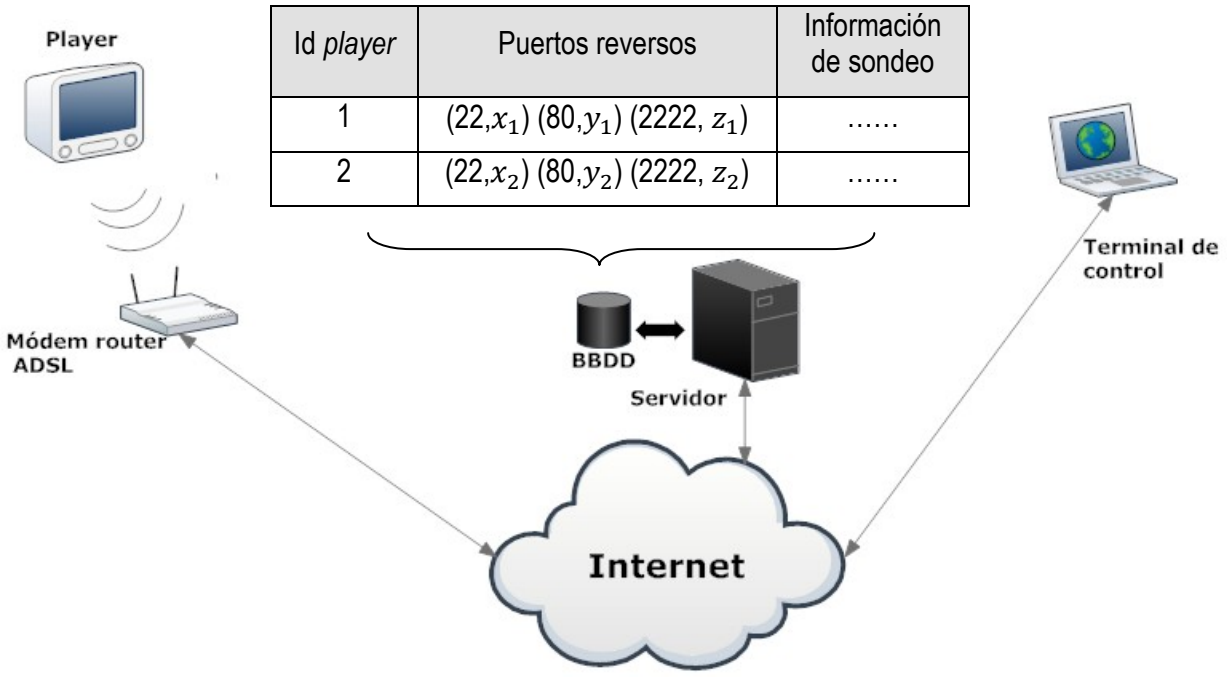

Figura 2-5: Comunicación del terminal de control con un *player* concreto a través del servidor

2.1.3.1. Monitorización de los players

Cada vez que un gestor de red se autentifica correctamente en el portal de gestión del servidor, éste marca con unos *flags* las entradas de los *players* correspondientes a monitorizar en la base de datos. De este modo, el servidor comienza a realizar sondeos periódicos a los *players* marcados y a registrar esa información en la base de datos de los *players*. La información de sondeo que se visualiza en el terminal de control es la guardada en la base de datos por el servidor, consiguiendo que, independientemente del número de terminales de control existentes, sólo se acceda una vez a cada *player* periódicamente.

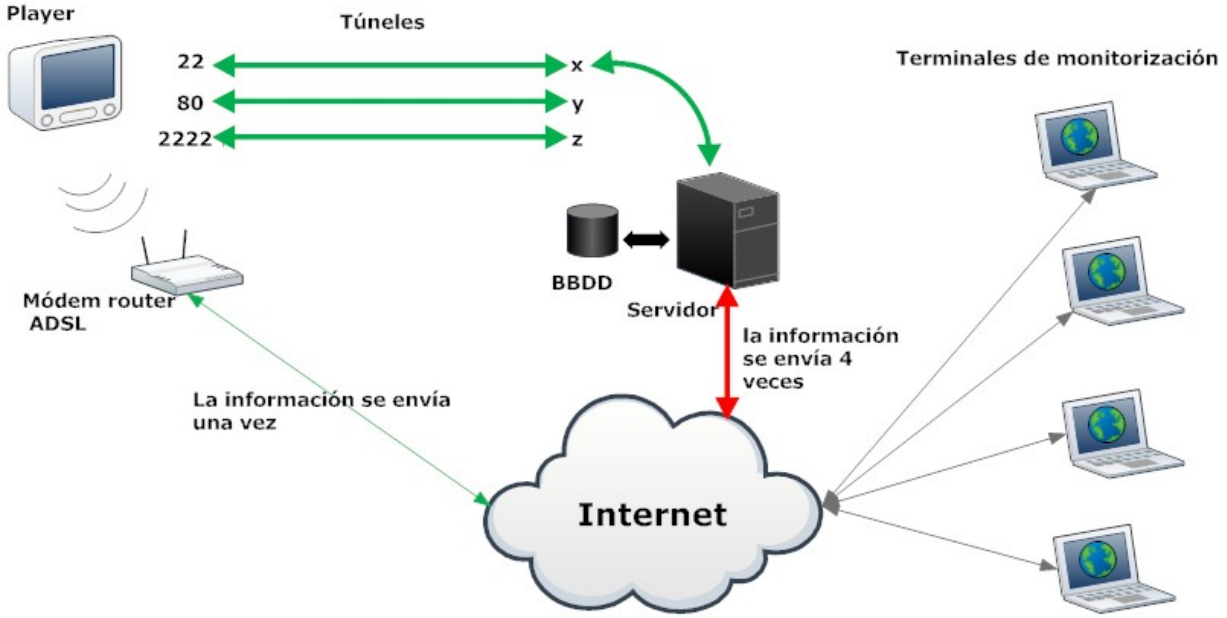

Figura 2-6: Ejemplo de monitorización de los parámetros básicos del *player* de forma remota

2.1.3.2. Gestión de los players

El acceso remoto para la gestión individual de cada *player* a través del servidor se realiza mediante la aplicación web de gestión local de que dispone[1]. Esta aplicación autentifica automáticamente al gestor de red en cualquier dispositivo que esté bajo su control pudiendo acceder a la página de gestión del aparato a través del túnel correspondiente.

El servidor busca en la base de datos el túnel asignado para gestión del *player* concreto y redirige la solicitud web de gestión a través de ese túnel mostrando la web de gestión en un *iframe* de la web del servidor. En caso de que el gestor de red no estuviese autentificado en el servidor e intentase acceder directamente al puerto correspondiente al túnel de gestión del *player*, la web de gestión del *player* no permitiría e[1]l acceso, pues el servidor no lo ha autentificado.

[1] Los *players* pueden tener un funcionamiento autónomo del resto de *players* y del servidor. Disponen de una página de gestión vía interfaz web que se puede localizar a través de protocolos Universal Plug and Play (UPnP*)* y *Simple Service Discovery Protocol (SSDP)* en la red local en la que está ubicado.

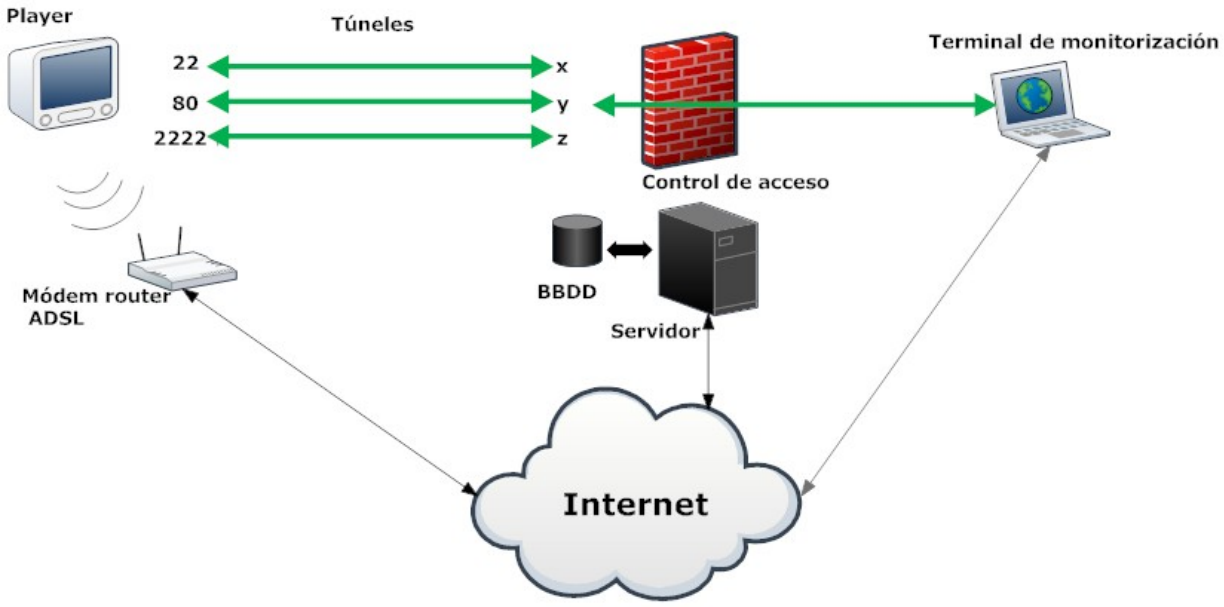

Figura 2-7: Ejemplo de gestión del *player* vía interfaz web de forma remota

A día de hoy se está trabajando en reforzar la seguridad programando en el propio servidor un firewall inteligente que permita el acceso a los puertos de un *player* en concreto según restricción de IPs y de gestores de red autentificados en la web de gestión del servidor, ofreciendo mayor seguridad. Sin este sistema, cualquier dispositivo puede acceder a la web de autentificación y servidor SSH de cualquier player por los puertos reversos.

La web de gestión del servidor se divide en dos áreas: La sección superior presenta el listado de los *players* a los que el gestor de red tiene acceso. Muestra información actualizada cada 7s (de izquierda a derecha) Id del *player*, alias, captura del contenido actualmente en pantalla, estatus del *player*, momento del último sondeo, lista de contenidos activa, contenido activo (número y *url*).

Figura 2-8: Captura de la interfaz web del servidor y de la interfaz de gestión de un player concreto

En la figura anterior, se ha seleccionado el *player* con *id* 26 para su gestión. En la parte inferior de la pantalla aparece la web de gestión del *player* a la que se accede a través del túnel que el servidor tiene preparado para ese *player* en concreto para tal efecto.

2.1.3.3. Distribución de contenidos

Otro aspecto fundamental en el rendimiento de la red DS, es la forma en la que los contenidos pesados son distribuidos por el gestor de red. Para aumentar la eficiencia de la red y facilitar la tarea de su distribución, el servidor central se puede encargar de la distribución de los contenidos, su validación y su actualización en caso de modificaciones. Para ello, el gestor de red, puede cargar contenidos en el servidor y luego asociarlos a *players* individualmente o a canales de contenidos para que posteriormente el servidor los distribuya. El servidor se encarga de verificar el estado de los contenidos en cada *player*, verificar la integridad de los mismos y preparar los paquetes de actualización para cada *player* con los archivos necesarios en cada caso. El funcionamiento es similar al de un repositorio de desarrollo pero en el que la iniciativa está en el servidor siempre y no en el cliente como suele ser tradicional. El servidor desarrolla la tarea de distribución de contenidos en *background* y puede monitorizarse si se desea a través de la interfaz web de administración del servidor.

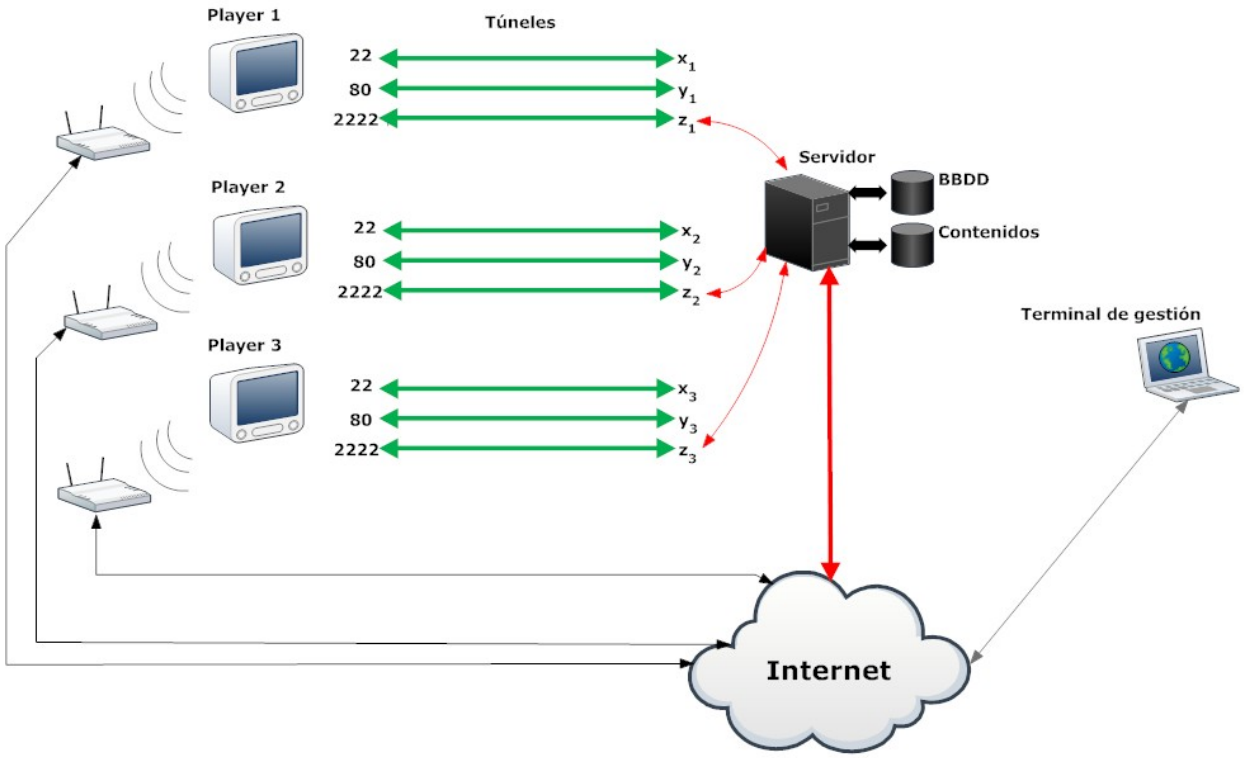

Figura 2-9: Ejemplo de distribución de contenidos de forma remota a través del servidor

Esta manera de distribuir contenidos permite al gestor de red reducir el tiempo necesario para llevar a cabo las tareas requeridas y además lo eximen de la monitorización del proceso de sincronización, ya que esta queda automatizada y coordinada por el servidor. De este modo si algún *player* se encuentra desconectado o recibe erróneamente la información relativa a los contenidos, el propio sistema lo tendrá en cuenta para solucionarlo autónomamente y de forma transparente.

QoS en la gestión de la red

3. QOS EN LA GESTIÓN DE LA RED

La arquitectura planteada para esta red DS ha intentado reducir los errores potenciales derivados del mal funcionamiento de la red, minimizar los problemas derivados de la configuración de los *routers* de acceso a Internet y simplificar la gestión de diversos *players* de forma simultánea. Con estas técnicas se ha mejorado notablemente la calidad percibida por el gestor de red frente a redes DS que no implementan estas medidas. Sin embargo existen aspectos intrínsecamente relacionados con el tráfico de datos que pueden mejorarse. A continuación se procede a un análisis de los factores que contribuyen a incrementar la latencia de las comunicaciones y entorpecer la gestión y funcionamiento de los dispositivos.

3.1. ASPECTOS MEJORABLES EN LA RED

Entre los factores que contribuyen a incrementar la latencia en las transmisiones existen diversos aspectos que han sido objeto de estudio y que, pese a no resultar determinantes, contribuyen a la reducción de las latencias experimentadas en las comunicaciones.

3.1.1. **Minimizar el tráfico de control**

La iniciativa de conexión a la red DS reside en el *player.* Por ese motivo, las tareas de monitorización y conexión a la red son realizadas por este dispositivo. Sin embargo, el gestor de red se conecta al servidor de DS y, por consiguiente, el servidor debe disponer de la información de estado de los *players* para poder ofrecérsela. Para ello el servidor realizará tareas de monitorización de los *players* con la que estar actualizado pero únicamente debe efectuar estos sondeos cuando el gestor de red se encuentra conectado.

La gestión local de la conexión recae en cada *player*. Cada elemento de la red, debe verificar la integridad de los túneles periódicamente para comprobar que los enlaces no se encuentran rotos y que la comunicación bidireccional es factible. Este sistema de verificación se produce a intervalos regulares no excesivamente seguidos para no sobrecargar en exceso el enlace, pero tampoco extremadamente espaciados, pues se correría el riesgo de dejar incomunicado el *player* durante ese tiempo.

En entornos no amigables donde el ancho de banda se encuentra muy restringido (Sistemas inalámbricos 3G/2G con cobertura EDGE o GPRS) este problema se acentúa pudiéndose presentar el caso en el que al encontrarse copado el enlace, se producen escenarios de congestión que impiden al algoritmo verificar el estado de los túneles. En consecuencia el *player* desconecta los túneles con el servidor y abre unos nuevos, interrumpiendo las comunicaciones en curso; pues considera que los túneles no funcionan cuando lo que realmente ocurre es un estado de congestión.

Para solucionar este problema, se incluyen una serie de comprobaciones pasivas sobre los enlaces que permitan verificar el correcto funcionamiento de los túneles sin necesidad de inyectar tráfico extra:

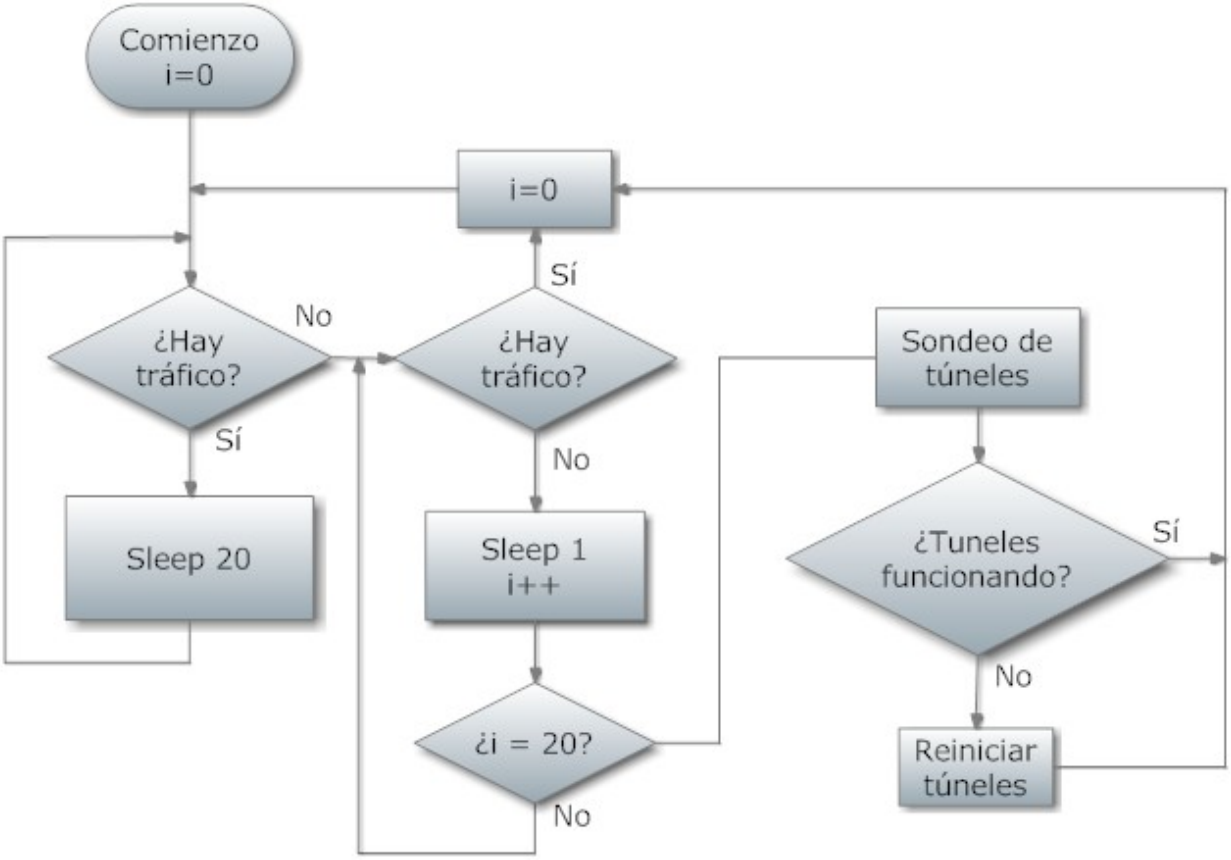

Figura 3-1: Diagrama de flujo para la comprobación de los túneles de comunicación

Básicamente el algoritmo se reduce a verificar si hay tráfico por los túneles mediante un *sniffer* basado en *tcpdump*. Las comprobaciones se hacen cada 20 segundos, y si no se detecta tráfico pasan a ser cada segundo. Si transcurren 20 comprobaciones seguidas sin tráfico se realiza un chequeo de conectividad de los túneles. De este modo sólo cuando no hay tráfico se hacen estas verificaciones, evitando sobrecargar los túneles con información de control, ya que el hecho de que exista tráfico por los túneles es indicativo de su correcto funcionamiento.

3.1.2. Sistema de distribución de contenidos en diferido

La optimización del procedimiento de distribución de contenidos supone otro aspecto fundamental que favorece un mejor funcionamiento de la red DS, pues uno de sus principales cometidos es el transporte de contenidos hacia los *players*.

El sistema de distribución empleado tiene muchas semejanzas con los repositorios de contenidos estándar (*svn*, *git*, etc). La diferencia con ellos radica en que el control de la operación de sincronización se realiza desde el servidor. Es el servidor el que toma la iniciativa y controla las etapas del proceso y principalmente por ese motivo se desarrolló un sistema específico para este escenario. El esquema de funcionamiento en una sincronización podría ser el siguiente:

Figura 3-2: Etapas en la actualización de los contenidos

La sincronización como tal se aplica a un canal de contenidos, es decir, un conjunto de carpetas y ficheros. La actualización comienza con la puesta al día del listado de ficheros y sus atributos que existen en ese canal de contenidos en el servidor. Esa lista de ficheros contiene, por cada fichero, un md5 que sirve para identificarlo.

Los *players*, a su vez, disponen de una lista de los ficheros existentes de ese canal en su memoria. Aquellos que van a realizar la sincronización actualizan también su listado de ficheros tal y como hace el servidor.

Una vez que el *player* acaba su tarea, el servidor solicita su listado de ficheros y lo compara con el que tiene. De ahí extrae la siguiente información:

- **Archivos a empaquetar para mandar:**
  Ficheros nuevos, no existentes en el *player* o cuyo md5 haya cambiado.

- **Archivos a copiar entre carpetas del *player:***
  Fichero nuevo con resumen md5 idéntico a otro ya existente.

- **Archivos a mover o Renombrar entre carpetas del *player:***
  Fichero nuevo con md5 idéntico a otro anterior que ya no existe.

- **Archivos a borrar:**
  Ficheros que ya no existen en el canal del servidor y todavía existen en el repositorio del *player* y no se ha aplicado en ellos la operación de mover o Renombrar.

El servidor empaqueta todos los ficheros necesarios y la lista de comandos a ejecutar a posteriori y lo manda al *player* concreto para el que lo ha procesado. De ese modo, las comunicaciones entre el servidor y cada *player* reducen drásticamente su carga.

Este procedimiento de actualización, sin embargo, puede resultar pesado en términos computacionales debido a las operaciones MD5 que involucran. Por ese motivo, existe la versión simplificada de sincronización en formato incremental en la que sólo se empaquetan los archivos nuevos y modificados prescindiendo de las comprobaciones MD5. Esto permite actualizaciones periódicas cada poco tiempo (15 min) pero no garantiza la integridad de los ficheros de los canales en los *players* totalmente. Si algún *player* se encuentra desconectado o tiene problemas en la actualización, perderá los datos. Lo mismo ocurre si algún usuario de red modifica o borra accidentalmente ficheros que formen parte del canal. Para evitar esto, siempre que se usan actualizaciones incrementales se programa al menos una vez al día una actualización integral que sincronice y valide todos los ficheros que forman parte del canal de contenidos a sincronizar.

3.1.3. Diferenciación del tráfico

Existen diversos problemas para mejorar el grado de QoS E2E en el escenario propuesto *player*-servidor. La conexión del *player* con *Internet* se suele realizar a través de una red local en la que se encadenan distintos enlaces:

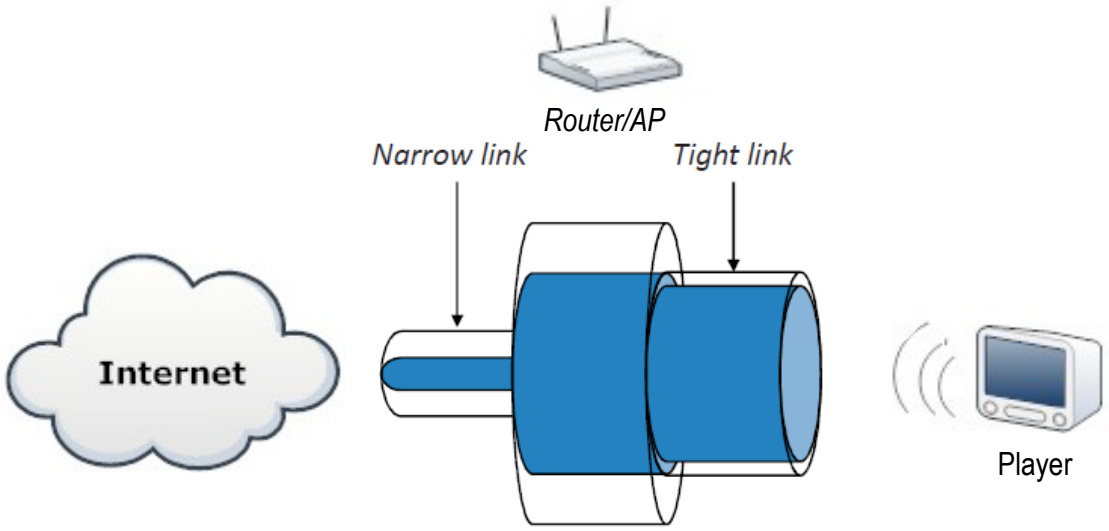

Figura 3-3: Problemática típica en enlaces encadenados tras los que ofrecer QoS E2E

"*Narrow link*" representa el enlace que fijará el límite máximo de ancho de banda disponible, mientras que "*Tight link*", fijará la máxima tasa binaria posible. En un escenario real, "*Narrow link"* puede estar representado por el enlace ADSL, mientras que "*Tight link*" puede representar el enlace WI-FI. Teniendo esta estructura en cuenta, podemos deducir que el ancho de banda máximo disponible para el enlace vendrá limitado por el ADSL. Podría ocurrir que el ancho de banda del servidor central con *Internet* fuera insuficiente, en cuyo caso, la suposición anterior no sería válida. Sin embargo, al ser un servidor dedicado y disponer de un *Service Level Agreement* (SLA) con la empresa concesionaria del mismo, se puede asegurar la disponibilidad de ancho de banda en ese extremo y tener monitorizado su consumo.

Para poder ofrecer QoS E2E en este ámbito debería existir, por tanto, algún tipo de ingeniería de tráfico en el *router* ADSL que realizase reserva de ancho de banda o *traffic shapping* sobre el tráfico cursado por el "*Narrow Link*", y esta posibilidad no resulta viable en general ya que sólo se dispone de acceso al *player* de *Digital Signage*. Sin embargo, sí es factible en redes planificadas de Digital Signage a desplegar en las que se pueda efectuar una planificación previa.

Por otro lado, se ha planteado la posibilidad de gestionar el tráfico cursado por el *player* desde el propio *player* con técnicas de planificación de tráfico. Sin embargo, para que funcionasen de forma efectiva, habría que calcular periódicamente el ancho de banda disponible E2E, pues en todas ellas se usa como parámetro. Obtener ese dato es complicado dado que el "*Narrow link*" no conecta directamente con el *player*, existiendo elementos intermedios de red con buffers de entrada de paquetes que dificultan la tarea de medir el ancho de banda extremo a extremo con un mínimo *overhead* [1]. Por consiguiente, en este escenario, resulta muy difícil de medir el ancho de banda E2E con cierta precisión, rapidez y manteniendo la fiabilidad frente a cambios en la red quedando esta alternativa descartada.

3.1.4. Optimización del protocolo SSH en comunicaciones tuneladas

El sistema de conexión de los *players* con el servidor que se ha planteado usa un sistema de túneles que utiliza TCP sobre TCP. Esta forma de tunelado tiene un comportamiento aceptable únicamente mientras exista suficiente ancho de banda en exceso en el conjunto de redes por las que circula el túnel. De no ser así, los temporizadores TCP pueden explicar y el rendimiento disminuye notablemente debido a las retransmisiones produciéndose lo que se conoce como "*tcp meltdown problem*" [2].

Para solucionar este problema, se propone la implementación de una versión modificada ya existente de OpenSSH en la que la tunelación se realizará mediante el protocolo UDP [3].

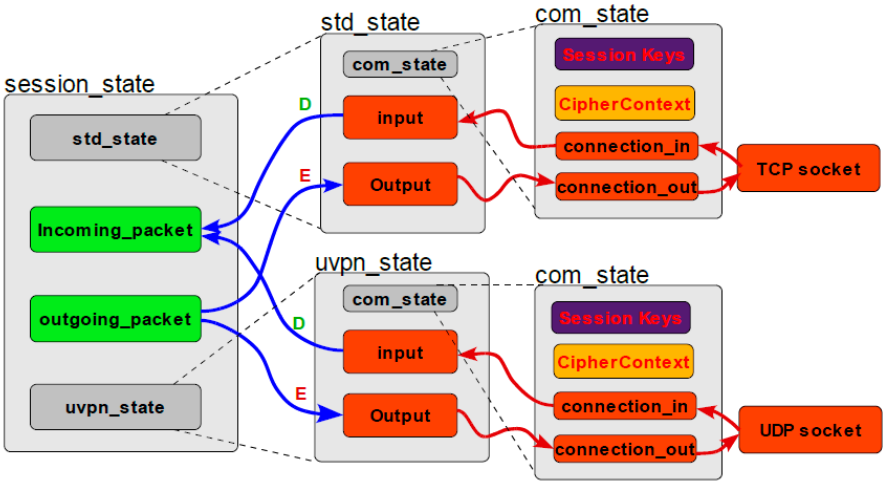

Figura 3-4: Modificaciones propuestas a OpenSSH para gestionar los túneles mediante sockets UDP

De este modo, una estructura de túneles TCP sobre UDP evita las ineficiencias derivadas de las pérdidas de *Acknoledgements* (ACKs) al usar TCP sobre TCP

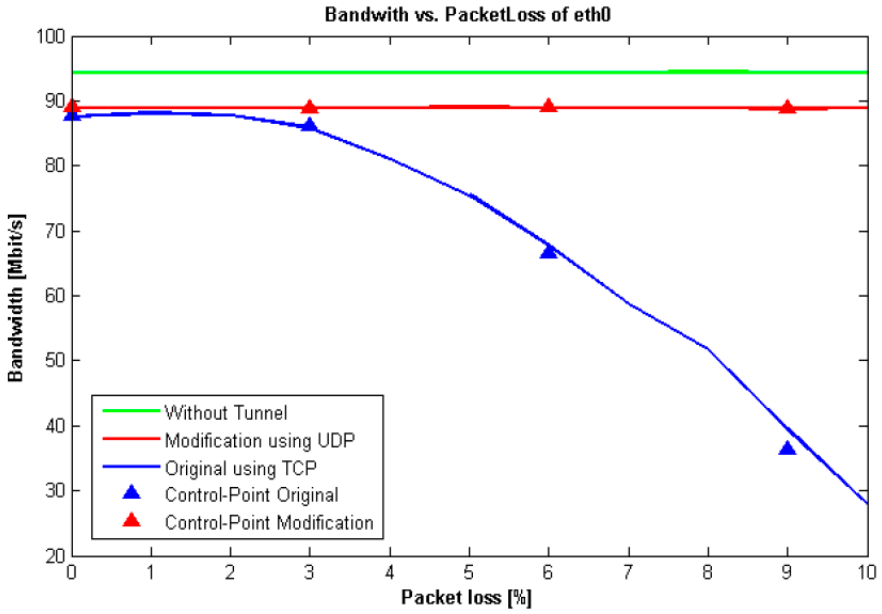

Figura 3-5: Resultado tras emular pérdidas de paquetes ACKs [3]

La implementación de esta versión modificada de OpenSSH no se aplicará inmediatamente, ya que pese a estar desarrollada en la actualidad, es necesario someterla a una batería de pruebas que certifiquen su correcto funcionamiento en los escenarios más comunes para intentar no comprometer el funcionamiento de la red DS. Por ello se propone además no prescindir de la implementación actual de OpenSSH y añadir esta nueva versión permitiendo utilizar una u otra según convenga.

3.2. MEJORA DE LA LATENCIA MEDIANTE EL CONTROL DE CONGESTIÓN

Todos los aspectos expuestos en el apartado anterior tratan por separado de minimizar la congestión en la red de diversas maneras, pero ninguna resulta suficiente. Para solucionar los problemas derivados de la congestión se debe gestionar la congestión de forma más global desde los protocolos de comunicación.

Para regular el tráfico generado por el *player* y evitar que los flujos de datos de menor relevancia dificulten las operaciones de control y/o gestión, se profundiza en la forma que SSH gestiona la transmisión de información. Para ello, hay que tener en cuenta que SSH versión 2 multiplexa los canales de cada conexión de la sesión [4] evitando enviar más datos relativos a un canal concreto hasta que no se indica explícitamente un mensaje de ventana disponible para ese canal. Con esta medida, se evita, por ejemplo, que una transmisión SFTP se adueñe del ancho de banda disponible en uno de los sentidos de las comunicaciones y permite a las aplicaciones de gestión y control competir por el acceso al ancho de banda.

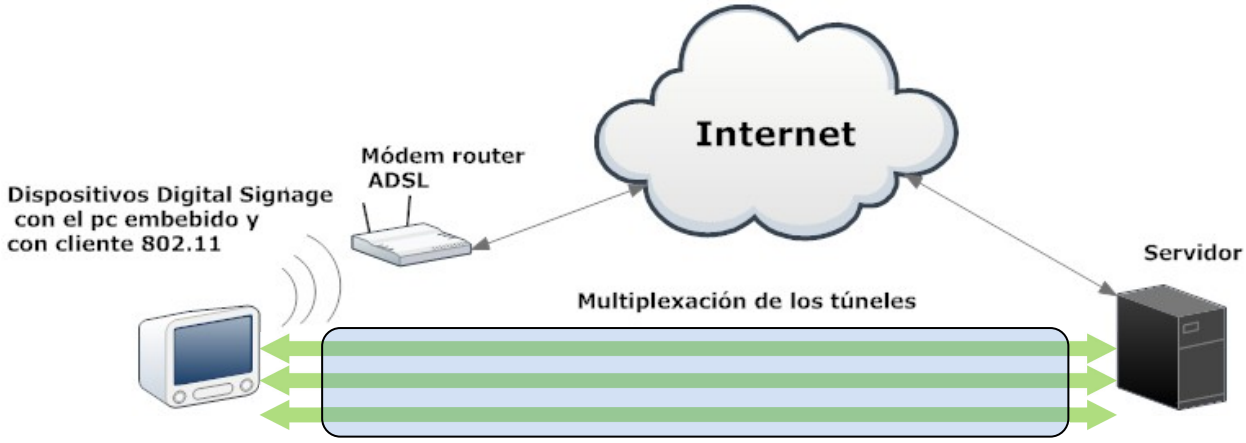

Figura 3-6: Establecimiento tradicional de los tres túneles en sesión única mediante multiplexación. Se plantea cambiar este esquema por sesiones independientes.

Esta forma de gestionar las comunicaciones en las sesiones SSH, beneficia al conjunto de las restantes transmisiones del dispositivo pero pueden afectar al resto de conexiones multiplexadas en esa misma sesión SSH. Por ello, parece recomendable establecer cada túnel en una sesión SSH independiente. De este modo, al utilizar tres sesiones SSH diferentes, se aplica el control de congestión de cada una de ellas reduciendo el efecto del tráfico pesado sobre el tráfico de control y gestión.

Para validar la efectividad de esta medida se propone medir el Round-*Trip delay Time* (RTT) de los túneles en ambos escenarios: Compartiendo sesión y estando multiplexado con una transmisión SFTP y encontrándose en sesiones SSH independientes. El RTT supone un indicador del grado de latencia experimentada en las aplicaciones web de control y gestión de la red DS.

3.2.1. Herramientas para medir el *Round-Trip delay Time* (RTT)

El RTT representa el tiempo que tarda un paquete de datos en viajar desde el emisor hasta el receptor y volver su ACK al emisor. Esta medida sirve para analizar determinados aspectos de la red permitiendo extraer conclusiones relativas al rendimiento de la red y su funcionamiento si se tiene en cuenta correctamente las condiciones en las que se realizan estas medidas.

Existen numerosas herramientas disponibles, pero sólo deben emplearse aquellas que resulten adecuadas para evaluar el RTT de una conexión tunelada. A continuación se exponen algunas de las herramientas analizadas.

3.2.1.1. Dtrace

Esta herramienta se define como un pseudo lenguaje de programación que se utiliza para monitorizar el estado del funcionamiento de los procesos internos del sistema operativo, permitiendo realizar modificaciones en ellos y facilitar la búsqueda de problemas. Dtrace hereda la sintaxis de C, complementándola con funciones y variables específicas para facilitar la depuración.

Dtrace dispone de un módulo que se carga con el *kernel* y con el que se comunica un comando que puede ejecutar *scripts dtrace*. Esta herramienta permite modificar dinámicamente los procesos del *kernel* del sistema para obtener información adicional de interés llamadas *probes*. Una *probe* consiste en una dirección de memoria o actividad del sistema a la que *dtrace* puede incorporarse para efectuar una serie de acciones, como obtener un volcado de *stack*, una marca de tiempo o un argumento de función. Las *probes* pueden entenderse como sensores que disparan sucesos que se encargan de recopilar la información de interés. Estas funciones se programan en los scripts y se relacionan a una *probe* específica para dispararlos. Los scripts son compilados en el momento de su ejecución y cargados cuando se dispara alguna de las *probes*

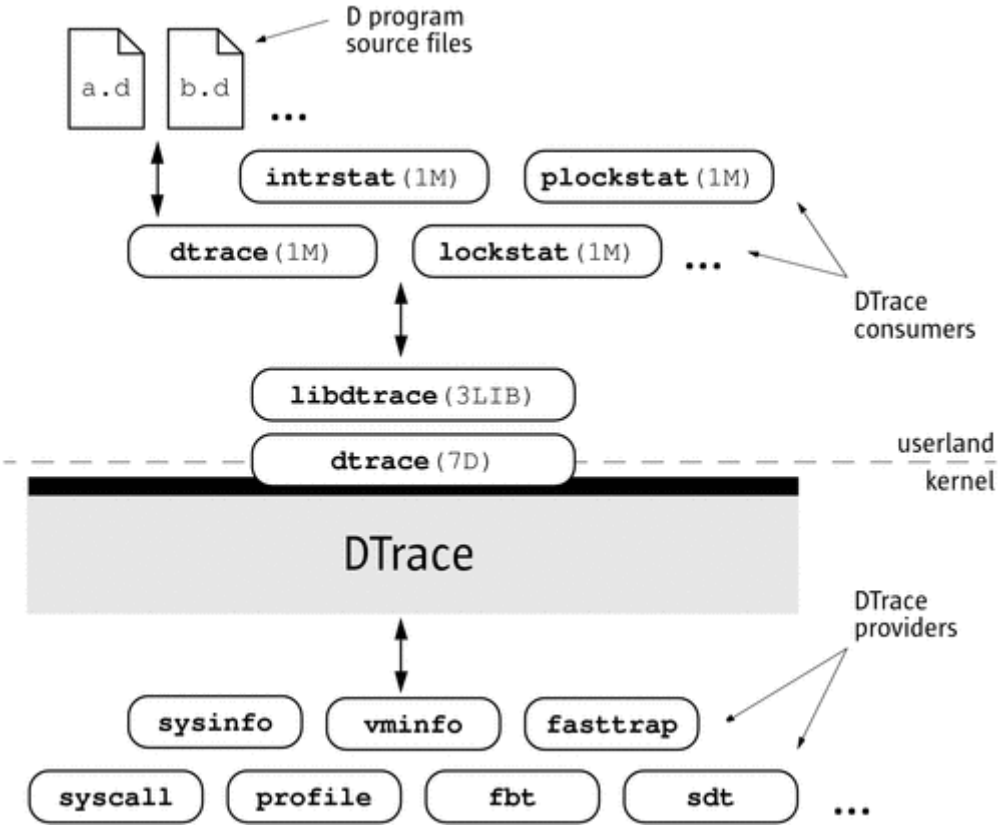

Figura 3-7: Arquitectura general de Dtrace [5]

Esta herramienta podría utilizarse para realizar mediciones de parámetros relacionados con el funcionamiento de TCP e IP en las comunicaciones entre el *player* y el servidor. Sin embargo, la versión de *dtrace* para *Linux* todavía sólo ofrece funcionalidad limitada. La herramienta es originaria de Solaris y para conseguir hacerla funcionar en *Linux* se deben seguir instrucciones específicas para generar su módulo del *kernel* [5][6]

Tras reiterados intentos no pudo instalarse en el *player*, ya que *dtrace* parece ser muy dependiente de la versión de *kernel* empleada. Una posible alternativa consistiría en instalar una máquina virtual Solaris y usar su dtrace, pero realizar eso en el *player* es complejo y por consiguiente esta herramienta se descarta para este propósito.

3.2.1.2. Hping3

Otra herramienta utilizada para medir el RTT es *Hping3*[7], pero tampoco resulta útil para este caso en concreto en el que se tiene que medir la latencia E2E de un túnel. La aplicación dispone de opciones para hacer ping a puertos utilizando los *flags* de TCP como el *Sync* para forzar una respuesta. Esta filosofía funciona para medir el RTT cuando se realiza de máquina a máquina, pero al invocar el comando contra el puerto de un túnel SSH, *Hping* se comunica con el *socket* que SSH prepara localmente para ese túnel de ese mismo extremo y por tanto no se mide el RTT, puesto que eso debería efectuarse en el socket del otro extremo del túnel y no es posible.

3.2.1.3. Tcpdump y tcptrace

Combinando *tcpdump y tcptrace*[8] se pueden utilizar para efectuar volcados de tráfico y analizar las trazas para interpretar los paquetes enviados y recibidos y obtener datos estadísticos de las latencias. Sin embargo, dado que las comunicaciones por el túnel se encuentran encriptadas, no resulta inmediata la medición del tráfico cursado por los túneles, teniendo que recurrir a interfaces *loop-back* puente que también deben monitorizarse. *Wireshark* puede obtener datos similares pero experimenta el mismo problema.

3.2.1.4. Medida directa del RTT sobre mensajes echo tunelados

Para realizar una medición RTT sobre los túneles se propone realizar las medidas de RTT sobre mensajes mandados a través del túnel al puerto *echo* del servidor destino.

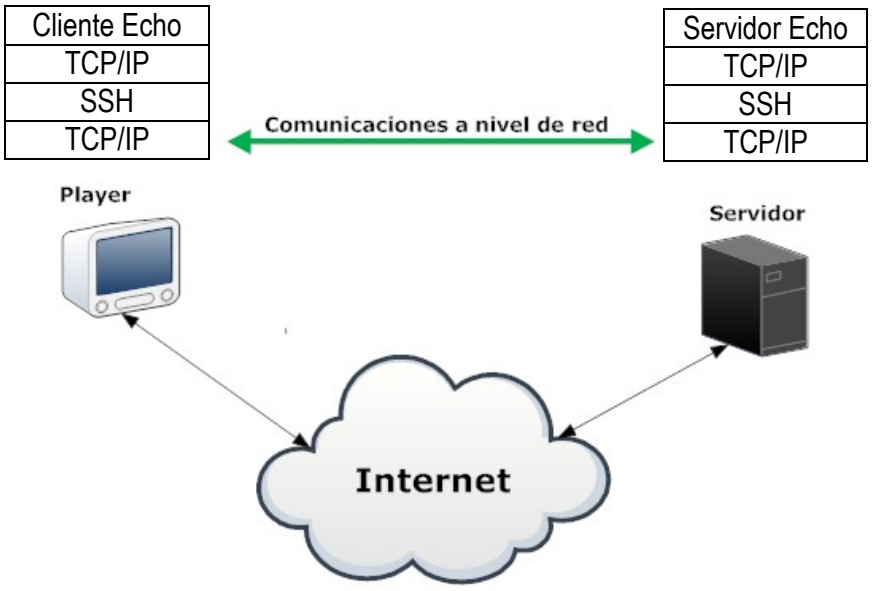

Figura 3-8: Escenario de funcionamiento cliente/servidor *echo* para medir RTTs a través de túneles SSH

Esta medida registra el RTT percibido por el sistema de gestión o el administrador de forma más realista, pues se lleva a cabo a nivel de aplicación. Para ello se emplea un cliente *echo* que se pueda tunelar por TCP*:*

```
#!/usr/bin/env python


import socket
host = '127.0.0.1'
port = 7777
size = 1024
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
s.send('Hola, mundo')
data = s.recv(size)
s.close()
print 'Recibido:', data
```

Las medidas de RTT se pueden obtener a partir del comando *time* ejecutado sobre el cliente echo generando un archivo de resultados tal y como se efectúa en el apartado 3.2.2.3 de esta memoria.

3.2.2. Impacto en la latencia del tráfico debido a los túneles que comparten sesión por multiplexación.

El objetivo de estas medidas es el de cuantificar el efecto en la latencia del tráfico de gestión y control que tiene en esta red el establecimiento de túneles multiplexados en una misma sesión en la que existe un túnel con tráfico pesado de datos. Todas las comunicaciones entre cada *player* y el servidor central se establecen en túneles que comparten una única sesión por *player*. Sin embargo, según lo expuesto en el apartado 3.2, puede mejorarse la latencia usando túneles con sesiones independientes debido a que el control de congestión que aplica SSH se efectúa por sesiones y no por túneles. Para validar este hecho se realizan algunas medidas que contrastan datos de latencia de túneles SSH cuya sesión se comparte con una transferencia masiva SFTP frente a otro túnel que, simultáneamente se monitoriza y cuya sesión SSH es independiente de la anterior.

3.2.2.1. Metodología

El procedimiento a seguir para poder cuantificar la latencia extra experimentada por un túnel SSH multiplexado con otro con trafico SFTP pesado frente a un túnel sin multiplexar en sesión SSH independiente consiste en evaluar el RTT del túnel independiente frente al RTT del túnel multiplexado con la transferencia SFTP. Los túneles se establecen desde un *player* de la red hasta el servidor DS en Montreal tal y como se expone en la siguiente tabla:

Tabla 3-1: Configuración de los túneles empleados en las pruebas experimentales

Puerto origen en *player*	Puerto destino en Servidor	Función	Comando ejecutado para sondear
7777	7	Medición de RTT a través de un túnel independiente	Cliente *echo*: echo_tunel_simple.py
7778	7	Medición de RTT a través de un túnel multiplexado con otro con tráfico pesado	Cliente *echo*: echo_tunel_multiple.py
2222	22	Túnel con tráfico pesado	SFTP

La única diferencia entre los dos clientes *echo* es el puerto al que lanzan sus peticiones de eco.

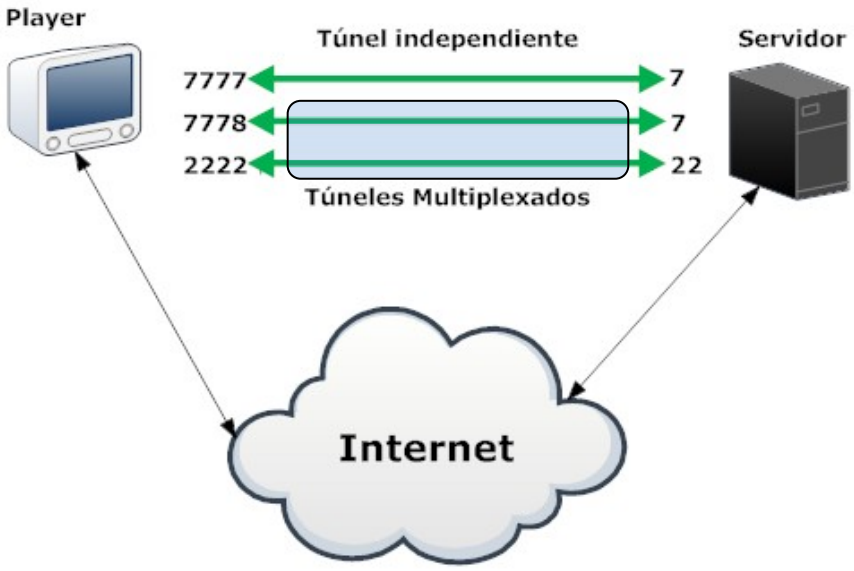

Figura 3-9: Escenario de pruebas propuesto para el análisis del impacto de tráfico pesado en túneles SSH

Se establecen tres comunicaciones tuneladas. Por dos de ellas se establece una comunicación SSH con el servidor y por la tercera un túnel SFTP. El túnel con comunicación SFTP comparte sesión SSH con una de las otras comunicaciones para poder estudiar el impacto del tráfico SFTP masivo en el túnel que comparte sesión frente al túnel de sesión independiente.

Una vez establecidos los túneles se ejecutarán de forma sincronizada sondeos *echo* a través del túnel independiente y del túnel multiplexado en varios escenarios: En primer lugar, se aplicarán estos sondeos en el túnel SSH independiente sin enviar datos por la conexión SFTP. De este modo se obtienen los valores de RTT de referencia en el mejor escenario posible: Sólo una comunicación SSH y la comunicación SFTP inactiva.

Posteriormente, se procede a realizar medidas de RTT con la comunicación SFTP a plena carga. Para ello se efectúan sondeos simultáneamente en ambos túneles SSH y se representan los valores RTT obtenidos en un gráfico dónde se podrán comparar las diferencias de latencia.

Finalmente se modifican ciertos parámetros de las comunicaciones para controlar la congestión y reducir la latencia. Se realizan sondeos en este escenario y se valoran los resultados obtenidos para exponer unas conclusiones.

3.2.2.2. Creación de los túneles

Para establecer los túneles se ejecutan los siguientes comandos:

```
ssh -L7777:127.0.0.1:7 ateire.com
ssh -L2222:127.0.0.1:22 -L7778:127.0.0.1:7 ateire.com
```

En la primera línea se crea un túnel simple, con puerto local en el *player* 7777 que conduce al puerto remoto *echo* (7) del servidor central. En la segunda se establece una sesión con dos túneles, el que tiene puerto local 2222 se usará para transmitir vía SFTP datos constantemente al servidor central e intentar ocupar todo el ancho de banda E2E disponible. El puerto 7778 local también se comunica con el puerto *echo* del servidor central.

3.2.2.3. Ejecución de los clientes echo para obtener RTTs

Las pruebas siguientes se realizan para comprobar si se perjudica la latencia del tráfico en un túnel que comparte sesión SSH *OpenSSH* v2 con otro que requiere un uso intensivo del ancho de banda. Para ello se habilita sobre el túnel de datos (2222) una sesión SFTP a través de la cual se transmite una ráfaga de datos constantemente a la velocidad que permite el escenario propuesto.

```
sftp -oPort=2222  127.0.0.1
```

Posteriormente se ejecutan dos clientes *echo*: Uno contra el puerto local 7777 (echo_tunel_simple.py) y otro contra el puerto local 7778 (echo_tunel_multiple.py) mediante un *script sh* que se encarga de ejecutarlos de forma sincronizada recursivamente y apuntando en un log los tiempos necesitados por cada cliente *echo* para ejecutarse. De este modo, el script *sh* genera las llamadas a los scripts *python* síncronamente y obteniendo medidas de tiempo que pueden compararse para poder deducir el tiempo extra del RTT según el túnel empleado.

sondeos_echo.sh

```
#!/bin/sh
ahora=`date +%s%N | cut -b1-13`
echo "" >echo_tunel_simple.log
echo "" >echo_tunel_multiple.log

function sondeo()
{
	tipo=$1
	momento=`date +%s%N | cut -b1-13`
	salida=`(time $tipo".py") 2>&1`
	rtt=`echo $salida | grep real | cut -f 2 -d 'm' | cut -f 1 -d 's'`
	transcurrido=$(($momento-$ahora))
	t=`awk 'BEGIN{printf("%0.3f", '$transcurrido' / '1000' )}'`
	echo $t $rtt >>$tipo".log"
}

for ((i = 0; i < 250; i++))
do

	sondeo echo_tunel_simple&
	pid_sondeo_tunel_simple=$!

	sondeo echo_tunel_multiple&
	pid_sondeo_tunel_multiple=$!

	wait $pid_sondeo_tunel_simple
	wait $pid_sondeo_tunel_multiple
done
```

Este script genera dos *logs*: “echo_tunel_simple.log” y “echo_tunel_multiple.log” con dos columnas de cifras. La primera indica el tiempo en milisegundos transcurrido desde que comienza la prueba y la segunda indica el RTT en milisegundos de cada petición. Los datos obtenidos son representados con *Gnuplot*. La ubicación geográfica del cliente y del servidor en estas pruebas es importante. El servidor está en Montreal, Canadá y el cliente en Zaragoza. Además debe tenerse en cuenta que los valores RTT obtenidos no representan exclusivamente la latencia del túnel, ya que la ejecución de los scripts también contribuye:

Tabla 3-2: Valores de referencia de latencias obtenidos como promedio de 16 medidas espaciadas a lo largo de las pruebas experimentales.

RTT Ping desde Zaragoza hasta Montreal	125 ms en promedio
Ejecución del cliente y su medición del tiempo empleado en una conexión no tunelada	260 ms en promedio
Ejecución del cliente conectándose a servidor *echo* en local	105 ms en promedio

3.2.2.4. Comparativa del RTT entre un túnel de sesión única y otro multiplexado con una conexión SFTP

En primer lugar se obtiene la medida de RTT en ausencia de tráfico en otros túneles.

Comandos GNUPlot:

```
set encoding iso_8859_1
set title "Comparativa en los RTT's de SSH"
set ylabel "RTT en segundos"
set xlabel "t(s)"
set xrange [0:90]
plot "echo_tunel_simple_sin_carga.log" with lines title "RTT del tunel sin carga”
```

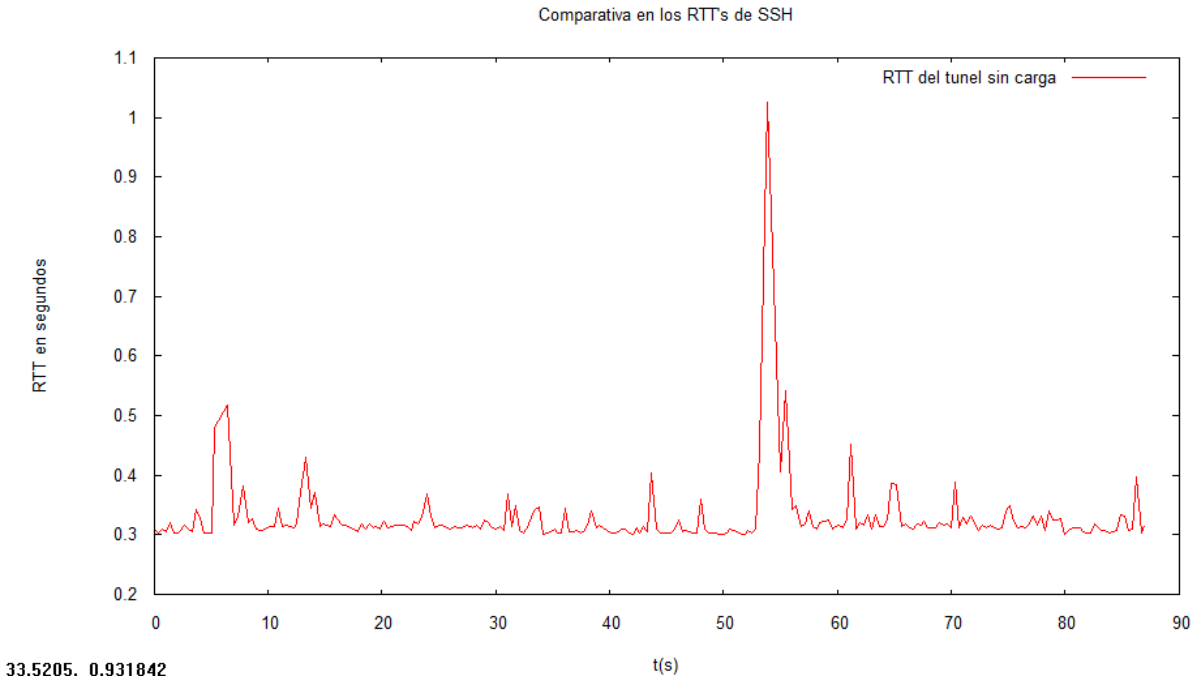

Figura 3-10: RTTs de un túnel SSH sin tráfico extra en otros túneles que comparten sesión

Tabla 3-3: Datos estadísticos de RTT de un túnel SSH en ausencia de carga en la conexión SFTP

	RTT Medio (ms)	Desviación estándar (ms)
túnel de sesión única	324	54

Si observamos la Tabla 3-2 se observa una sobrecarga de 64 ms debido a las conexiones tuneladas. Esta sobrecarga aumentará en caso de pérdida de paquetes debido a la apilación *TCP/IP* sobre *TCP/IP* que tiene lugar en el túnel y que se pone de manifiesto en el segundo 54 donde ocurre un incremento de latencia notable.

Los datos referentes al escenario en el que el segundo túnel comparte sesión con una transferencia SFTP se representan del siguiente modo:

Comandos GNUPlot:

```
set encoding iso_8859_1
set title "Comparativa en los RTT's de SSH"
set ylabel "RTT en segundos"
set xlabel "t(s)"
set xrange [0:150]
set yrange [0:2]
plot "echo_tunel_simple.log" with lines title "RTT en tunel de sesi\363n \372nica", \
"echo_tunel_multiple.log" with lines title "RTT en tunel de sesi\363n compartida"
```

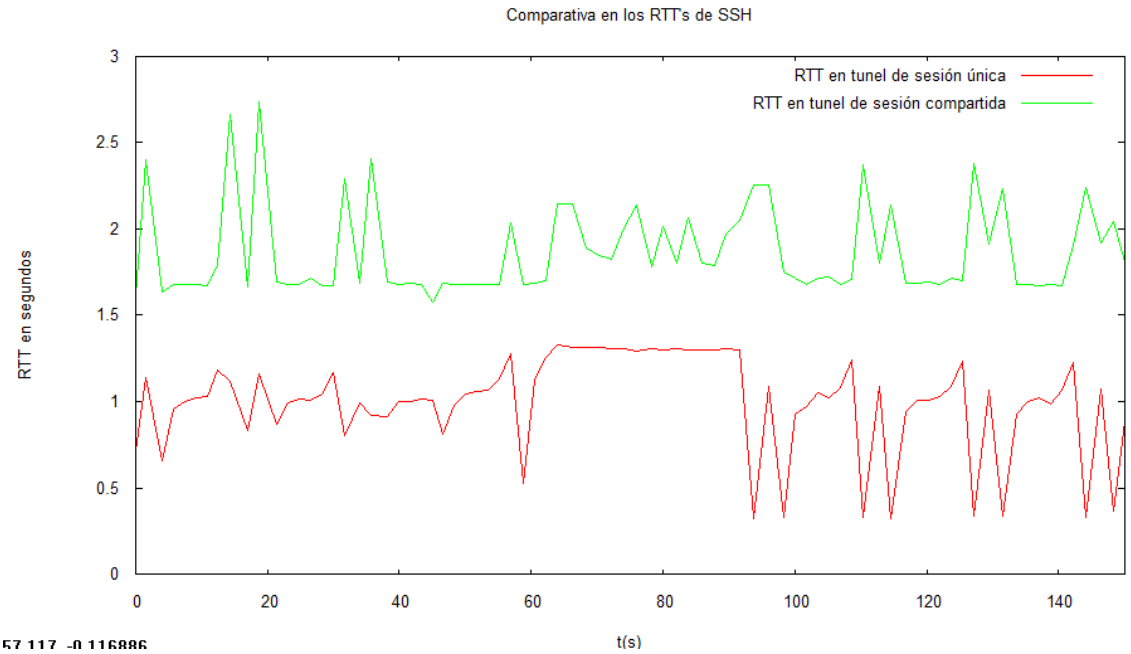

Figura 3-11: Comparativa de RTT's entre túneles SSH de sesión única y con sesión compartida.

Tabla 3-4: Datos estadísticos de RTT en túneles SSH con tráfico en túneles que comparten sesión

	RTT Medio (ms)	Desviación estándar (ms)
túnel de sesión compartida	1909	330
túnel de sesión única	1041	295

En este escenario, la tasa de transferencia de SFTP es constante y alrededor de los 600 *kbytes*/s.

De los resultados anteriores se deduce que la latencia en el túnel de sesión compartida es consecuencia de un estado de congestión de los buffers intermedios. La sesión SSH frena las peticiones de eco del túnel que comparte sesión con el de datos SFTP y sufre una latencia mayor respecto al túnel con sesión independiente debido a que la sesión SSH aplica control de congestión sobre todos los canales abiertos. Por ese motivo, el túnel SSH que va independiente dispone de más facilidad para transmitir y sufre latencias menores ya que no tiene que soportar el *buffer* del otro túnel de la transferencia SFTP y lo

evita. En este escenario queda por tanto patente que resulta conveniente el uso de sesiones individuales para cada túnel.

3.2.2.5. Ajuste fino del tamaño de buffer de SFTP

Para reducir el efecto de la latencia al máximo se propone controlar la congestión. Para ello, se puede reducir el tamaño del *buffer SFTP* y en consecuencia su ventana de congestión evitando sobrecargar los buffers intermedios en el enlace con el servidor. El tamaño por defecto del buffer son 16384 *kbytes* y en esta prueba se reduce a 3072 *kbytes.*

Comando ejecutado:

```
sftp -oPort=2222 -B 3072 127.0.0.1
```

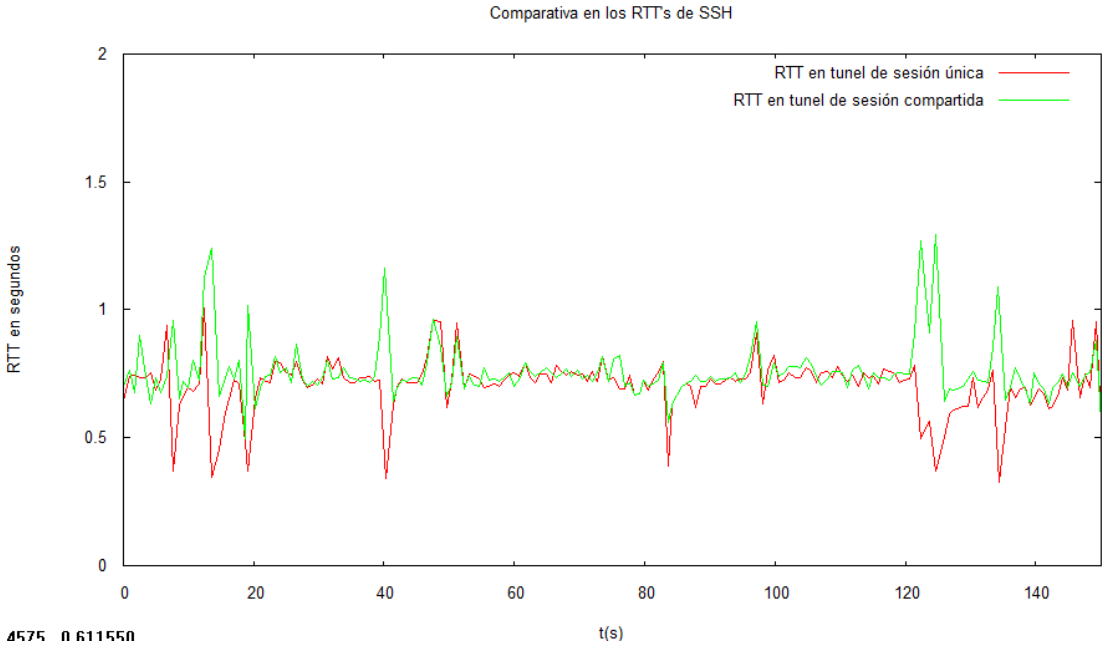

Figura 3-12: Comparativa de *RTT* entre túneles SSH con de sesión única y compartida, ambos con el tamaño de *buffer* ajustado.

Tabla 3-5: Datos estadísticos de RTT en túneles SSH con tráfico en túneles que comparten sesión con el tamaño del buffer SFTP ajustado

	RTT Medio (ms)	Desviación estándar (ms)
túnel de sesión compartida	755	10
túnel de sesión única	706	11

En este escenario, pese a reducir el tamaño del *buffer* a 3072, la tasa se transferencia SFTP se mantiene alrededor de los 600 *kbytes/s,* por lo que el ajuste del buffer no repercute negativamente en el *throughput* de la transferencia SFTP. Sin embargo, los resultados de latencia mejoran notablemente especialmente en el canal que comparte sesión con dicha transferencia SFTP.

El tamaño de buffer elegido tiene una justificación. Empíricamente se han realizado pruebas para controlar el ancho de ventana usado por SFTP y la latencia experimentada. La reducción del buffer hasta el valor propuesto únicamente ha conllevado reducción de latencia al evitar saturación en los buffers intermedios. Reducirlo más incurriría en un desaprovechamiento de ancho de banda. Al realizar un ajuste fino sobre la cantidad de información capaz de absorber por el enlace se evitan situaciones de congestión que conlleven descartes de paquetes, retransmisiones y retardos innecesarios. Se aprovecha mejor el ancho de banda y se puede disminuir la latencia.

Este comportamiento justificaría la implementación de un sistema que mantenga controlado el RTT regulando el flujo de datos. Para ello se podría utilizar el ancho de ventana de congestión y el RTT para calcular el ancho de banda esperado y compararlo con el obtenido. Según fuese el resultado se puede deducir si se está en congestión o no. Esta idea de control de congestión no es nueva y ya ha sido desarrollada por TCP Vegas, que propone entre otras cosas un algoritmo diferente al sistema reactivo de control de congestión RENO que clásicamente implementa TCP.

3.2.2.6. Control de congestión con TCP Vegas

Las principales diferencias de TCP Vegas frente a Reno son:

- La mejora del algoritmo de retrasmisión haciéndolo más efectivo. Incrementa ligeramente el ancho de banda.

- Un mecanismo de prevención de congestión mejorado: *Reno* es reactivo, pues han de perderse datagramas para que TCP considere que existe congestión. Vegas, propone usar los RTTs como indicador de la capacidad E2E del enlace. Mientras los tiempos RTT no aumenten, se incrementa el ancho de ventana. Además, Vegas intenta optimizar la cantidad de información excedente que se puede colocar en los buffers intermedios del enlace. Dejarlos vacíos implica una pérdida de efectividad frente a posibles aumentos del ancho de banda disponible mientras que llenarlos demasiado puede conducir a situaciones de congestión que generan pérdidas de paquetes; aspecto que se intentan evitar.[9]

- La modificación del algoritmo de *slow-start*: Sólo se aplica en uno de cada dos RTTs. La ventana de congestión se mantiene estable para medir el ancho de banda esperado a partir del tamaño de la ventana y del RTT y determinar de ese modo si se está en congestión o no. Cuando el ancho de banda esperado cae por debajo del conseguido, *Vegas* cambia del modo *slow-start* al modo lineal basado en el incremento y reducción del ancho de ventana del algoritmo de prevención de congestión. Este método evita las pérdidas iniciales en el establecimiento de una conexión con *RENO*, y aunque aún en estas circunstancias pueden ocurrir situaciones de congestión, son de menor calado y se converge antes hacia el ancho de banda disponible.

Sin embargo, TCP Vegas sufre serios problemas en entornos que practiquen *rerouting* a los paquetes TCP para realizar, por ejemplo, balanceado de carga en el servidor. En estos escenarios, Vegas no funciona correctamente porque calcula de forma errónea los anchos de banda disponibles, pudiendo llegar a trabajar en constante congestión sin percatarse de ello [10]. Sin embargo, en el escenario que planteamos para nuestra red DS, este supuesto no se presenta y por consiguiente este defecto no se

producirá nunca haciendo altamente recomendable la implementación de TCP Vegas tanto en los *players* como en el servidor.

El *player* dispone de un Linux embebido en el que se puede activar el sistema de control de congestión propuesto. Para ello se reconfigura el *Kernel* y se compila de nuevo con dicha opción seleccionada:

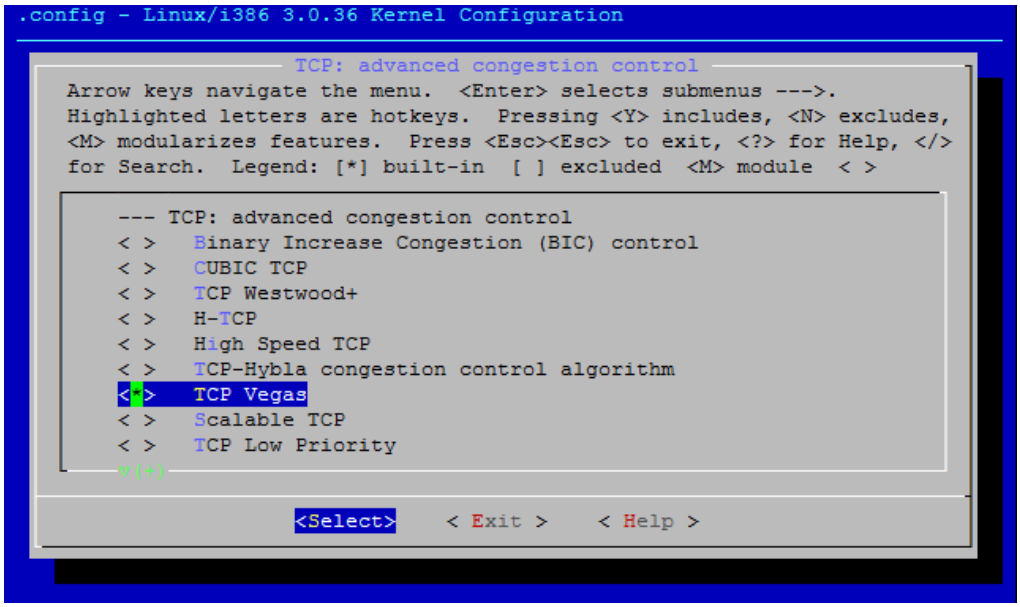

Figura 3-13: Modificación del *kernel* del *player* para dar soporte a TCP Vegas

En el servidor central resulta más sencillo; pues al estar basado en *Linux Centos Server*, el *kernel* dispone de una gran variedad de módulos precompilados que sólo requieren de su activación en caso de necesitarse:

Activación temporal de TCP Vegas de forma temporal

```
echo Vegas > /proc/sys/net/ipv4/tcp_congestion_control
```

Para efectuar este cambio de forma permanente se puede proceder de dos maneras:

- Editando **/etc/sysctl.conf** para modificar **net.ipv4.tcp_congestion_control=Vegas**
- Ejecutando el siguiente comando:

```
/sbin/sysctl -w net.ipv4.tcp_congestion_control=Vegas
```

Una vez realizados estos cambios se procede a realizar la medición de RTTs con los mismos parámetros que en el apartado anterior para comparar los resultados entre una comunicación tunelada de sesión única y una compartida.

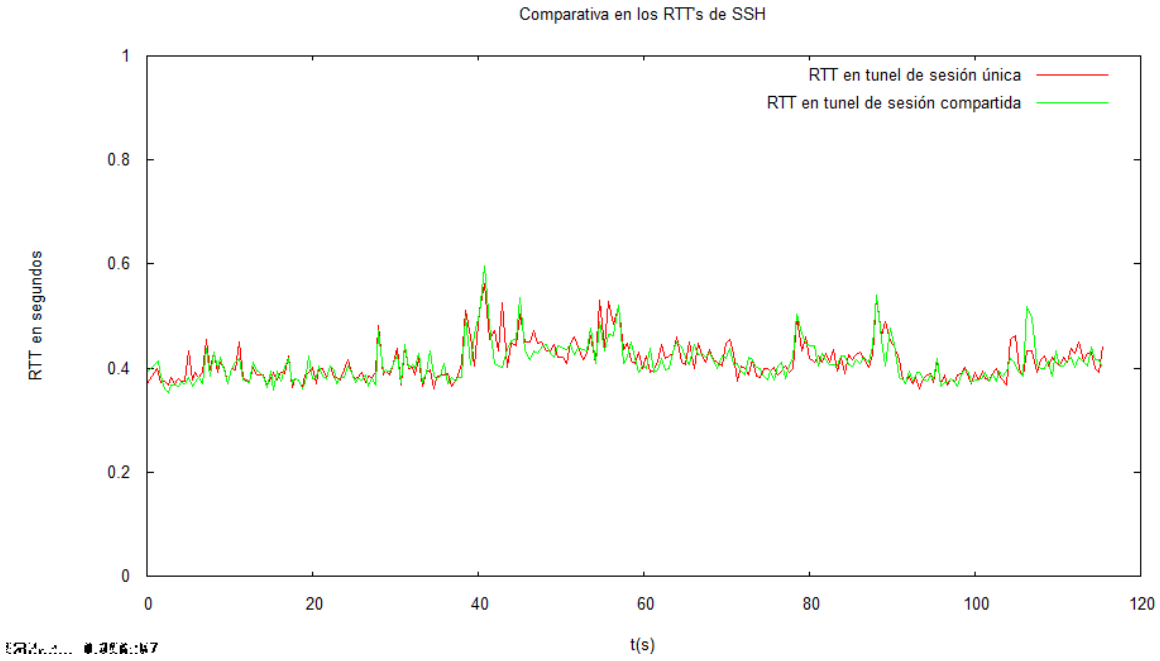

Figura 3-14: Comparativa de RTTs de túneles SSH con sesión única y con sesión compartida funcionando con TCP Vegas

Tabla 3-6: Datos estadísticos de RTT en túneles SSH SSH con sesión única y con sesión compartida funcionando con TCP Vegas

	RTT Medio (ms)	Desviación estándar (ms)
túnel de sesión compartida	408	35
túnel de sesión única	411	35

La tasa de transferencia SFTP se incrementó ligeramente para alcanzar los 620 *Kbytes*/s y los RTTs se redujeron a valores muy próximos al escenario en el que no se transmite nada por SFTP. La desviación estándar incluso se mejora, por lo que se puede concluir que el control de congestión efectuado por TCP Vegas se realiza satisfactoriamente y la latencia se mantiene controlada incluso en escenarios con transmisiones de datos masivas. Otro aspecto a destacar es la irrelevancia del status del túnel, ya que al quedar controlada la congestión el control de congestión inherente a SSH nunca se efectúa y por consiguiente el resultado es equivalente en túneles de sesión única y túneles de sesión SSH compartida con transmisiones SFTP masivas. Por ese motivo, se concluye que el control de congestión por TCP Vegas implementado tanto en el servidor como en los *players,* soluciona definitivamente todos los problemas derivados de la congestión.

QoS con enlaces 802.11

4. QOS CON ENLACES 802.11

La influencia del tipo de acceso a Internet que dispone el *player* es muy elevada dependiendo de la tecnología empleada. Concretamente, en entornos *wireless* 802.11 existen algunas herramientas que permiten gestionar el tráfico en escenarios con congestión en la red local para mejorar la QoS percibida. Para ello, la problemática a tratar en este apartado consiste en cómo abordar la planificación y marcaje del tráfico de salida del dispositivo para intentar favorecer el correcto funcionamiento del *player* independientemente del estado de congestión de la red local. Sólo se contempla trabajar en el tráfico *upload*, puesto que por norma general se desconoce el tipo de punto de acceso 802.11 presente (b/g/n, fabricante, si es compatible con Wireless MultiMedia (WMM), etc).

.

4.1. DESCRIPCIÓN DE WIRELESS MULTIMEDIA (WMM)

La tecnología WMM representa una mejora de la subcapa MAC para añadir prestaciones de calidad de servicio a las redes Wi-Fi. WMM recoge partes del estándar 802.11e y supone una extensión del sistema de coordinación distribuida (DCF) para acceder al medio inalámbrico. Este sistema intenta ofrecer a todos los dispositivos la misma prioridad y oportunidad de acceder al medio basándose en el algoritmo "*listen-before-talk*". En este escenario, cada cliente espera un tiempo aleatorio de *backoff* y entonces se transmite sólo si no hay otro dispositivo transmitiendo en ese momento. Esta forma de evitar colisiones da a los dispositivos la oportunidad de transmitir pero bajo determinadas circunstancias de alta carga de tráfico se ve afectado el rendimiento de este sistema y su ecuanimidad con los diversos clientes Wi-Fi.

Tabla 4-1: Listado de las categorías de acceso en WMM y su correspondencia con las prioridades 802.1d de Ethernet [11]

Categoría de acceso (AC)	Descripción	802.1d *tags*
Prioridad de voz WMM	La prioridad más alta. Permite múltiples llamadas de voz sobre IP (*VoIP)* con baja latencia.	7,6
Prioridad de video WMM	Prioriza el tráfico de vídeo sobre otros datos de tráfico.	5,4
Prioridad "*Best Effort*" WMM	Tráfico de dispositivos que carecen de prestaciones QoS ó tráfico menos sensible a latencia pero al que le afectan retardos elevados como la navegación por Internet.	0,3
Prioridad *Background* WMM	Tráfico de baja prioridad como descargas de ficheros y trabajos de impresión que no tienen requisitos estrictos de latencia y ancho de banda.	2,1

4.2. WMM Y EDCA: CLASIFICACIÓN DE LAS COLAS

WMM introduce capacidad de priorización de tráfico basado en las cuatro clases de tráfico ya listadas en la Tabla 4-1. Las categorías de acceso reciben más prioridad para transmitir cuanto mayor sea su nivel de clasificación. Esta forma de abordar la gestión del tráfico solventa la falta de diferenciación de los flujos y hace posible la priorización del acceso al medio de unos frente a otros. Además, la planificación del sistema WMM permite la coexistencia de dispositivos compatibles con WMM con aquellos que no lo son. Esta planificación llamada denominada EDCA se puede efectuar de modo distribuido sin intervención del AP y se incorpora del estándar 802.1e

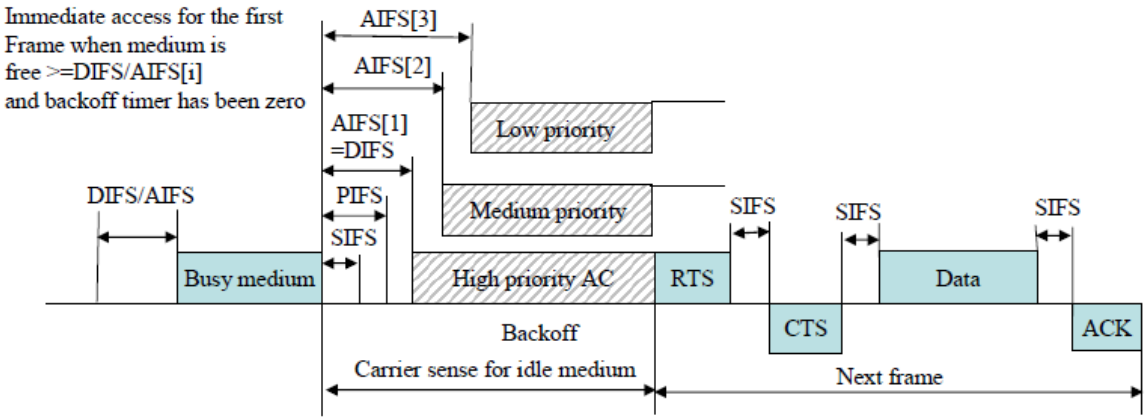

Figura 4-1: Relaciones existentes en el espacio entre trama (IFS)[12]

La priorización de WMM funciona tal y como se muestra en la Figura 4-2 y en la Figura 4-3. Las aplicaciones asignan cada paquete de datos a una clase de acceso concreto (*AC*). Los paquetes, se añaden posteriormente a cada una de las cuatro colas de transmisión independientes en el cliente (una por cada *AC*: voz, video, *best effort* y *background*).

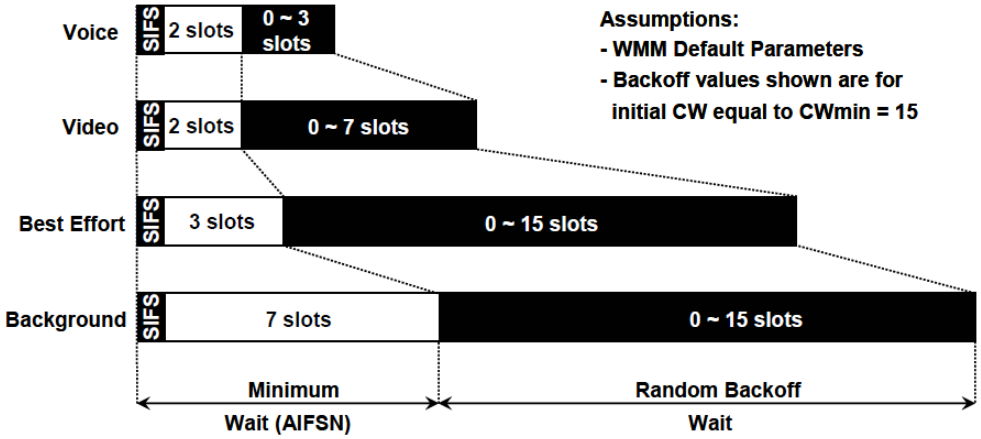

Figura 4-2: Temporizaciones en WMM [13]

El cliente dispone de un sistema para resolver colisiones internas en las contiendas por acceder al medio que se resolverán de forma similar a como se actúa posteriormente entre estaciones. Para ello se

establece un algoritmo distribuido que determina qué cliente dispondrá de la oportunidad para transmitir (TXOP)

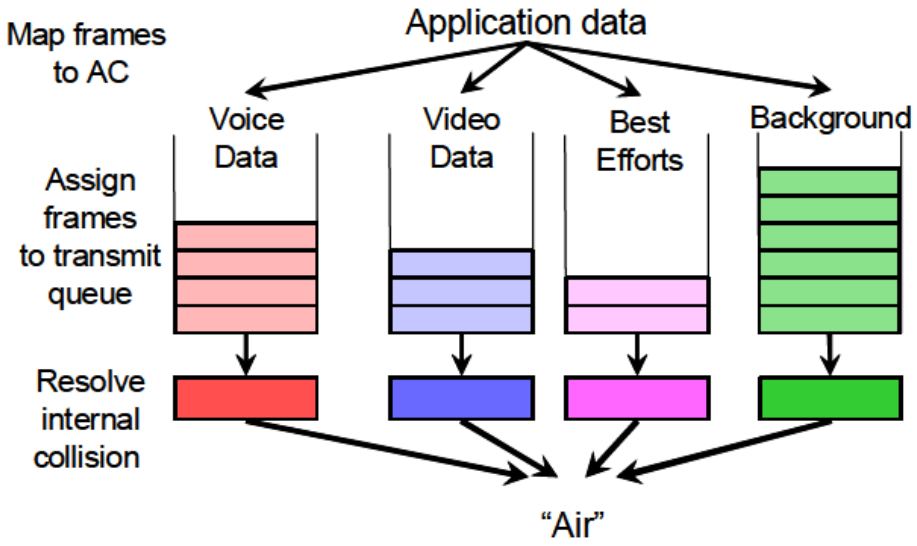

Figura 4-3: Colas de transmisión en un cliente WMM [13]

El algoritmo de resolución de colisiones responsable de la priorización de tráfico es de carácter probabilístico y depende de dos parámetros que varían para cada clase de acceso (AC):

- El tiempo mínimo entre tramas o el *Arbitrary Inter-Frame Space Number* (AIFSN).
- El tamaño de la ventana de contención, a veces referido como tiempo aleatorio de *backoff.*

Estos dos valores son pequeños para tráficos de alta prioridad y mayores según disminuya la prioridad. Para cada AC, se calcula un tiempo de *backoff* como la suma del AIFSN y un valor aleatorio comprendido entre el 0 y el valor de la ventana de contención (CW) que varía a lo largo del tiempo. Inicialmente el CW se inicializa a un valor que depende del AC, pero tras cada colisión, el CW se dobla hasta que se alcanza el valor máximo (también dependiente del AC). Tras una transmisión exitosa el valor del CW se reinicia a su estado inicial dependiente del AC.

El AC con el *backoff* menor obtiene el TXOP. Las tramas con AC superiores disminuyen progresivamente su *backoff* para incrementar la probabilidad de acceder al medio (un *TXOP*)

Tabla 4-2: Parámetros por defecto de *EDCA* para 802.11 y WMM [11]

	CWmin		CWmax		AIFSN
	OFDM	CCK	OFDM	CCK	
AC_BK	15	31	1023		7
AC_BE	15	31	1023		3
AC_VI	7	15	15	31	2 or 1 (AP)
AC_VO	3	7	7	15	2 or 1 (AP)

Slot Time = 9µs
SIFS Time = 16µs, .11a
= 10µs, .11g

WMM recoge en sus especificaciones el sistema EDCA y opcionalmente otras prestaciones de 802.11e como WMM *Scheduled Access* (WMM-SA), *Direct Link Setup* (DLS) *block acknowledgment* (B-ACK), y técnicas de ahorro energético. Resulta conveniente, por tanto, averiguar el grado de

aprovechamiento de WMM por un cliente compatible con esta tecnología que accede a una red 802.11 que no da soporte. En ese escenario se empleará EDCA.

4.3. MAPEADO QOS EN WMM

Para poder emplear EDCA en el *player*, se ha de indicar al dispositivo 802.11 cuál es la prioridad de la información que en cada momento debe transmitir. Para ello, se puede realizar un marcado en el nivel de enlace, el nivel de funcionamiento de WMM, o en el nivel de red, pues el estándar especifica que WMM también ha de ser capaz de interpretar niveles de prioridad IP del campo TOS [13]

4.3.1. Mapeado QoS desde el nivel de enlace

Para conseguir que el tráfico que generamos en los *players* se clasifique en un tipo de AC o en otro, se puede marcar a nivel 2 el tráfico para que, posteriormente y según 802.1p, el driver Wi-Fi del *player* pueda clasificarlo en el AC correspondiente. Este marcaje va en una etiqueta opcional de *Ethernet*, la 802.1Q, y al incluirla, se recalcula el código de redundancia cíclica de trama (*CRC)*.

Preámbulo	Inicio de trama	*MAC* destino	*MAC* origen	Tipo/Long.	Datos	*CRC*	*Gap* entre tramas
7 *Bytes*	1 *Byte*	6 *Byte*	6 *Byte*	2 *Byte*	Hasta 1500 *Bytes*	4 *Bytes*	12 *Bytes*

Preámbulo	Inicio de trama	*MAC* destino	*MAC* origen	Tipo/Long.	Etiqueta 802.1Q	Datos	*CRC*	*Gap* entre tramas
7 *Bytes*	1 *Byte*	6 *Byte*	6 *Byte*	2 *Byte*	4 *Bytes*	Hasta 1500 *Bytes*	4 *Bytes*	12 *Bytes*

Figura 4-4: Modificación en la trama Ethernet para incluir el marcaje 802.1Q

Sin embargo, en *Linux*, la posibilidad de marcar directamente tráfico a nivel 2 está muy restringida. La forma más directa de hacerlo es utilizando VLANs y valiéndonos de 802.1Q. Para ello, se crearían interfaces virtuales por las que se redirigir el tráfico que considerásemos oportuno. Este tráfico quedará marcado con las *tags* 802.1Q.

Marcar directamente 802.1p teóricamente es posible con un nuevo filtro de iptables [21] según el cual se pueden poner etiquetas 802.1p si el driver lo permite. En primer lugar indicamos a Eth0 como interfaz con VLANs configuradas de prioridad 2 y 3

```
vconfig set_egress_map eth0 3 3
vconfig set_egress_map eth0 2 2
```

Luego usamos la capacidad de editar los *buffers* de socket del *kernel* de *Linux*:

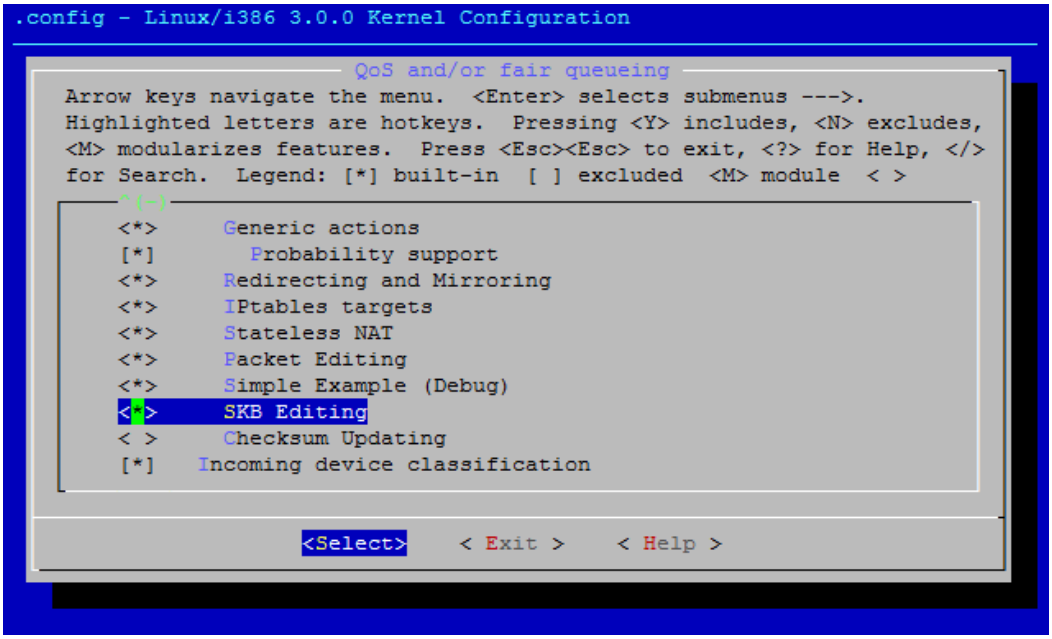

Figura 4-5: Activación de *SKB editing* en *Networking support -> Networking options -> QoS and/or fair queuing*

```
# ej de filtrado por ToS [22] y marcado a nivel skb (Linux Socket Buffers) [23]:
tc qdisc add dev eth0 root handle 1: sfq
tc filter add dev eth0 parent 1: protocol ip u32 match ip tos 0x48 0xff action skbedit priority 2
tc filter add dev eth0 parent 1: protocol ip u32 match ip tos 0x68 0xff action skbedit priority 3
tc filter add dev eth0 parent 1: protocol ip u32 match ip tos 0x00 0xff action skbedit priority 0
```

Para validar el marcado, habría que capturar ese tráfico realizando las conexiones a un *hub* en el que conectemos el terminal en modo *promiscuo*, ya que un *router* convencional que no implemente VLANs suprime dichas *tags*. Por otro lado, la validación del marcaje a nivel 802.1p o q en la máquina origen o destinataria tampoco es posible. Según [24] No se tienen garantías de poder capturar el tráfico 802.1q en el terminal que manda el tráfico ya que el driver de la tarjeta puede quitar la etiqueta antes de que podamos obtenerla con *wireshark* o *tcpdump*.

La alternativa podría ser trabajar en un *switch* que permita monitorización de tráfico de puertos. Para ello, se puede utilizar el software *DD-WRT* corriendo en alguno de los dispositivos compatibles [25] y duplicar y redirigir el tráfico objeto del análisis al terminal con *wireshark* [26]. Sin embargo, dado que no se dispone en este momento del material necesario para proceder con las medidas empíricas, se procede a realizar las pruebas de marcaje a nivel 3.

4.3.2. Mapeado QoS desde el nivel de red

Para poder clasificar los tráficos en los 4 ACs de WMM tenemos otra alternativa. Aunque el nivel dos de capa OSI no debería poder examinar los niveles superiores para averiguar su calidad de servicio, es una práctica que se emplea de facto y por consiguiente marcando adecuadamente el campo TOS de IP se puede conseguir clasificación de QoS a nivel inferior.

Una vez mapeado la QoS a nivel 2, WMM lo interpreta clasificando el tráfico en colas según la Tabla 4-4.

En IPv4 el campo de *Type of Service* (ToS) se emplea para definir el campo *Differenciated Services Code Point* (DSCP) [14] y el *Explicit Congestion Notification* (ECN) [15]

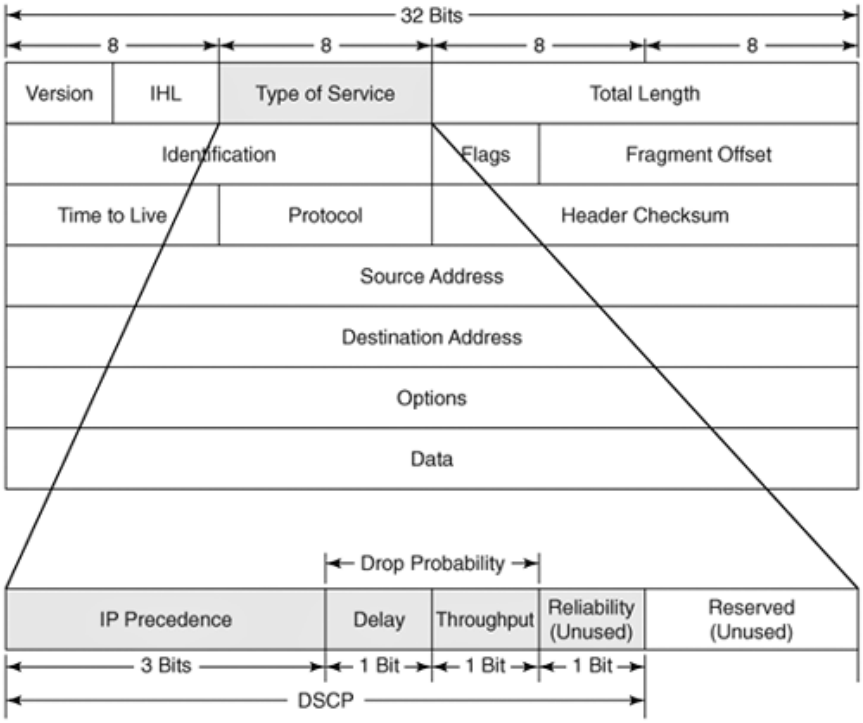

Figura 4-6: Ubicación del campo ToS en IPV4 y sus partes principales.

Tabla 4-3: Nomenclatura de los bits del campo *ToS* en *IPv4*[16]

S5 (P2)	S4 (P1)	S3 (P0)	S2	S1	S0	CN	CN

Los campos P2, P1, P0 se dedican para definir el nivel de prioridad del tráfico. Los bits S2, S1 y S0 caracterizan el tráfico con requisitos de retraso (s2) y ancho de banda (s1). S0 no se usa y siempre está a cero. A su vez, CN1 y CN0 que se plantearon como señalización de congestión [15] han quedado obsoletos y tampoco se emplean. Por norma general van a 0.

En base a los bits de prioridad, se pueden definir los siguientes niveles:

Tabla 4-4: Correspondencia de los niveles de prioridad *DSCP* con los *tags* 802.1D y los AC de 802.1e y WMM[13]

P2 P1 P0	802.1D	AC
0 0 0	0	BE
0 0 1	1	BK
0 1 0	2	BK
0 1 1	3	BE
1 0 0	4	VI
1 0 1	5	VI
1 1 0	6	VO
1 1 1	7	VO

En la Tabla 4-4, se muestra la relación entre los distintos niveles de prioridad a nivel 3, 2 y 802.11 cuando se emplea 802.11e/WMM

Respecto los bits S, se define el comportamiento de los paquetes por salto ó *Per Hop Behavior* (PHB).

- ***Default* PHB*:*** *DSCP*: "00000000". Este es el ToS que recibe el tráfico por defecto, recibiendo un comportamiento de *Best Effort*.

- ***Expedited Forwarding* (EF) PHB**[17]**:** Este tipo de marcaje se reserva para tráfico con altos requisitos de prioridad, y condiciones de ancho de banda y retardo estrictas. Este tipo de información sirve a los *routers* que realizan gestión del ToS para darlos prioridad o descartarlos si son incapaces de cursar el tráfico con las prestaciones requeridas por el flujo de datos. Al marcar un flujo de datos con características rígidas en términos de ancho de banda o retardo se da a entender al *router* que si no se consigue tramitar dicho tráfico con esos requisitos no servirá y se procede a descartarlo.

- ***Assured Forwarding* (AF) PHB** [18]**:** El marcaje AF se utiliza en tráfico al que se desea priorizar frente al tráfico con prioridad estándar. Existen en total nueve tipos distintos de tráfico AF dependiendo de su nivel de prioridad y su caracterización en los bits S.

Según [19], sólo determinados *codepoints* (valores del TOS) serán aceptados como tal en los *routers*, tratando el resto de combinaciones como tráfico por defecto bien convirtiendo el TOS a 0x00 ó simplemente ignorándolo. Todas las clases de tráfico vistas hasta el momento se pueden clasificar en la siguiente tabla:

Tabla 4-5: Valores del ToS reconocidos por *DiffServ* y su correspondencia con *DSCP codepoints* y *PHB's*

Prioridad	*DSCP* Class (*Per Hop Behavior*)		*DSCP Decimal*	*DSCP Codepoints*	*TOS (Hex)*	*Bits TOS*		
						P's	*S'*	*CN*
7	-----		56	0x38	0xE0	111	000	00
6	-----		48	0x30	0xC0	110	000	00
5	*Expedited Forwarding (EF)*		46	0x2E	0xB8	101	110	00
4	*Assured Forwarding (AF4)*	*AF41*	34	0x22	0x88	100	010	00
		AF42	36	0x24	0x90	100	100	00
		AF43	38	0x26	0x98	100	110	00
3	*Assured Forwarding (AF3)*	*AF31*	26	0x1A	0x68	011	010	00
		AF32	28	0x1C	0x70	011	100	00
		AF33	30	0x1E	0x78	011	110	00
2	*Assured Forwarding (AF2)*	*AF21*	18	0x12	0x48	010	010	00
		AF22	20	0x14	0x50	010	100	00
		AF23	22	0x16	0x58	010	110	00
1	*Assured Forwarding (AF1)*	*AF11*	10	0x0A	0x28	001	010	00
		AF12	12	0x0C	0x30	001	100	00
		AF13	14	0x0E	0x38	001	110	00
0	*Default*		0	0x00	0x00	000	000	00

La información proporcionada en la Tabla 4-4 y en la Tabla 4-5 no aclara perfectamente cómo se corresponden los campos DSCP a niveles inferiores y viceversa, ya que los niveles de prioridad 6 y 7 no están definidos como *PHB's* y según [19] un *router* que implemente *DiffServ* puede tratar de forma impredecible estos tipos de tráfico. En estos casos, algunos fabricantes toman iniciativas propias al respecto, como Cisco propone en sus dispositivos:

Tabla 4-6: Correspondencias *QoS* en los *AP's* de Cisco[20]

AVVID 802.1 UP-Based Traffic Type	*AVVID IP DSCP*	*AVVID 802.1p UP*	*IEEE 802.11e UP*
Network control	-	7	-
Inter-network control (LWAPP control, 802,11 management	48	6	7
Voice	46 (*EF*)	5	6
Video	34 (*AF41*)	4	5
Voice Control	26 (*AF31*)	3	4
Background (gold)	18 (*AF21*)	2	2
Background (gold)	20 (*AF22*)	2	2
Background (gold)	22 (*AF23*)	2	2
Background (silver)	10 (*AF11*)	1	1
Background (silver)	12 (*AF12*)	1	1
Background (silver)	14 (*AF13*)	1	1
Best Effort	0 (*BE*)	0	0, 3
Background	2	0	1
Background	4	0	1
Background	6	0	1

Por consiguiente, independientemente de que funcione correctamente las correspondencias de Tabla 4-4 en la que se expone la equivalencia QoS de los valores de prioridad de *TOS* con 802.1e/WMM, en un escenario real en el que haya un *router* intermedio el marcado de prioridades 6 y 7 puede perderse y por tanto no es recomendable usarlos. En el caso en que el propio *player* disponga de tarjeta Wi-Fi incorporada sí pueden usarse esos dos valores de prioridad, puesto que los paquetes pasarían directamente del nivel 2 al driver de la tarjeta Wi-Fi y éste realizaría la gestión adecuada de colas.

4.4. MEDIDAS EXPERIMENTALES

Se plantean dos escenarios de pruebas para verificar y cuantificar el efecto del tráfico diferenciado en los flujos de datos. En el primer caso se busca averiguar si el efecto de los flujos de tráfico pesado del propio dispositivo sobre los flujos de control se puede mitigar en la interfaz 802.11 con el uso de las prestaciones EDCA. Para ello se trabaja con un sólo *player* transmitiendo dos flujos de datos simultáneamente a los que se le aplica *DiffServ* empleando el sistema de colas de transmisión mostrado en la figura 4-3. En el segundo escenario se busca comprobar la efectividad de EDCA a la hora de priorizar tráfico entre dos interfaces 802.11 diferentes. Un *player* transmitirá información con prioridad alta frente a las transmisiones de prioridad estándar de un ordenador convencional que comparte red Wi-Fi con el *player*.

En ambos escenarios el *player* utiliza un *kernel* estándar con algoritmo de control de congestión TCP Reno para centrarse en la diferenciación de flujos y su tratamiento en la interfaz inalámbrica. Las pruebas realizadas se efectuaron con un dispositivo USB 802.11 conectado directamente al *player*. En particular se empleó un *D-Link DWL-G122* con *chipset* rt73 de *Ralink*.

4.4.1. **Primer escenario**

En este primer escenario se pretende verificar si el driver Wi-Fi junto con el hardware del chip interpreta adecuadamente el marcado de los paquetes para hacer uso de las colas específicas de cada clase de tráfico. Esto permitirá dar más prioridad a unos flujos concretos del *player* frente a otros del mismo *player*. Este chip implementa las cuatro clases de tráfico de WMM. Otros fabricantes y modelos implementan sólo tres o dos de ellas, como *Realtek* en sus chips RTL8192CE/CU.

Figura 4-7: Escenario de pruebas para comprobar el funcionamiento de EDCA en caso de flujos de tráfico múltiples en el propio dispositivo

Se comienza verificando la inicialización correcta del chip en el *player* mediante los mensajes *debug* lanzados por el *driver* a través del *kernel*.

```
[    7.464815] phy0 -> rt73usb_validate_eeprom: EEPROM recovery - NIC: 0xffef
[    7.464824] phy0 -> rt73usb_validate_eeprom: EEPROM recovery - Led: 0xe000
[    7.464830] phy0 -> rt73usb_validate_eeprom: EEPROM recovery - RSSI OFFSET A: 0x0000
[    7.465052] phy0 -> rt2x00_set_chip: Info - Chipset detected - rt: 2573, rf: 0002, rev: 000a.
[    7.502244] ieee80211 phy0: Selected rate control algorithm 'minstrel_ht'
[    7.525370] Registered led device: rt73usb-phy0::radio
[    7.525434] Registered led device: rt73usb-phy0::assoc
[    7.525518] Registered led device: rt73usb-phy0::quality
[   20.812606] phy0 -> rt2x00lib_request_firmware: Info - Loading firmware file 'rt73.bin'.
[   20.868563] phy0 -> rt2x00lib_request_firmware: Info - Firmware detected - version: 1.7.
[   20.938551] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 0 - CWmin: 3, CWmax: 4, Aifs: 2, TXop: 102.
[   20.940543] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 1 - CWmin: 4, CWmax: 5, Aifs: 2, TXop: 188.
[   20.942545] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 2 - CWmin: 5, CWmax: 10, Aifs: 3, TXop: 0.
[   20.944543] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 3 - CWmin: 5, CWmax: 10, Aifs: 7, TXop: 0.
[   22.994454] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 2 - CWmin: 4, CWmax: 10, Aifs: 3, TXop: 0.
[   22.996427] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 3 - CWmin: 4, CWmax: 10, Aifs: 7, TXop: 0.
[   22.998424] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 1 - CWmin: 3, CWmax: 4, Aifs: 2, TXop: 94.
[   23.000423] phy0 -> rt2x00mac_conf_tx: Info - Configured TX queue 0 - CWmin: 2, CWmax: 3, Aifs: 2, TXop: 47.
```

Figura 4-8: Mensajes *debug* enviados por el módulo Ralink WI-FI del *Kernel* captados con *dmesg*

Para realizar las medidas se genera tráfico en el *player* con destino el servidor de la red local. El tráfico es UDP unidireccional. El uso del medio inalámbrico se reserva en este escenario al enlace entre el *player* y el *router* Wi-Fi. El *router* se conecta al servidor local por *Ethernet* cableado.

```
ra0     IEEE 802.11bg  ESSID:"Ateire"
        Mode:Managed  Frequency:2.412 GHz  Access Point: 2C:B0:5D:D9:DB:37
        Bit Rate=54 Mb/s   Tx-Power=20 dBm
        Retry  long limit:7   RTS thr:off   Fragment thr:off
        Encryption key:off
        Power Management:on
        Link Quality=66/70  Signal level=-44 dBm
        Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
        Tx excessive retries:0  Invalid misc:64   Missed beacon:0
```

Figura 4-9: Estado del dispositivo USB 802.11 utilizado durante las pruebas.

La transferencia de datos se realiza en diversos flujos con la intención de sobrepasar en conjunto la capacidad del canal y de ese modo forzar el reparto de ancho de banda en base a la clase de tráfico. Debe tenerse en cuenta que los paquetes UDP se generan con tamaño de datos reducido (200 *bytes*) para favorecer el efecto de la sobrecarga de 802.11 y facilitar la lectura gráfica de los resultados. Además, para comprobar el comportamiento del *scheduler* de WMM, los tráficos diferenciados se inician en distintos momentos temporales para comprobar que la política aplicada en cada cola se adapte a la situación de tráfico en cada momento:

- La medida experimental comienza con tráfico de tipo AF41 (Prioridad 4) hacia el puerto 5003
- Posteriormente, a los 10s se transmite:
    - Tráfico con DSCP 56, hacia el puerto 5000 (Prioridad 6) y tráfico con DSCP 48 hacia el puerto 5001 (Prioridad 7). Ambos marcajes son ignorados por el driver y tratados como tráfico *Best Effort* por lo que se expuso en el apartado anterior.
    - También se transmite tráfico AF31, AF21, AF11 y *Best Effort* hacia los puertos 5004, 5005, 5006 y 5007 respectivamente con prioridades 3,2,1 y 0.
    - Para finalizar, también se transmite tráfico sin marcar al puerto 5008.
- Finalmente a los 20s, se comienza a transmitir tráfico EF y AF31 hacia los puertos 5002 y 5004 con prioridades 5 y 3 respectivamente.

Todos los flujos generados tienen una duración de 40s, permitiendo que, al empezar unos antes que otros finalicen en momentos diferentes, pudiendo analizar las medidas tomadas por el *scheduler* WMM para repartir el ancho de banda libre en cada momento.

Generación de los flujos de datos UDP:

```
sleep 10 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5000 &
sleep 10 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5001 &
sleep 20 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5002 &
sleep 0 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5003 &
sleep 20 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5004 &
sleep 10 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5005 &
sleep 10 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5006 &
sleep 10 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5007 &
sleep 10 && iperf -c 192.168.1.108 -u -b 5m -t 40 -l 200 -p 5008 &
```

El marcaje de los paquetes se realiza utilizando *iptables*:

```
iptables -t mangle -A OUTPUT -p udp --dport 5000 -j DSCP --set-dscp 56
iptables -t mangle -A OUTPUT -p udp --dport 5001 -j DSCP --set-dscp 48
iptables -t mangle -A OUTPUT -p udp --dport 5002 -j DSCP --set-dscp 46
iptables -t mangle -A OUTPUT -p udp --dport 5003 -j DSCP --set-dscp 34
iptables -t mangle -A OUTPUT -p udp --dport 5004 -j DSCP --set-dscp 26
iptables -t mangle -A OUTPUT -p udp --dport 5005 -j DSCP --set-dscp 18
iptables -t mangle -A OUTPUT -p udp --dport 5006 -j DSCP --set-dscp 10
iptables -t mangle -A OUTPUT -p udp --dport 5007 -j DSCP --set-dscp 0
```

El resultado de la simulación puede analizarse en el gráfico siguiente:

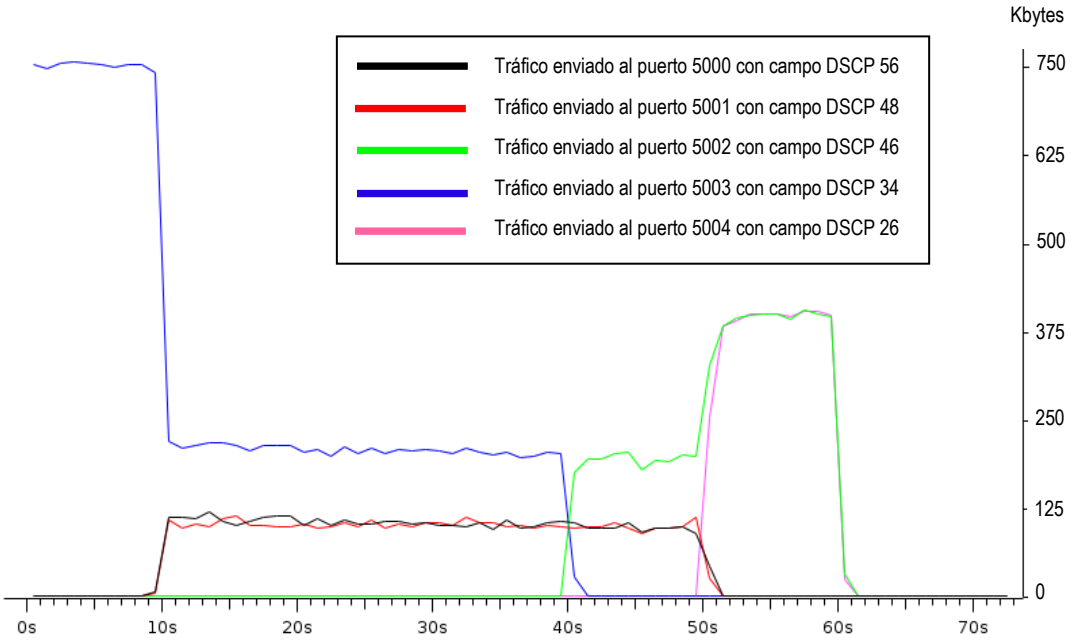

Figura 4-10: Resultados de las medidas del primer escenario: Flujos con diversas clasificaciones *ToS* modelados automáticamente por EDCA

Tabla 4-7: Correspondencias empíricas entre *Class of Service* (CoS) y *Differenciated Services Code Point* (DSCP) en el primer escenario

			PRIO 1 AC_VO	PRIO 2 AC_VI	PRIO 3 AC_BE	*PRIO 4* *AC_BK*
DSCP	*CoS*	Puerto dst.				
56		5000			X	
48		5001			X	
46	EF	5002		X		
34	AF41	5003	X			
26	AF31	5004				X
18	AF21	5005			X	
10	AF11	5006			X	
0	0	5007			X	

De la información obtenida se deduce que el dispositivo utiliza cuatro colas de tráfico en total, como se preveía. Las clases de prioridad más altas (DSCP 56 y 48) son tratadas como tráfico por defecto y sólo los tráficos con clase de servicio *Expedited Forwarding* (EF) y *Assured Forwarding* 4 y 3 (AF4* y AF 3*) son diferenciados realmente. La gráfica anterior omite el tráfico de los puertos 5005, 5006, 5007 por ser muy similares al 5000 y 5001 ya representados clasificados de facto como en la cola *Best Effort* (AC_BE).

Un dato relevante se deduce del comportamiento del tráfico con CoS EF, aparentemente no tratado como correspondería, pues recibe menos recursos que el tráfico *Best Effort*. La causa a esto puede deberse a una implementación especial de la cola de tráfico AC_BE, pues parece ser ajena al WMM *Scheduler* del dispositivo. Los parámetros de modelado de la cola parecen adecuados (CW_MIN,

CW_MAX, TXOP), pero no se ajustan en base a la ocupación en el resto de colas. Por lo tanto, el tráfico AC_BE no es modelado, pudiendo llegar a recibir mejor trato que tráficos con mayor prioridad como es el caso del experimento. Debe notarse que este efecto sólo es perceptible en el caso en que existan numerosos flujos de datos de tipo AC_BE y puede que represente un *bug* del *firmware* o del *driver*. En todo caso, se comprueba que la implementación de la Tabla 4-4 en la que se muestra la correspondencia ente los valores de 802.1d y los niveles de prioridad DSCP no se produce en absoluto.

4.4.1.1. Uso de clientes 802.11

El comportamiento con dispositivos 802.11 alternativos varía enormemente en base al chipset empleado y el tipo de driver usado. Se plantearon experimentos similares con un *router/AP* 802.11 *TP-LINK TL-MR3220* que se utilizaba como cliente 802.11. El *player* se conectó con este dispositivo mediante *Ethernet* cableado y el *router/ap* servía de cliente WI-FI a la red objetivo. El *router/ap* funcionaba en modo *Wireless Distribution System* (WDS) para poder cumplir este objetivo.

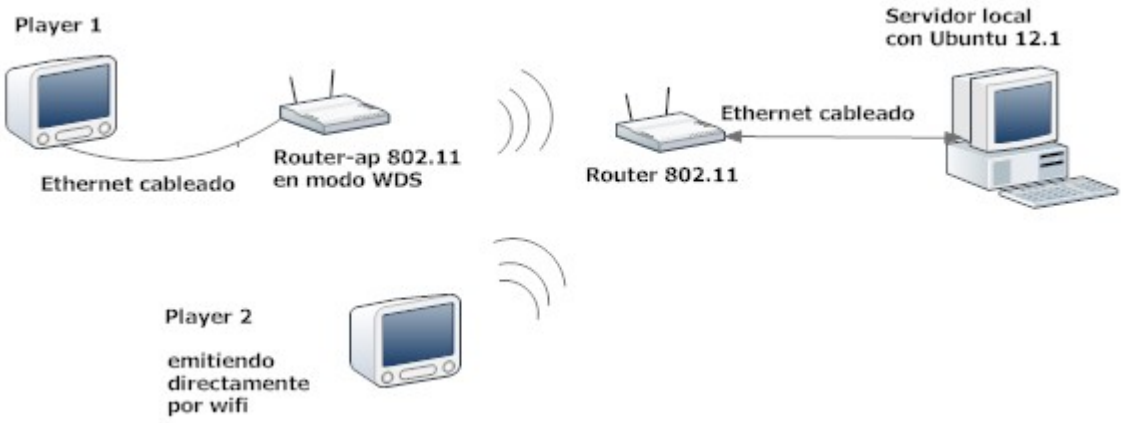

Figura 4-11: Alternativa al uso de hardware USB 802.11: Cliente Wi-Fi.

El *player* 1 genera tráfico UDP marcado con prioridad alta con destino el servidor de red local y el *player* 2 genera tráfico UDP sin marcar con el mismo destino

El tráfico UDP generado por ambos *players* tiene las siguientes características

- Ancho de banda objetivo a 100 Mbps con el propósito de intentar saturar la red para forzar reparto de recursos.
- Duración 30 segundos en los dos flujos.
- Longitud de *payload* 200 *bytes* por paquete

Los comandos ejecutados para establecer el segundo escenario son los que siguen:

Comandos lanzados en *player* 1

```
#Limpieza total de reglas iptables
iptables -F
iptables -t nat -F
iptables -t mangle -F
iptables -X

#Trafico prioritario nivel 4 AF43 en DS
iptables -t mangle -A OUTPUT -p udp --dport 5002 -j DSCP --set-dscp 34

#Generación de tráfico desde DS
iperf -c 192.168.1.108 -u -b 100m -t 30 -l 200 -p 5002
```

Comandos lanzados en el *player* 2

```
#Limpieza total de reglas iptables
iptables -F
iptables -t nat -F
iptables -t mangle -F
iptables -X

#Trafico sin marcar
iptables -t mangle -A OUTPUT -p udp --dport 5002 -j DSCP --set-dscp 0

#Generación de tráfico desde el ordenador
iperf -c 192.168.1.100 -u -b 100m -t 30 -l 200 -p 5001
```

El comportamiento del tráfico capturado se muestra en la siguiente figura:

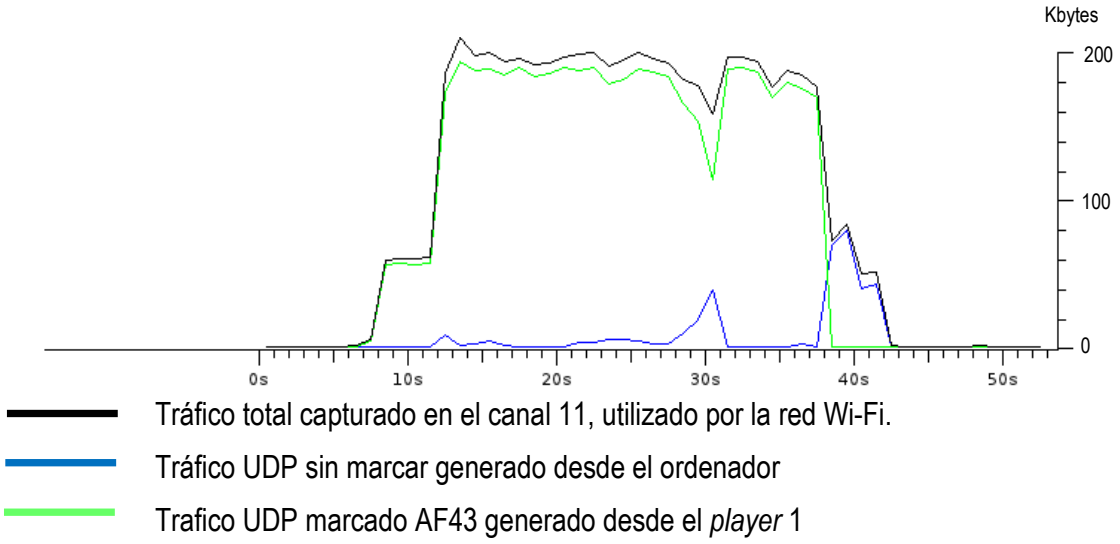

Figura 4-12: Medidas tomadas con un portátil en modo monitor a nivel 802.11

En la Figura 4-13, se observa cómo domina el canal el tráfico UDP del *player* 1 marcado como tráfico de alta prioridad frente al tráfico sin marcar del ordenador. Es apreciable que cuando se produce un desvanecimiento en el cliente 802.11 usado por el *player* 1 para radiar se aprovecha por el ordenador para transmitir.

Sin embargo, medidas que se realizaron posteriormente reflejan que en el cliente Wi-Fi del *player* el tráfico cursado siempre se está procesando como tráfico de alta prioridad. El cliente Wi-Fi usado por el *player* 1 se conecta a la red como "*router* en extensión de la red", usando prestaciones de red distribuida WDS. En el estándar 802.11 se define que la comunicación entre puntos de acceso de la misma red en modo WDS debería hacerse por cable. No obstante, el dispositivo empleado en el caso analizado permite realizar este enlace por Wi-Fi, característica que lo convertía a efectos prácticos en un cliente Wi-Fi para cualquier dispositivo que use Ethernet cableado (*player* 1). Sin embargo, como esta prestación extra no se encuentra incluida en el estándar, el fabricante la implementó de modo que las comunicaciones entre APs de la misma red se produzcan siempre con la máxima prioridad a nivel 802.11, ignorando en todo momento el marcaje aplicado por el *player* 1. Esto resulta comprensible, puesto que al funcionar un *router/ap* en modo WDS, las comunicaciones con el *router* de la red deberían gozar de la mayor prioridad posible; puesto que se entiende que aglutina comunicaciones de otros usuarios hacia el *router* de la red.

Existen otros dispositivos que funcionan como auténticos clientes Wi-Fi que se están analizando como alternativa. Un potencial candidato es el *Pepwave surf on-the-go* descrito brevemente en el Anexo.

4.4.2. Segundo escenario

En el segundo escenario, se plantea analizar el grado de funcionamiento de EDCA a la hora de priorizar los flujos de datos del *player* frente a otros flujos de la red 802.11 de otros dispositivos. Se busca verificar que el marcado propuesto consigue que el sistema EDCA trate de forma diferente a los flujos de datos, dando mayor prioridad a los del *player*. Para ello se establecen transferencias de datos UDP desde el *player* y el ordenador hacia un servidor local de red con el fin de analizarlas y verificar que las transferencias del *player* con alta prioridad son capaces de conseguir mayor ancho de banda que las del ordenador.

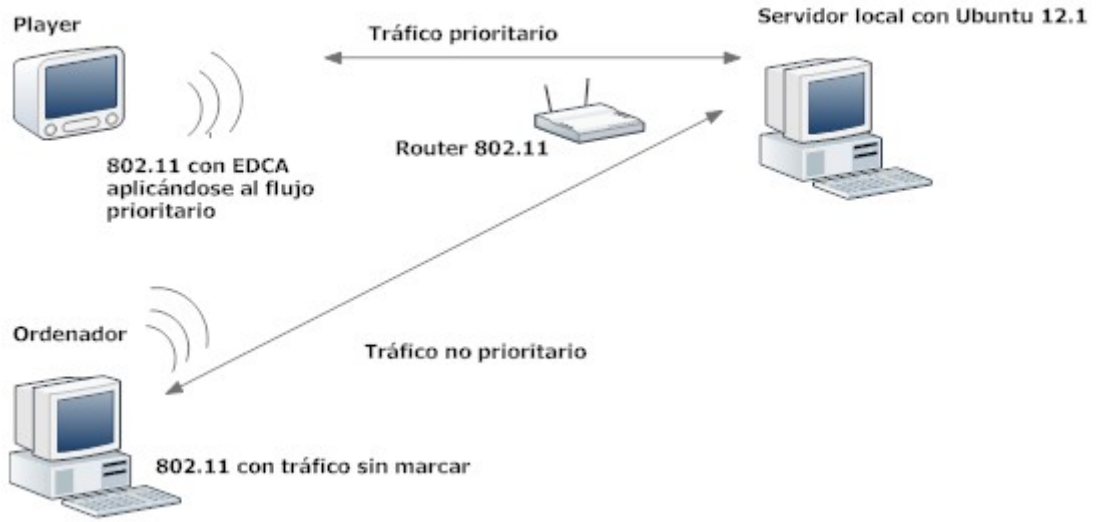

Figura 4-13: Escenario de pruebas para comprobar el funcionamiento de *EDCA* en caso de que exista tráfico de fondo sustancial de otro dispositivo de la red.

La transferencia de datos desde el *player* se realiza en diversos flujos que sobrepasan la capacidad del canal. Debe tenerse en cuenta que los paquetes UDP se generan con tamaño de datos reducido (200 bytes) para favorecer el efecto de la sobrecarga de 802.11 y facilitar la lectura gráfica de los resultados.

```
sleep 10 && iperf -c 192.168.1.108 -u -b 5m -t 60 -l 200 -p 5002 &
sleep 20 && iperf -c 192.168.1.108 -u -b 5m -t 50 -l 200 -p 5003 &
sleep 0 && iperf -c 192.168.1.108 -u -b 5m -t 70 -l 200 -p 5004 &
```

El marcaje de los paquetes se realiza utilizando *iptables* del *player*:

```
iptables -t mangle -A OUTPUT -p udp --dport 5002 -j DSCP --set-dscp 46
iptables -t mangle -A OUTPUT -p udp --dport 5003 -j DSCP --set-dscp 34
iptables -t mangle -A OUTPUT -p udp --dport 5004 -j DSCP --set-dscp 26
```

Desde el ordenador se establece un único flujo de datos

```
sleep 30 && iperf -c 192.168.1.108 -u -b 5m -t 20 -l 200 -p 5005 &
```

La configuración de la tarjeta de red del ordenador durante las pruebas es la siguiente

```
wlan0     IEEE 802.11bgn  ESSID:"Ateire"
          Mode:Managed  Frequency:2.412 GHz  Access Point: 2C:B0:5D:D9:DB:37
          Bit Rate=150 Mb/s   Tx-Power=17 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:on
          Link Quality=51/70  Signal level=-59 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:26  Invalid misc:424   Missed beacon:0
```

Figura 4-14: Estado de la interfaz de red 802.11 del ordenador utilizado durante las pruebas

Los resultados de las pruebas pueden apreciarse en la siguiente figura:

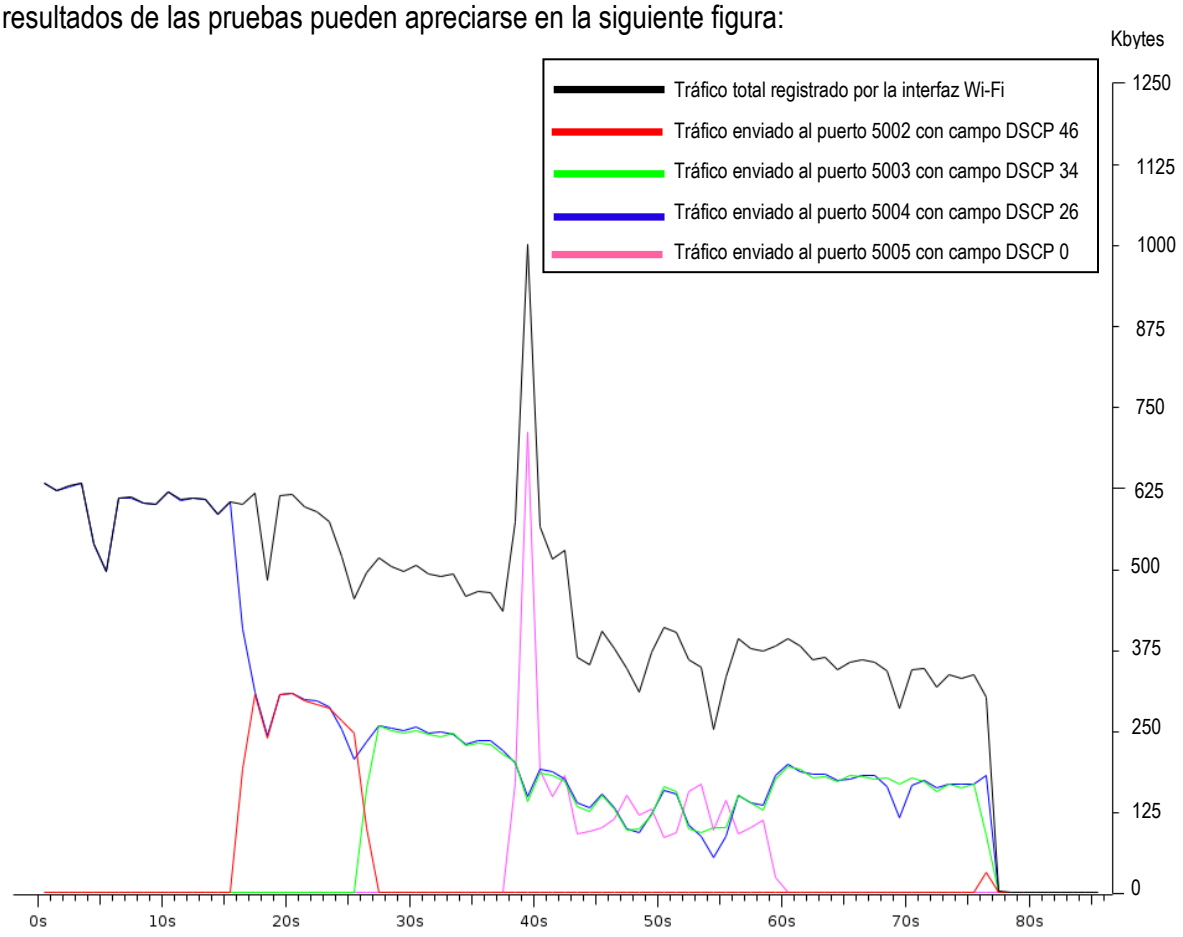

Figura 4-15: Resultados de las medidas del segundo escenario: Tráfico del *player* con distintas prioridades compaginado con tráfico de un ordenador de la misma red sin ningún tipo de prioridad.

Tabla 4-8: Correspondencias empíricas entre *CoS* y *DSCP* en el segundo escenario

				PRIO 1 AC_VO	PRIO 2 AC_VI	PRIO 3 AC_BE	PRIO 4 AC_BK
DSCP	*CoS*	Disp. origen	Puerto dst.				
46	EF	*Player 1*	5002				X
34	AF41	*Player 1*	5003	X			
26	AF31	*Player 1*	5004		X		
0	*0*	*Ordenador*	5005			X	

Existen varios aspectos a destacar de este escenario. En primer lugar, el tráfico EF no se trata localmente igual que en el ejemplo anterior y por lo tanto no se gestiona adecuadamente por la política de colas de este driver. En primer lugar cabe destacar el cambio de comportamiento del tráfico *EF* frente al tráfico AF31 y el tráfico *Best Effort* existente en la red. Esto puede deberse a un *bug* en el planificador WMM del driver. Por otro lado, confirmar que el tráfico AF41 sigue siendo el de mayor prioridad del *player*, pues en pequeños desvanecimientos del canal, este tráfico persiste en detrimento de AF31.

El ordenador sólo transmite tráfico *background* frente al *player* que transmite flujos con niveles de prioridad superiores. De la gráfica se deduce que el ancho de banda empleado por los flujos es similar en AF41, AF31 y BE, por lo que pese a todo, se concluye que el *player* dispone de mayor canal disponible

que el ordenador, pues aunque los flujos de datos están parejos (salvo el espurio inicial) la transferencia neta de información del *player* en total es mayor que la del ordenador. Por otro lado, el ordenador dispone de una tasa máxima de transferencia de 155 Mbps negociada con el AP mientras que el *player* sólo una tasa de 54 *Mbps*, y pese a ello el *player* ha sido capaz de transmitir mayor cantidad de información por lo que EDCA ha funcionado consiguiendo diferenciar el tráfico del *player* frente al del ordenador sin necesidad de configurar nada en el *router*. Sin embargo esa diferenciación en las distintas clases de tráfico del player no se produce tal y como se especifica en el estándar y debe tenerse en cuenta a la hora de efectuar el marcaje si se elije este chip WI-FI para trabajar con los *players*.

Conclusiones

5. CONCLUSIONES

En este documento se ha expuesto el funcionamiento de una red profesional de *Digital Signage* en la que tanto el *player* como el server se han diseñado para trabajar conjuntamente con ese cometido para ofrecer flexibilidad y escalabilidad. Esto permite reducir los problemas derivados del establecimiento manual de las conexiones y la conectividad en movilidad. Sin embargo, el sistema de conexiones empleado se encuentra con problemas de latencia debidos fundamentalmente a dos aspectos: La congestión en las comunicaciones tuneladas y los problemas derivados del acceso a Internet a través de *hotspots* o puntos de acceso Wi-Fi saturados.

Para abordar la latencia de las comunicaciones tuneladas, se ha expuesto el funcionamiento de las conexiones entre el *player* y el servidor y se han propuesto medidas que mejoran puntualmente la latencia como minimizar el tráfico de control, emplear un sistema de distribución de contenidos en diferido, diferenciar el tráfico en la red local y emplear una versión modificada de OpenSSH que use UDP en las comunicaciones tuneladas.

No obstante, la causa fundamental de la latencia elevada es la congestión que determinados flujos de datos del *player* (transferencias SFTP) causan en el resto de comunicaciones simultáneas entre el *player* y el servidor. Para evitar esto último se plantea el separar las sesiones SSH de las transferencias SFTP del resto de comunicaciones. De este modo se facilita el acceso al ancho de banda del resto de flujos aprovechando que cada sesión de SSH realiza control de congestión. Por otro lado, se observa que la congestión se puede llegar a evitar sin perjudicar el *throughput* regulando el *buffer* del cliente SFTP que manda la información. Este efecto se consigue de forma transparente sustituyendo el algoritmo Reno de control de congestión de TCP por Vegas. Este algoritmo limita la tasa de transferencia de cada flujo ajustando el ancho de ventana de contención TCP en base a la capacidad del canal estimada con los RTT de los ACK. La validez de TCP Vegas se pone de manifiesto en pruebas experimentales en las que se monitoriza el RTT de peticiones de *echo* tuneladas en un escenario en el que el *player* está transmitiendo datos vía SFTP. TCP Vegas evita que se produzca la congestión permitiendo que siempre exista ancho de banda disponible y la latencia se mantenga controlada.

La latencia también puede depender en algunos casos del estado de la red Wi-Fi local, especialmente en lugares públicos donde el acceso se comparte como en *hotspots*, comercios, ferias, etc. En estas circunstancias se plantea la posibilidad de mejorar la conectividad de dispositivos DS mediante el empleo de EDCA, disponible en dispositivos 802.11 con capacidades WMM. Se ha propuesto utilizar un dispositivo USB que implemente estas funcionalidades y se han valorado en dos escenarios. En primer lugar se verifica la capacidad de priorizar entre varios flujos del mismo dispositivo y en segundo lugar se comprueba si es capaz de priorizar tráfico del dispositivo DS frente al existente en la red.

Los resultados obtenidos indican que el desarrollo del driver y el hardware concreto pueden ser un problema, pues el comportamiento del mismo varía dependiendo del chip y fabricante y además pueden existir *bugs* de driver por resolver. Para evitar esto último se propone la utilización de clientes WIFI 802.11 con hardware y driver propietario incluido en el firmware que permitan a un player con interfaz Ethernet 802.3 conectarse por cable a este dispositivo y de forma transparente acceder a una red 802.11. Este cliente Wi-Fi, al tener capacidades WMM puede recibir paquetes marcados a nivel ToS y mapear su QoS a 802.11.

Anexo: Dispositivos hardware

6. ANEXO: DISPOSITIVOS HARDWARE

6.1. PEPWAVE SURF ON-THE-GO

Este dispositivo se plantea como posible cliente Wi-Fi a utilizar para gestionar el tráfico diferenciado y aplicar EDCA.

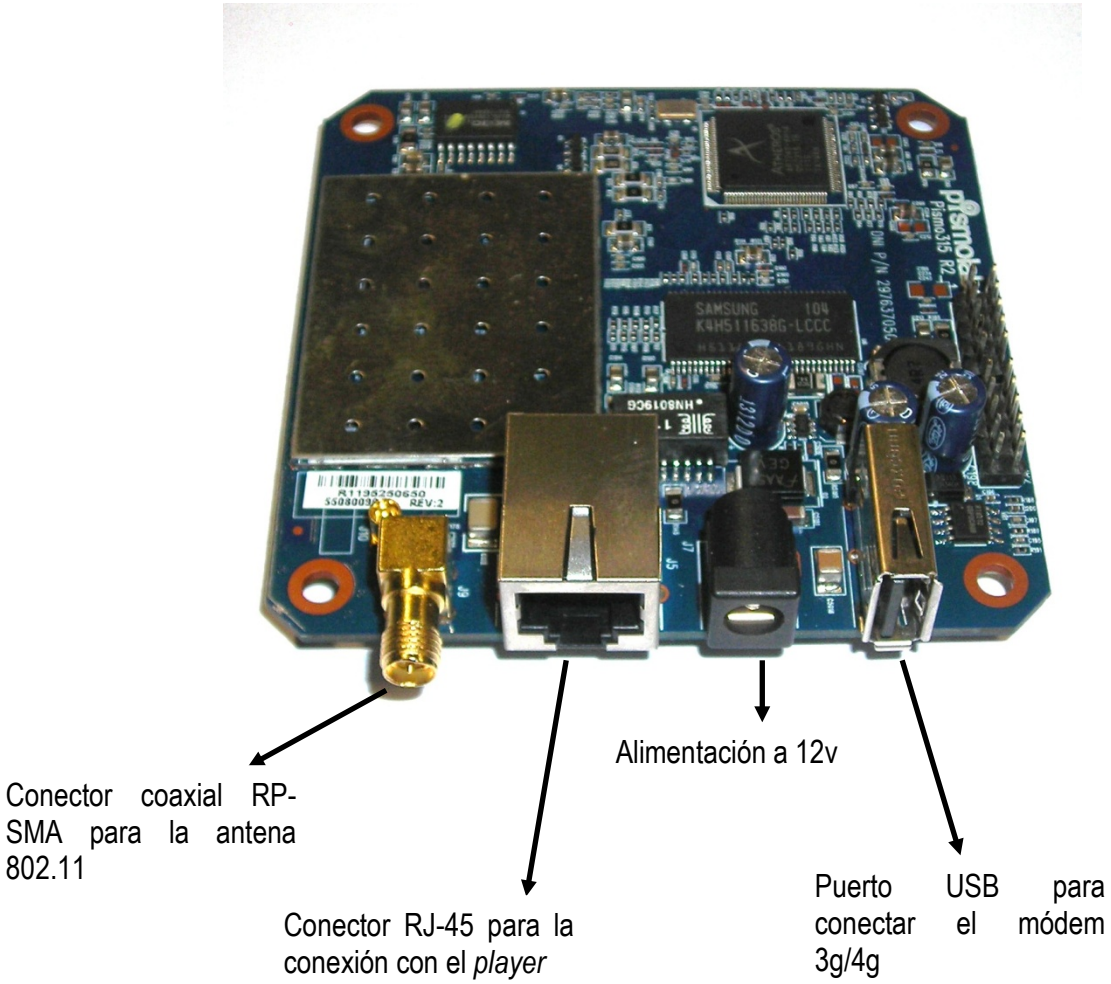

Figura 6-1: Unidad *PEPWAVE Surf on-the-go* para integración

El dispositivo, posee una web de control que se monta mediante un *proxy* en el servidor web del *player* pudiendo gestionarlo todo desde la web del propio *player*. Este aparato sirve de puente entre los niveles de red del *player* y del punto de acceso volviendo transparente el enlace 802.11.

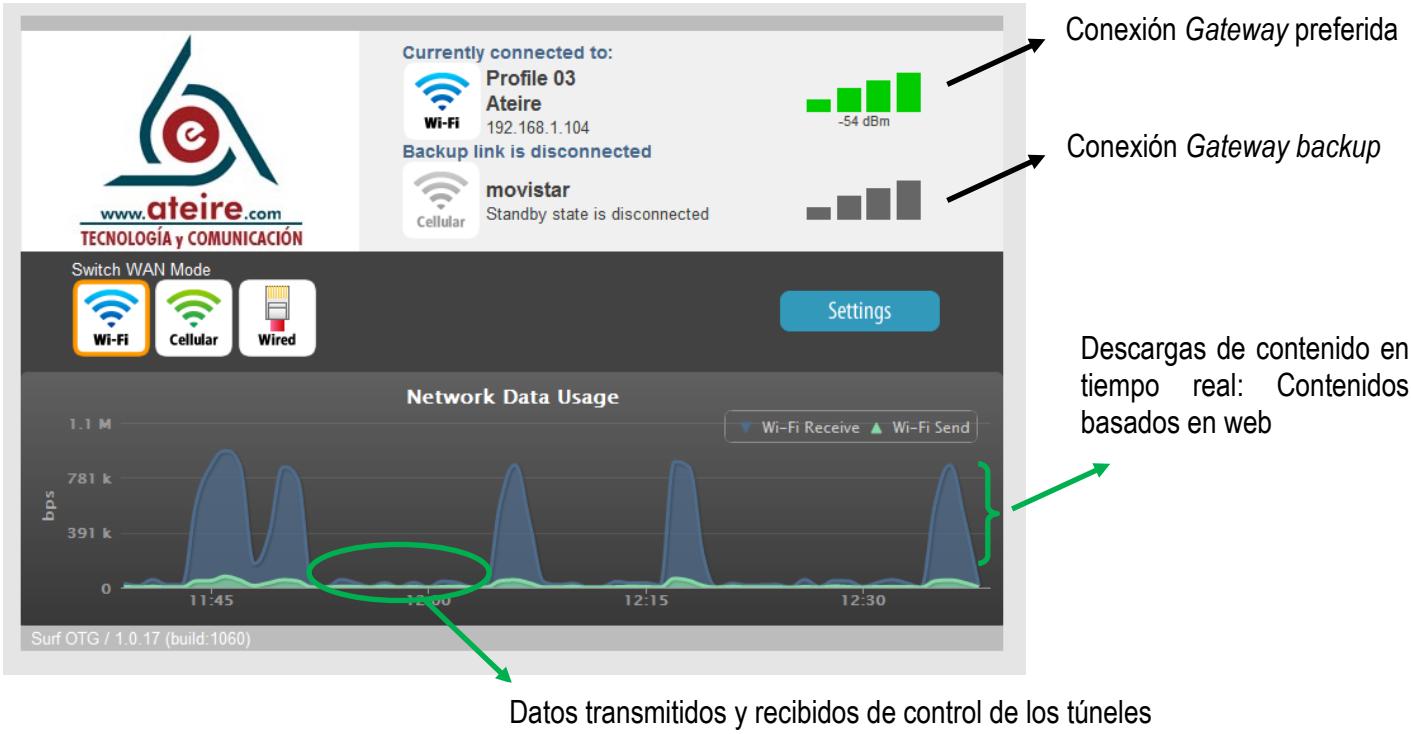

Figura 6-2: Interfaz gráfica de control del *Pepwave Surf on-the-Go* dando cobertura a un *player* DS en funcionamiento

6.2. ESD2: DIGITAL SIGNAGE PLAYER

El Esd2 V2.0 es el *player* de *Digital Signage* que se ha empleado en este documento. En su versión estándar está compuesto por un chasis de aluminio que disipa directamente los 8w del bus N10/ICH7 y *GPU* 945GME de *Intel* (7w) junto con el procesador *Intel Atom* n270 (1w) siendo por tanto totalmente *fanless*. Dispone de aceleración por hardware vía PCI-E Broadcom BCM70015 [27] compatible con la reproducción de videos mp4 (h264) de hasta 1920x1080 y perfil 4 de compresión. Este dispositivo incluye también su componente flash 10.3 para dar soporte a los contenidos basados en esta tecnología de Adobe®. También ofrece compatibilidad con contenidos web html5 y dispone de un motor PHP 5.1 local para ejecutar scripts web en el propio dispositivo.

Figura 6-3: Esd2: Player de Digital Signage desarrollado en Ateire Tecnología y Comunicación

El disco duro alberga una partición encriptada para el sistema operativo. La llave de encriptación es diferente para cada dispositivo y se genera durante el arranque del *initrd* de *Linux*. De ese modo se garantiza cierto grado de inviolabilidad en el sistema estadístico de pases de publicidad y del resto del S.O. La otra partición del disco duro se dedica a los datos de contenidos exclusivamente. Ambas particiones funcionan bajo *Reiser FS* 3.6.21 que favorece la integridad del sistema de archivos en caso de fallos eléctricos. Además, esta plataforma cuenta con *watchdog* que vigila que el sistema responda cada 60 segundos para que, en caso de no hacerlo, se reinicie eléctricamente.

También dispone de puerto RS-232 para regular aspectos de la pantalla como encendido/apagado, señal de entrada, etc. Dispone de conectividad *Ethernet* vía rj-45 1Gbps y 802.11 b/g/n vía RTL8191SEvA de *Realtek*.

Figura 6-4: Esd2: Interior del dispositivo

6.3. RASPBERRY PI COMO DIGITAL SIGNAGE PLAYER

Esta plataforma basada en SoC (*System On a Chip*) *BCM2835* [28] que consiste en un dispositivo de muy bajo consumo basado en tecnología *ARM11*. El tamaño de la placa es similar al de una tarjeta de crédito y su precio cercano a los 30€. Integra una GPU *VideoCore* IV, 256/512 *MB* de *RAM* y el procesador *ARM1176JZF-S* a 700*Mhz*. Este dispositivo complementa al Esd2 y se emplea actualmente como interfaz GPIO (General Purpose Input Output) programable mediante llamadas HTTP. Este dispositivo es considerado también como una futura alternativa de bajo costo al player actual y sólo necesita tener operativo el driver gráfico que está actualmente en desarrollo.

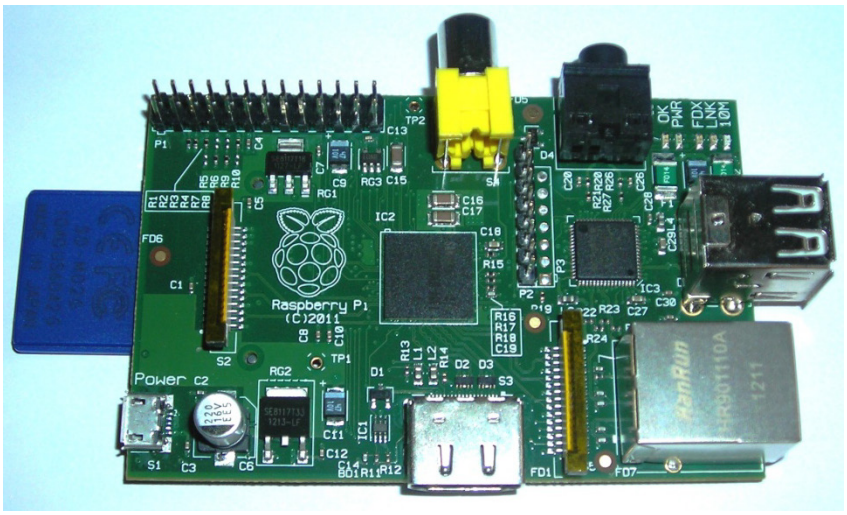

Figura 6-5: Raspberry PI modelo B

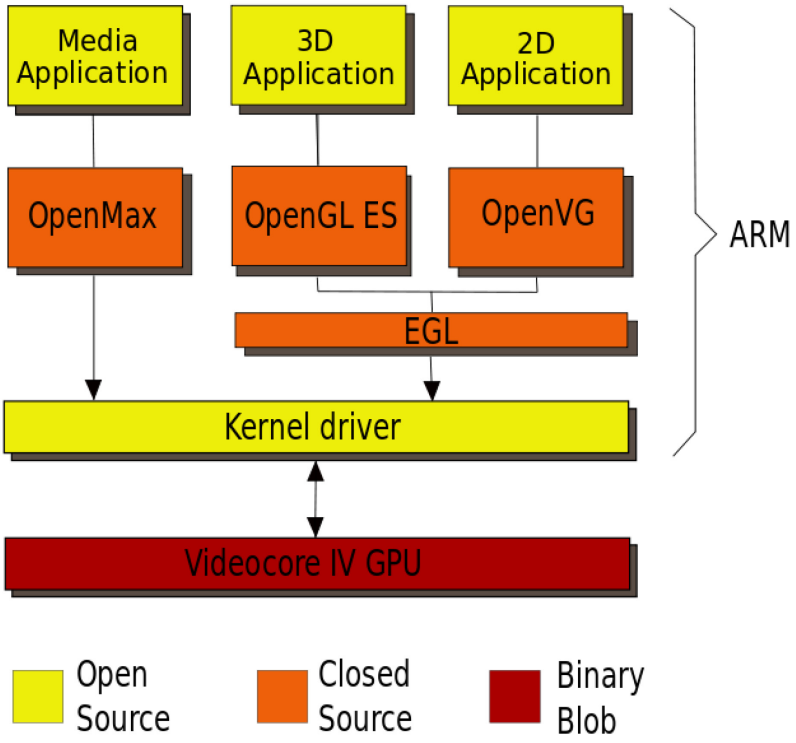

Figura 6-6: Arquitectura interna de Raspberry PI

Debian dispone de un repositorio específico para *Raspberry* llamado *Raspbian* que está desarrollando y optimizando un sistema totalmente abierto para esta plataforma. Sin embargo, y pese a recientes avances en la optimización de la Unidad de Punto Flotante integrada (FPU)[29], la utilización de la Unidad de Procesamiento Gráfico (GPU*)* desde entornos X de forma integral está aún por llegar. Sin embargo, existen desarrolladas aplicaciones *stand-alone* que permiten reproducir vídeo directamente sobre el *framebuffer* sin necesidad del entorno X usando las prestaciones decodificadoras h.264 de la GPU. En concreto, *omxplayer* se ha testeado con éxito y reproduce vía GPU vídeos mp4 de 1080p [30]

Sin embargo, el desarrollo de esta plataforma como *player* potencial se encuentra en espera debido a la excesiva demanda de dispositivos y su escasa producción [31]. No obstante, desde finales de 2012 la oferta y la demanda de los dispositivos se ha estabilizado y Farnell, una de sus comercializadoras, ha anunciado que ya es capaz de dar respuesta a los pedidos en plazos de tres semanas.

Acrónimos

7. ACRÓNIMOS

Acrónimo	Término
AC	*Access Category*
AC_BE	*Access Category Best Effort*
AC_BK	*Access Category Background*
AC_VI	*Access Category Video*
AC_VO	*Access Category Voice*
ACK	*Acknoledgement*
ADSL	*Asymmetric Digital Subscriber Line*
AF	*Assured Forwarding*
AIFS	*Arbitration Inter-Frame Space*
AIFSN	*AIFS-Number*
AP	*Access Point*
B-ACK	*Block-ACKnowledgement*
BD	Base de Datos
BW	*BandWidth*
CCK	*Complementary Code Keying*
CTS	*Clear To Send*
CW	*Contention Window*
DCF	*Distributed Coordination Function*
DIFS	*DCF Inter-Frame Space*
DLS	*Direct Link Setup*
DS	*Digital Signage*
DSCP	*Differenciated Services Code Point*
E2E	*End to end*
ECN	*Explicit Congestion Notification*
EDCA	*Enhanced Distributed Channel Access*
EDGE	*Enhanced Data Rates for* GSM *Evolution*
EF	*Expedited Forwarding*
FPU	*Floating-Point Unit*
Gbps	*Gigabits per second*
GPIO	*General Purpose Input Output*
GPRS	*General Packet Radio Service*
GPU	*Graphics Processing Unit*
HCCA	HCF *Controlled Channel Access*
HCF	*Hybrid Coordination Function*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IP	*Internet Protocol*
LAN	*Local Area Network*
MAC	*Media Access Control*
Mbps	*MegaBits per second*
OFDM	*Orthogonal Frequency Division Multiplexing*
PCF	*Point Coordination Function*

PHB	*Per Hop Behaviour*
PIFS	DCF *Inter-Frame Space*
QoS	*Quality of Service*
RTS	*Request To Send*
RTT	*Round-Trip delay Time*
SFTP	*Secure File Transfer Protocol*
SIFS	*Short Inter-frame Space*
SLA	*Service Level Agreement*
SoC	*System on Chip*
SSDP	*Simple Service Discovery Protocol*
SSH	*Secure SHell*
SSID	*Service Set IDentifier*
TCP	*Transmission Control Protocol*
TXOP	*Transmission Opportunity*
UDP	*User Datagram Protocol*
UPnP	*Universal Plug and Play*
USB	*Universal Serial Bus*
VLAN	*Virtual* LAN
VoIP	*Voice over Internet Protocol*
WDS	*Wireless Distribution System*
Wi-Fi	*Wireless Fidelity*
WMM	Wi-Fi *MultiMedia*
WMM-SA	WMM *Scheduled Access*

Bibliografía

8. BIBLIOGRAFÍA

[1] **C. Santander,** «End-to-end available bandwidth estimation and monitoring», University of South Florida, 2009.

[2] **O. Hondaa, H. Ohsakia, M. Imasea, M. Ishizukaba, J. Murayama**, «Understanding TCP over TCP: Effects of TCP Tunneling on», *IEIC Technical Report (Institute of Electronics, Information and Communication Engineers),* vol. 104, nº 438, pp. 108-122, 2004.

[3] **M. Karlsson, A. Habib**, «SSH over UDP», University of Gothenburg, Sweden, 2012.

[4] **T. Ylonen, C. Lonvick**, Ed., «RFC4254: The Secure Shell (SSH) Connection Protocol», 2006. http://tools.ietf.org/html/rfc4254.

[5] **J. Haslam**, «Dtrace introduction», Oracle, Sun Microsystems, Noviembre de 2011. https://wikis.oracle.com/display/DTrace/Introduction.

[6] **Foxtrot Systems Ltd**, «Repositorio dtrace para Linux», 9 de Octubre de 2012. https://github.com/dtrace4linux.

[7] **S. Sanfilippo**, «hping.org», Noviembre de 2005. http://www.hping.org/.

[8] **S. Ostermann**, «tcptrace.org», Abril de 2003. http://www.tcptrace.org/.

[9] **Lawrence S. Brackmo, Larry L. Petterson**, «TCP Vegas: End to end congestion avoidance on a Global Internet», vol. 13, nº 8, pp. 1465-1480, Octubre de 1995.

[10] **Richard J. La, Jean Walrand, Venkat Anantharam**, «Issues in TCP Vegas», Berkley, California, 2001.

[11] **G. Smith, DSP Group**, «802.11 QoS Tutorial», 10 de Noviembre de 2008. http://www.ieee802.org/1/files/public/docs2008/avb-gs-802-11-qos-tutorial-1108.pdf.

[12] **IEEE Computer Society**, «IEEE Std 802.11e™-2005», 11 de Noviembre de 2005. http://standards.ieee.org/getieee802/download/802.11e-2005.pdf.

[13] **Wi-Fi Alliance**, «Wi-Fi CERTIFIED™ for WMM™ - Support for Multimedia Applications with Quality of Service in Wi-Fi® Networks», 1 de Septiembre de 2004. http://www.wi-fi.org/files/wp_1_WMM%20QoS%20In%20Wi-Fi_9-1-04.pdf.

[14] **K. Nichols, S. Blake, F. Baker, D. Black**, «RFC2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers», Diciembre de 1998. http://tools.ietf.org/html/rfc2474.

[15] **K. Ramakrishnan, S. Floyd, D. Black**, «RFC3168: The Addition of Explicit Congestion Notification (ECN) to IP», http://tools.ietf.org/html/rfc3168.

[16] **The Linux Foundation**, «Queues for QoS 802.11e/WMM EDCA», http://linuxwireless.org/en/developers/Documentation/mac80211/queues.

[17] **IETF Network Working Group**, «RFC3246: An Expedited Forwarding PHB (Per-Hop Behavior)», http://tools.ietf.org/html/rfc3246.

[18] **IETF Network Working Group**, «RFC2597: Assured Forwarding PHB Group», Junio de 1999. http://tools.ietf.org/html/rfc2597.

[19] **D. Grossman, Motorola, Inc**., «RFC3260: New Terminology and Clarifications for Diffserv», Abril de 2002. http://tools.ietf.org/html/rfc3260.

[20] **Cisco Systems**, «Unified Wireless QoS»,

http://www.cisco.com/en/US/docs/solutions/Enterprise/Mobility/emob41dg/ch5_QoS.html#wp1051421.

[21] **Peter P. Waskiewicz Jr**.,LAN Access Division, Intel Corp., «Converged Networking in the Data Center», de *Linux Symposium*, Montreal, Quebec Canada, 2009.

[22] **T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Larroy**, «Linux Advanced Routing & Traffic Control», 19 de Mayo de 2012. http://lartc.org/howto/lartc.qdisc.filters.html.

[23] **Intel Corporation (e1000-eedc)**, «Data Center Bridging (DCB) for Intel® Network Connections», 4 Diciembre de 2010. ftp://ftp.supermicro.com/CDR-NIC_1.22_for_Add-on_NIC_Cards/Intel/LAN/APPS/FCOEBOOT/DOCS/dcb.htm.

[24] **Wireshark**, «Wireshark VLAN capture setup», http://wiki.wireshark.org/CaptureSetup/VLAN.

[25] **DD-WRT**, «Supported devices», http://www.dd-wrt.com/wiki/index.php/Supported_Devices.

[26] **Thatexplainsalot**, «Use Wireshark And DD-WRT Router Firmware To Imitate Port Monitoring On A Router Switch Port», http://thatexplainsalot.com/?p=292.

[27] **Broadcom**, «Low-Power PCI Express® HD Media Processor», http://www.broadcom.com/products/Consumer-Electronics/Netbook-and-Nettop-Solutions/BCM70015 .

[28] **Broadcom**, «High Definition 1080p Embedded Multimedia Applications Processor», http://www.broadcom.com/products/BCM2835.

[29] **L. Upton**, «Raspbian sneak peek from Dom – performance increases up the wazoo!», http://www.raspberrypi.org/archives/1565.

[30] «Omxplayer builds», http://omxplayer.sconde.net/.

[31] **The Guardian**, «Raspberry Pi demand running at '700 per second'», 5 de Marzo de 2012. http://www.guardian.co.uk/technology/2012/mar/05/raspberry-pi-demand.