ALGORITMO HTB ADAPTABLE A VARIACIONES DE ANCHO DE BANDA EN UN ENLACE INALÁMBRICO PUNTO A PUNTO 802.11

CARLOS EDUARDO TOBAR GUERRERO
ANDRO JOHNNY MAMIÁN NARVÁEZ

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
POPAYÁN
2010

ALGORITMO HTB ADAPTABLE A VARIACIONES DE ANCHO DE BANDA EN UN ENLACE INALÁMBRICO PUNTO A PUNTO 802.11

CARLOS EDUARDO TOBAR GUERRERO
ANDRO JOHNNY MAMIÁN NARVÁEZ

Trabajo de Grado para optar al título de
Ingeniero en Electrónica y Telecomunicaciones

DIRECTOR*A: Ing. Claudia Milena Hernández B*

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
Departamento de Telecomunicaciones
Grupo I+D Nuevas Tecnologías en Telecomunicaciones – GNTT
Línea de investigación: Gestión Integrada de Redes y Servicios y Arquitecturas de Telecomunicaciones
Popayán
2010

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ACRÓNIMOS

ACG	Control de Ganancia Automático (*Automatic Gain Control*)
ACK	Acuse de Recibo (*Acknowledgement*)
ASK	Modulación por desplazamiento de Amplitud (*Amplitude Shift Keying*)
AWGN	Ruido Blanco Gausiano Aditivo (*Additive White Gaussian Noise*)
BER	Tasa de error de bit (*Bit Error Rate*)
BPSK	Modulación por desplazamiento de Fase (*Binary Phase-Shift Keying*)
CBQ	Encolamiento Basado en Clases (*Class Based Queueing*)
CBR	Tasa de Bit Constante (*Constant Bit Rate*)
CIF	Equitativo independiente de las condiciones del canal (*Channel – Condition Independent Fair)*
CIF-Q	Encolamiento de Paquetes Equitativo Independiente de las Condiciones del canal (*Channel–Condition Independent Packet Fair Queueing*)
DHCP	Protocolo de Configuración Dinámica del Servidor (*Dynamic Host Configuration Protocol)*
DRR	(*Deficit Round Robin)*
FEC	Clase Equivalente de Envío (*Forwarding Equivalence Class*)
FIFO	Primero en Entrar Primero En Salir (*First In First Out*)
FCFS	Primero en Llegar Primero en ser Servido (*First Come First Served*)
GoS	Grado de Servicio (*Grade of Service*)
GPS	Compartición de Procesador Generalizado (*Generalized Processor Sharing)*
H-FSC	Curva de Servicio Jerárquica y Equitativa (*Hierarchical Fair Service Curve)*
HTB	Token Bucket Jerárquico (*Hierarchical Token Bucket*)
HTTP	Protocolo de Transferencia de Hipertexto (*Hyper Text Transfer*

Protocol)

IEEE Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers)*

IP Protocolo Internet (*Internet Protocol*)

ITU Unión Internacional de Telecomunicaciones (*International Telecommunication Union*)

IWFQ Encolamiento Equitativo Inalámbrico Idealizado (*Idealized Wireless Fair Queuing*)

LAN Red de Area Local (*Local Area Network*)

LCFS Último en Llegar Primero en Ser Servido (*Last Come First Served*)

LTFS Servidor Equitativo de Largo Término (*Long-Term Fairness Server*)

LZMA Algoritmo en Cadena de Lempel – Ziv – Markov (*Lempel-Ziv-Markov chain-Algorithm)*

MOS Puntuación Media de Opinión (*Mean Opinion Score*)

MTU Unidad de Transferencia Máxima (*Maximum Transfer Unit*)

NIC Tarjeta de Interfaz de Red **(***Network Interface Card***)**

NVRAM Memoria de Acceso Aleatorio no Volátil (*Non-volatile Random Access Memory*)

PC Computadora Personal (*Personal Computer*)

PCM Modulación por Pulso Codificado (*Pulse Code Modulation*)

PGPS Procesador de Paquetes Generalizado Compartido (*Packet Generalized Processor Sharing*)

PHB Comportamiento por salto (*Per Hop Behavior*)

PSK Modulación por Desplazamiento de Fase (*Phase-Shift Keying*)

QAM Modulación de Amplitud en Cuadratura (*Modulation Amplitude Quadrature)*

QoS Calidad de Servicio (*Quality of Service*)

QPSK Modulación de Cuadratura por Desplazamiento de Fase (*Quadrature Phase Shift Keying*)

RF Radio Frecuencia (*Radio Frequency*)

SBFA **(***Server Based Fairness Approach*)

SFQ Encolamiento Equitativo con Inicio de tiempo (*Start-time Fair Queuing*)

SNR	Relación Señal a Ruido (*Signal to Noise Ratio*)
SSH	Interprete de Ordenes Segura (*Secure Shell*)
STFQ	Encolamiento Equitativo Probabilístico (*Stochastic Fair Queuing*)
TIA	Asociación de la Industria de las Telecomunicaciones (*Telecommunications Industry Association*)
VBR	Tasa de Bit Variable (*Variable Bit Rate*)
VLAN	Red de Area Local Virtual (*Virtual Local Area Network*)
WAN	Red de Area Extensa (*Wide Area Network*)
WDS	Sistema de Distribución Inalámbrico *(Wireless Distribution System)*
WEP	Privacidad Equivalente a Cableado *(Wired Equivalent Privacy)*
WLAN	Red de Area Local Inalámbrica **(***Wireless Local Area Network*)
WPA	Acceso Protegido WiFi (*Wifi Protect Access*)
WPS	Configuración Protegida WiFi (*Wi-Fi Protected Setup*)
WRR	(*Weighted Round Robin)*
WWW	Red Global Mundial (*World Wide Web*)

INTRODUCCIÓN

En los últimos años, la industria de las telecomunicaciones ha experimentado notorios avances, los cuales han sido posibles por el constante desarrollo tecnológico y la creación de servicios con requerimientos más complejos. Esto se ha visto reflejado en las redes inalámbricas, que se han convertido en una de las mejores alternativas entre tecnologías de telecomunicaciones debido a su fácil implementación, bajo costo y flexibilidad.

Entre las tecnologías inalámbricas se encuentran opciones como: WiMax, Zegbee, Wi-Fi, entre otras; siendo Wi-Fi una de las más utilizadas en países en vía de desarrollo por su fácil implementación y bajo costo, además puede operar en una banda libre.

Sin embargo, el desempeño de Wi-Fi es bajo debido a los fenómenos presentes en el medio de propagación y su limitado ancho de banda, produciendo una baja calidad en algunos servicios. Estas dificultades han influido en el desarrollo e implementación de técnicas que permitan utilizar eficazmente los recursos disponibles en este tipo de redes.

Entre las técnicas sobresalen aquellas que realizan una adecuada gestión del ancho de banda del canal inalámbrico, denominadas planificadores de tráfico, como: HTB (*"Hierarchical Token Bueket"*) y HFSC (*"Hierarchical Fair Service Curve"*), también los gestores de colas SFQ (*"Stochastic Fairness Queuing"*) y RED (*"Random Early Detection"*),dichas técnicas están basadas en clasificación de colas para la implementación de prioridades y diferenciación de servicios, destacándose HTB por su simplicidad en la configuración y aceptable desempeño en redes inalámbricas, es uno de los más utilizados para brindar calidad de servicio. Aunque existen notables avances en planificación de tráfico no hay algoritmos que actúen dinámicamente con respecto al ancho de banda del canal.

En HTB el ancho de banda total se configura por el administrador de red antes de correr el algoritmo, por lo cual no se tienen en cuenta las fluctuaciones que puede presentar el ancho de banda en el canal a lo largo del tiempo, causadas por fenómenos en el medio de propagación.

La implementación del **algoritmo HTB adaptable a variaciones de ancho de banda en un enlace inalámbrico punto a punto 802.11** puede ser una opción para mejorar la asignación de recursos del canal inalámbrico que se realiza en HTB, ya que debido a las fluctuaciones de ancho de banda existentes en el canal que afectan la capacidad de transmisión de información, los recursos asignados a cada clase de tráfico pueden resultar exagerados o insuficientes. Por ejemplo, cuando la calidad del canal inalámbrico disminuye el ancho de banda del mismo disminuye, por lo que los recursos asignados son mayores a la capacidad del canal. Cuando la calidad del canal mejora, el ancho de banda se incrementa y los recursos asignados no están acordes con el ancho de banda en ese instante en el canal, por lo cual no se aprovechan de forma eficiente.

Este documento contiene seis capítulos que recopilan el proceso de investigación, desarrollo e innovación. El primer capítulo, “Calidad de servicio”, presenta algunos métodos de clasificación y conformación de tráfico, con el fin de brindar una visión del funcionamiento del algoritmo HTB, posteriormente se mencionan conceptos básicos sobre VoIP y los factores que afectan la QoS en VoIP.

El capítulo dos, “Fundamentos para el desarrollo de sistema HTB adaptable”, presenta algunas características del estándar 802.11, los esquemas de modulación utilizados, conceptos sobre la modulación adaptativa y la influencia de la SNR sobre el rendimiento de las redes inalámbricas, como soporte teórico del desarrollo del monitor de ancho de banda del canal inalámbrico. Posteriormente se realiza un análisis del código HTB original con el fin de entender su funcionamiento y realizar las modificaciones pertinentes que permitan adecuar el código al sistema propuesto; y por último se realiza una descripción del hardware y diferentes tipos de software.

El capítulo tres, “Diseño del sistema HTB adaptable”, describe el diseño del monitor de ancho de banda, la modificación del algoritmo HTB, para que asigne dinámicamente el ancho de banda a cada una de las clases de tráfico y por último se describe de forma general el funcionamiento del sistema HTB adaptable a variaciones del ancho de banda.

El capítulo cuatro, “Desarrollo del sistema HTB adaptable”, presenta el desarrollo del sistema de monitoreo de ancho de banda y del algoritmo HTB modificado, además de los pasos necesarios para la compilación del nuevo sistema HTB adaptable.

El capítulo cinco, “Pruebas del sistema HTB adaptable”, Realiza una descripción de los requerimientos necesarios para implementar el sistema HTB adaptable, se muestra la configuración software y hardware empleados en el enlace inalámbrico, y el tipo de pruebas que se realizan con el fin de evaluar el funcionamiento del algoritmo HTB adaptable a variaciones del ancho de banda.

El capítulo seis, “Análisis de resultados”, está enfocado en analizar los resultados obtenidos con las pruebas descritas en el capítulo anterior, teniendo en cuenta tasas de transmisión de los dos tipos de tráfico, retardo y jitter del tráfico de voz. Con el propósito de concluir que tan eficiente es el algoritmo HTB adaptable comparado con el algoritmo HTB original.

Y por último, el capítulo siete presenta las “Conclusiones y recomendaciones”, elaboradas a partir del desarrollo del presente trabajo de grado.

1. CALIDAD DE SERVICIO

El objetivo de este capítulo es aclarar conceptos teóricos necesarios para el desarrollo del proyecto, por lo que se mencionan diferentes métodos de planificación de tráfico, además de un estudio de la transmisión de voz sobre protocolo IP (VoIP), incluyendo ventajas, protocolos utilizados, y los factores que afectan la calidad de la voz en redes de paquetes.

1.1 CONTROL DE ADMISIÓN, VIGILANCIA, CONFORMACIÓN Y PLANIFICACIÓN DE TRÁFICO

1.1.1 Control de admisión

Las redes "*best-effort*"[1] y la entrada incontrolada de paquetes no garantizan QoS; una admisión inadecuada de un flujo puede poner en peligro la QoS de una conexión ya existente, por lo tanto la admisión de nuevos flujos debe ser controlada cuidadosamente para proteger la QoS de los flujos admitidos anteriormente [1].

El proceso de controlar la entrada de nuevos flujos se conoce como control de admisión, básicamente se encarga de aceptar o rechazar flujos en la red. Hay dos formas de aplicar el control de admisión: una forma natural es de manera automática durante la señalización de QoS y otra manual, es decir, alguien (normalmente el sistema de el administrador o el operador de red) tiene que tomar una decisión sobre la aceptación de nuevas peticiones. Sin embargo no es práctico realizar un control manual para pequeñas escalas de tiempo, como segundos, pero es ideal para escalas de tiempo más grandes, como días o meses.

La asignación de estos recursos es totalmente variable ya que para las fuentes que manejan un tráfico CBR[2] reservar un ancho de banda con tasa máxima es suficiente y adecuado; pero para las fuentes que manejan tráfico VBR[3] reservar un ancho de banda con una tasa máxima no sería ideal ya que se desperdiciaría ancho de banda porque las fuentes de tráfico VBR no envían el tráfico todo el tiempo a dicha tasa. Si se reserva tasa promedio, los paquetes se retrasarán durante el envió de ráfagas [2]. Lo ideal es reservar entre la tasa promedio y la tasa máxima.

[1] "Mejor esfuerzo" es la mejor forma de definir aquellos servicios para los que no existe garantía de calidad de servicio (QoS)
[2] Es el tráfico de velocidad constante.
[3] Es el tráfico de velocidad variable.

1.1.2 Vigilancia de tráfico (traffic policing)

La vigilancia ("*policing"*) de tráfico es el procesamiento de paquetes que cumplen con un perfil definido, esto se realiza paquete a paquete revisando que se cumpla con las políticas establecidas en la QoS [3].

Si los paquetes que intentan entrar violan las políticas, el sistema puede rechazarlos totalmente o aceptarlos como paquetes de menor prioridad [1]; cuando no hay congestión en el sistema los paquetes no conformes pueden usar el ancho de banda disponible siempre y cuando estos no interfieran en la transmisión adecuada del tráfico conforme. La figura 1.1 [3] ilustra el concepto de *"policing"* en una red que brinda QoS.

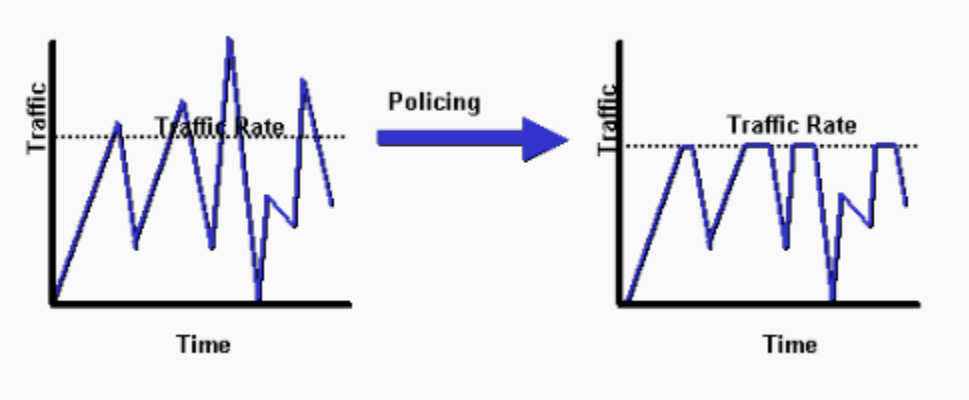


Figura 1.1 Concepto de policing en una red con QoS

Los requerimientos básicos de diseño y operación de cualquier algoritmo de control de tráfico son:

- No se debe descartar o disminuir prioridad de paquetes que cumplen con las políticas establecidas en la configuración de QoS.
- Se deben detectar todos los paquetes que violan las políticas de configuración de la QoS y realizar acciones oportunas como el descarte o disminución de prioridad de aquellos paquetes.
- Debe operar en tiempo real y no retrasar los paquetes admitidos.
- Debe ser lo más sencillo posible para poner en práctica.

1.1.3 Conformación de tráfico

La conformación de tráfico es un mecanismo que se utiliza para controlar la cantidad y el volumen de tráfico enviado a la red, además de la tasa a la cual éste se envía [4]. Para ello se realiza una especificación y descripción de cómo se van a introducir los flujos de

paquetes, proceso denominado modelamiento de tráfico. Si los terminales envían paquetes sin conformarlo o modelarlo, pueden ser detectados como no conformes en el borde de la red y sujetos a descarte [5].

Aunque la vigilancia y la conformación de tráfico parece que realizaran la misma función, hay diferencias conceptuales y técnicas claras. La conformación de tráfico no descarta los paquetes inmediatamente, los almacena en buffers y los envía en determinado tiempo. En contraste en la vigilancia de tráfico simplemente detecta las violaciones a las políticas de QoS, e inmediatamente descarta los paquetes no conformes o les disminuye prioridad.

Entre los métodos de conformación de tráfico, predominan dos [5], [1]: la implementación "*Leaky-Bucket"* y la implementación "*Token-Bucket"*.

1.1.3.1 Leaky bucket

Este algoritmo regula el tráfico a modo de un cubo virtual ("*bucket"*) con goteo ("*leaky"*) tal como se representa en la Figura 1.2 [5]. Esta implementación se utiliza para controlar la tasa a la cual los paquetes se envían a la red, conformándolos en flujos constantes y ordenados [6].

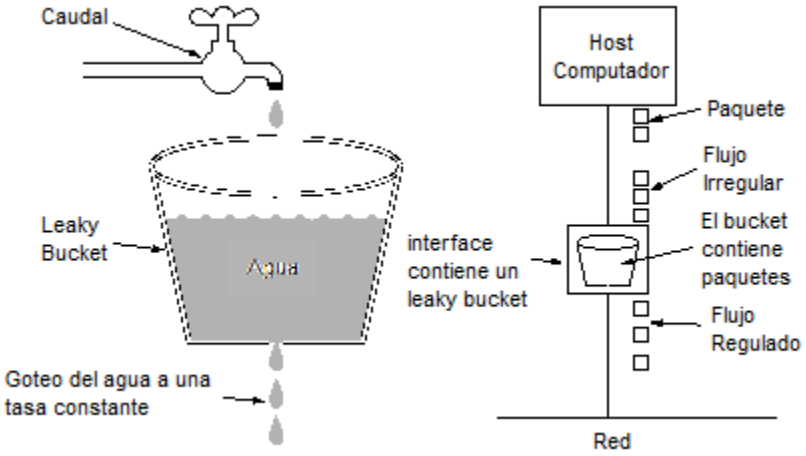


Figura 1.1 Concepto de leaky bucket

En el funcionamiento del algoritmo principalmente se tiene en cuenta la capacidad del cubo y una tasa fija. Cada vez que el cubo tenga paquetes se envían a la red a la tasa fija negociada, todo paquete es introducido en el cubo hasta que este llegue a su tope, cuando el cubo alcance el límite de su capacidad todos los paquetes que lleguen serán declarados como no conformes y se descartarán, así se limita la tasa de transmisión de tráfico al valor de la tasa negociada [1]**.**

Con la implementación de "*leaky bucket"* el flujo de tráfico entrante en la red es predecible y controlado. El principal inconveniente es cuando el volumen de paquetes es ampliamente mayor que el tamaño del contenedor virtual ("*bucket"*), debido a que este descarta paquetes cuando el contenedor está lleno [1].

1.1.3.2 Token Bucket

El algoritmo "*token*[4] *bucket"* en lugar de paquetes contiene "*tokens"*. Este método es más eficiente ya que permite que los flujos de tráfico tengan niveles configurables de transmisión de datos.

Para que se puedan transmitir los datos tienen que existir "*tokens"* en el "*bucket"*, mientras esto pase la fuente puede insertar datos a la tasa deseada [7] tal como se muestra en la figura 1.3 [5]. Cuando se acaban los "*tokens"* los paquetes tienen que esperar a que se generen nuevos *"tokens"*, lo que implica que la tasa de transmisión disminuya a la tasa promedio.

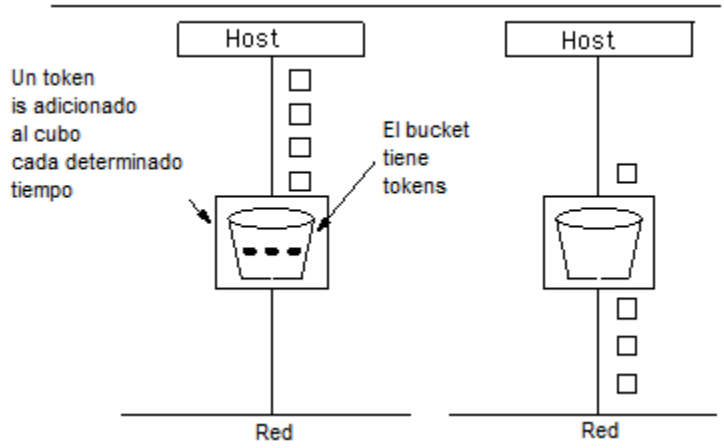


Figura 1.2 Concepto de token bucket

La implementación de un *"token bucket"* necesita de un contador con almacenamiento virtual. Este contador se establece en cero, a medida que llegan paquetes se va incrementando en uno hasta el valor máximo del tamaño de ráfaga u ocupación total del "*bucket"*, posteriormente se va decrementando a medida que cada paquete es aceptado en la red.

Cabe aclarar que existe "*Token Bucket"* de conformación y *"Token Bucket"* de "*Policing"* [5], la diferencia radica en las acciones tomadas cuando no hay *"tokens"*. En la implementación de conformación, los paquetes son almacenados en los buffers y esperan hasta que haya suficientes "*tokens"* para ser transmitidos; en la implementación de "*policing"* el paquete es descartado inmediatamente o reclasificado y remarcado [5].

[4] Es un ticket o testigo que permite a quien lo posea enviar mensajes por la red.

1.1.4 Planificación de paquetes

La planificación de paquetes es un mecanismo para brindar QoS a nivel de red que permite realizar una mejor gestión de tráfico y se encarga del comportamiento temporal de los paquetes desde que ingresan a la cola hasta que salen hacia el próximo salto [8].

En las redes actuales de conmutación de paquetes, el ancho de banda del enlace se comparte entre múltiples fuentes de tráfico, implementando básicamente dos mecanismos: el encolamiento y la planificación. Los enlaces transmiten un paquete al tiempo desde los "*buffers"*[5], como se tienen múltiples paquetes en el "*buffe*r", los algoritmos de planificación definen el plan de transmisión de los paquetes en el enlace.

- Las disciplinas de servicio o planificadores se diseñan para cumplir con requerimientos específicos y tienen unas características básicas como:
- Aislamiento de flujos: Aislar un canal de los efectos indeseables de otros.
- Garantía de *"Throughput"*: Capacidad de garantizar *"throughput"* de corta duración para sesiones libres de error y *"throughput"* de larga duración para todas las sesiones.
- Ecuanimidad y eficiencia: Capacidad de ceder ancho de banda que no esté en uso a estaciones que temporalmente excedan su ancho de banda reservado para el canal. Se debe asegurar ecuanimidad entre sesiones.
- Escalabilidad y complejidad: Baja complejidad, desde el punto de vista de encolamiento y des-encolamiento[6] de un paquete, factor muy importante en el caso de redes inalámbricas que presentan limitaciones de recursos de procesamiento y de energía. Además, el algoritmo debe operar eficientemente a medida que aumenta el número de sesiones compartiendo el canal [8].

La planificación en redes inalámbricas es mucho más compleja ya que la intensidad de la señal depende del estado del medio de transmisión por el cual viaja la información, originando retardos y pérdida de paquetes en la información transmitida.

A continuación se realiza un breve recuento de disciplinas y planificadores de tráfico utilizados en la configuración interna de HTB:

1.1.4.1 First In First Out (FIFO) / First Come First Served (FCFS)

"First In First Out" (FIFO) es una disciplina de encolamiento simple denominada también *"First Come First Served"* (FCFS); su política de encolamiento de primero en entrar primero en salir es una de las más sencillas y empleadas en equipos que no manejan ningún tipo de QoS ya que el tráfico no recibe tratamiento especial [9].

[5]Es una ubicación de memoria en un computador o en un dispositivo digital reservada para el almacenamiento temporal de información digital, mientras que está esperando ser procesada.
[6] Proceso inverso al encolamiento

Los paquetes llegan a una única cola por cada salida de la tarjeta de interfaz de red (NIC), si esta cola está completamente llena los paquetes recién llegados se eliminan hasta que haya un espacio en el *"buffer"* y se pueda aceptar el tráfico entrante, esta propiedad es llamada "*tail dropping"*[7] [10].

Esta disciplina es ideal para servicios como *"best effort"* y no valida en entornos interactivos ya que los flujos no son aislados y la disciplina no toma en cuenta requerimientos específicos como "*delay"* y "*throughput"*. Además el "*tail dropping"* resulta inadecuado para flujos sensibles a retardos, donde los paquetes son descartados cuando superan un umbral de retardo en su transmisión.

1.1.4.2 Weighted Round Robin (WRR)

"Weighted Round Robin" [11] asume que el tamaño promedio del paquete a transmitir es conocido. A Cada flujo se asigna una variable peso[8]**,** la cual depende de la capacidad establecida en el canal.

En WRR cada clase tiene un contador que especifica el número de paquetes que se pueden enviar. El valor del contador inicialmente se fija en el valor del peso asignado a esa clase, después de enviar un paquete el contador de la clase se reduce en uno, cuando el valor del contador o la longitud de la cola llega a cero en todas sus clases, todos los contadores se restablecen a sus valores de peso [12], [13].

Aunque este sistema es fácil de implementar y adecuado para el procesamiento de paquetes, tiene varias desventajas: cuando llega un paquete más grande que el tamaño promedio, puede ser retrasado induciendo un mal funcionamiento del sistema, ya que aquella cola deberá esperar una siguiente ronda para ser transmitida. Además el algoritmo no es adecuado si el tamaño del paquete es desconocido o muy diferente al tamaño promedio, ya que un paquete de gran tamaño puede llegar a consumir más recursos de los que le fueron asignados [14].

1.1.4.3 Deficit Round Robin (DRR)

Debido a las fallas que presenta WRR cuando se tienen paquetes de tamaños variables, se creó el planificador de tráfico "*Deficit Round Robin"* [11] la cual es una extensión de SFQ.

Esta disciplina realiza un seguimiento a las diferentes colas presentes en la red durante una ronda, cuando alguna cola no puede enviar un paquete porque el tamaño de este es más grande que el asignado en su correspondiente turno, la cantidad de fragmentos de cola que falta se añade en la siguiente ronda. Para esto, cada cola cuenta con unas variables llamadas "*Quantum"*[9] y *"déficit"*.

---

[7] *"Tail dropping"* es un sistema de gestión de colas que decide cuando descartar paquetes
[8] Capacidad del canal asignada a un flujo
[9] Indica la cantidad de bits a enviar en cada turno de transmisión

Al principio de cada ronda el valor de la variable "*Quantum"* es igual al valor del Contador "*Déficit"* de cada cola. Entonces, si el tamaño del paquete HOL (cabecera de línea) de la cola es menor o igual al "*quantum"*, este paquete se procesa y el contador "*Déficit*"[10] se decrementa en el tamaño del paquete, si el tamaño del paquete es mayor al tamaño del "*quantum"* entonces el paquete tendrá que esperar otra ronda para ser servido y el contador "*déficit"* se incrementará en el tamaño del "*quantum"*. En caso de existir una cola vacía el contador "*Déficit"* se decrementa a cero.

1.1.5 Planificación de paquetes en entornos inalámbricos

La planificación de paquetes en una red inalámbrica es más complicada que en redes cableadas ya que el canal es influenciado por factores adicionales que un mecanismo de planificación debe tener en cuenta [15]:

- Errores dependientes de la localización, el más común de ellos debido a la múltiple propagación.
- Mayor probabilidad de errores de transmisión causada generalmente por el ruido y la interferencia en el canal de radio.
- Incremento y decremento dinámico del número de estaciones.
- Interferencia con otras frecuencias de radio.

En el caso de redes cableadas, debido a que no se presentan fluctuaciones considerables en la intensidad de la señal, es común que todos los usuarios tengan buenas condiciones de transmisión. En redes inalámbricas debido a los factores presentes en el canal no siempre todos los usuarios obtienen buena señal; para enfrentar este problema se hace necesario un método de compensación inalámbrica [16].

Los mecanismos de planificación de paquetes en redes inalámbricas más utilizados son: el encolamiento equitativo inalámbrico idealizado ("*Idealized Wireless Fair Queuing", IWFQ*) [17], [18], que es una adaptación de ("*Weighted Fair Queuing"* WFQ) [19] para manejar los errores dependiendo de la localización de la estación. Otros mecanismos muy utilizados en la planificación de paquetes inalámbricos son: ("*Wireless Packet Scheduling", WPS*) [17] que es una aproximación al mecanismo IWFQ que utiliza WRR y asume que todos los paquetes tienen el mismo tamaño, "*Channel-Condition Independent Packet Fair Queuing",* CIF-Q) [19], [20] que es una adaptación del planificador cableado ("*Star-time Fair Queueing",* SFQ) [21], (*"Wireless Fair Service",* WFS) [22], [17] que es una adaptación de ("*Weighted Fair Queuing"* WFQ) [23] creado con el fin de manejar más eficientemente flujos sensibles a retardos. Todos estos mecanismos utilizan implícitamente compensación inalámbrica.

[10] Este contador se encarga de almacenar el valor del *quantum* obtenido al transmitir un paquete y que permitirá posteriormente enviar paquetes de longitud variable.

1.1.6 Algoritmos jerárquicos de enlace compartido

Los algoritmos jerárquicos de enlace compartido permiten una planificación de paquetes basada en una estructura de enlace compartido jerárquico. Estos algoritmos comparten el exceso de ancho de banda, que en vez de ser distribuido proporcionalmente se distribuye de acuerdo a la configuración que previamente realiza el administrador. Esta sección describe dos de los principales algoritmos de enlace compartido: "*Class Based Queuing"* (CBQ) y *"Hierarchical Token Bucket"* (HTB).

1.1.6.1 Class Based Queuing (CBQ)

"*Class Based Queueing"* CBQ [24], [25] es un mecanismo de control de tráfico que divide el ancho de banda de un enlace entre diferentes tipos de flujos para realizar una gestión más eficiente de los recursos. Este algoritmo es adecuado para compartir enlaces y aplicaciones con consideraciones de tiempo real, ya que aísla los diferentes tipos de tráfico separándolos en diferentes clases y tratándolos de acuerdo a sus requerimientos.

CBQ ofrece una estructura jerárquica que contiene diferentes clases de tráfico. La estructura jerárquica que se conforma en cada enlace representa la repartición del total de ancho de banda entre las clases (clase padre, clases intermedias y clases *"leaf"*) como se muestra en la figura 1.4 [24].

Todos los paquetes que ingresan se asocian a una clase *"leaf"*, como los nodos D, E, B o C de la Figura 1.4. Los nodos intermedios no transmiten paquetes como podría ser el caso del nodo A de la Figura 1.4, estos nodos sólo dan información de cómo se mantiene o distribuye el ancho de banda sobrante y guardan estadísticas de los nodos pertenecientes a las clases *"leaf"*, ya que los paquetes que terminan en las clases "leaf" necesariamente tienen que pasar por las clases intermedias.

Una característica principal de este planificador es que cada clase siempre recibe el ancho de banda mínimo que se le asigno sin importar que tan congestionada este la red, esto es una garantía para clases con mucha demanda.

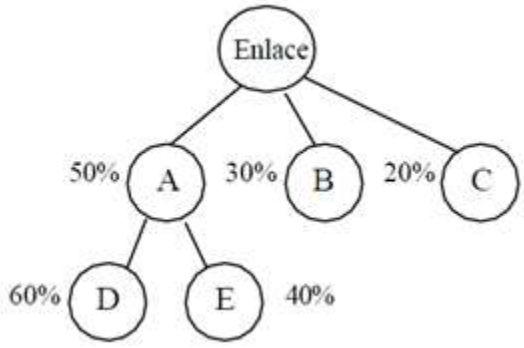


Figura 1.3 Ejemplo de configuración con CBQ

Otra característica importante de CBQ es el préstamo de ancho de banda entre diferentes clases; el ancho de banda que alguna clase no usa puede ser usado por otra clase que necesite más ancho de banda del asignado si las políticas establecidas lo permiten.

1.1.6.2 Hierarchical Token Bucket (HTB)

El *"Token Bucket Jerárquico"* (HTB) [26] es una evolución de CBQ, posee menos variables de funcionamiento y la sintaxis de codificación es mucho más simple, estas características le brindan mayor escalabilidad y flexibilidad.

Es una disciplina de encolamiento multibanda porque permite generar varias colas "paralelas" en un solo enlace físico, simulando enlaces más lentos enviando diferentes clases de tráfico por cada uno, es jerárquica porque permite la clasificación de paquetes por clases realizando una subdivisión interna de su estructura que puede ser configurada por el usuario, útil cuando hay diferentes tipos de tráfico que necesitan diferente tratamiento.

HTB define tres tipos de clases: *"root"*, intermedia y *"leaf"*, como se muestra en la figura 1.5 [26]. La clase *"root"* representa el enlace, las clases intermedias se asocian con usuarios y las clases *"leaf"* corresponden a las aplicaciones que generan el tráfico; cuando se recibe un paquete de datos en la interface de red el planificador recibe una petición para desencolar; con base en la marca que establece el firewall, la qdisc raíz se encarga de enviarle el paquete de datos a algunas de las clases.

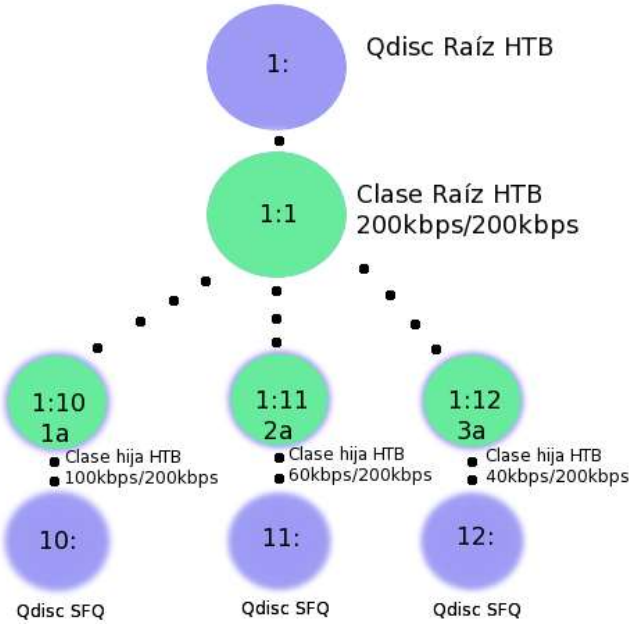


Figura 1.4 Estructura de distribución de ancho de banda en HTB

Cada clase está caracterizada por su tasa actual R, tasa asegurada AR, tasa máxima CR, prioridad P y quantum Q. En la clase principal el valor para AR y CR debe ser el mismo.

El objetivo de HTB es proporcionar los recursos mínimos asignados a cada clase; cuando las clases demandan menos recursos que los asignados, la parte que no se utiliza (el ancho de banda de exceso) se distribuye entre las clases que lo soliciten [27].

Para gestionar el control de recursos se utiliza un mecanismo denominado *"token buckets"* internos que originan el nombre de este algoritmo. Cada vez que hay un *"token"* se le permite a uno de los paquetes que está esperando, ser enviado (ver figura 1.6) [27].

Se pueden presentar tres posibilidades de transmisión dependiendo del ritmo de llegada de los *"tokens"*:

- Si los paquetes llegan al mismo ritmo de los *"tokens"* estos irán saliendo inmediatamente.
- Si los paquetes llegan más rápido que los *"tokens"* deberán esperar un tiempo hasta que haya *"tokens"* disponibles, si esta situación persiste se empezarán a descartar paquetes, con lo cual se limita el ancho de banda.
- Si los paquetes llegan más lentos que los *"tokens"* automáticamente a los paquetes se le asignarán *"tokens"*, los *"tokens"* restantes se irán acumulando por si en determinado tiempo hay alguna aceleración en la llegada de los paquetes.

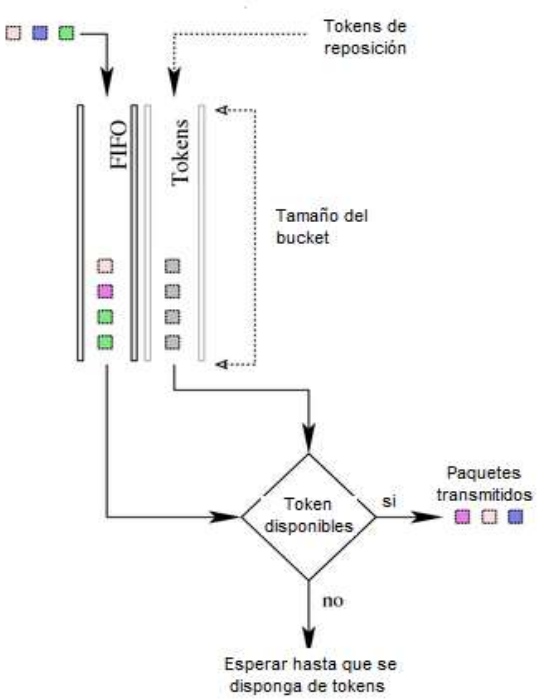


Figura 1.5 Proceso de transmisión de paquetes en HTB

La transmisión de los paquetes se controla mediante el algoritmo *"Deficit Round Robin"* [28]. Si la clase no ha superado su límite de AR puede transmitir sin condiciones, en el

caso que sobrepase el AR pero esté por debajo de su CR solamente puede transmitir si puede tomar prestado el ancho de banda (*"tokens"*) de otras clases. La cantidad de ancho de banda ofrecida a cada clase es proporcional al parámetro Q, mientras que el parámetro P especifica el orden con el cual se ofrecen los recursos.

HTB puede coexistir con colas que sigan disciplinas diferentes, lo que permite ejercer control sobre parámetros adicionales, al igual que hace posible ajustar los sistemas de planificación de tráfico hasta satisfacer las necesidades del administrador de red.

HTB también utiliza el concepto de colores, para identificar el modo de una clase. Con base en los parámetros: tasa actual (R), "*ceil"* (CR) y "*rate"* (AR) se definen los tres colores:

- **Rojo**: R>CR, no puede transmitir.
- **Amarillo**: R≤CR y R>AR, puede tomar ancho de banda prestado.
- **Verde**: Otro caso, puede transmitir.

1.2 VOZ SOBRE IP

La tecnología de VoIP[4] permite digitalizar la voz y empaquetarla para que sea enviada por una red de datos usando el protocolo IP, las funciones principales que debe realizar un sistema de VoIP son:

1. Digitalización de la voz
2. Paquetización de la voz
3. Enrutamiento de los paquetes

La transmisión de voz sobre IP tiene ciertas ventajas frente a la transmisión de voz en redes de circuitos conmutados, como el mejor aprovechamiento del ancho de banda, ya que VoIP no utiliza circuitos físicos para la conversación, sino que envía varias conversaciones sobre el mismo circuito virtual. Cuando se producen silencios en una conversación, los paquetes de otras conversaciones son transmitidos [29].

La principal desventaja de la transmisión de voz a través de redes de paquetes es que la calidad de la conversación puede ser deficiente, debido a que las redes de transmisión de datos están diseñadas para retransmitir paquetes en caso de pérdida, lo que no es apropiado para la transmisión de voz. Además es posible que los paquetes de voz no lleguen al destino en el mismo orden en el que fueron transmitidos a causa de que estos pueden tomar diferentes rutas entre la fuente y el destino.

Para garantizar interoperabilidad entre redes de datos y redes telefónicas o entre equipos de diferentes operadores, es necesario un grupo de protocolos, entre los más conocidos están H.323 y SIP. H.323 es una serie de normas de la ITU para comunicaciones multimedia entre terminales, equipos y servicios. Estas normas no garantizan calidad de servicio, aunque la ITU ha publicado recomendaciones sobre este tema, como G.113 [29].

[4] Voz para ser transmitida por redes IP

1.2.1 IAX2 (Inter-Asterisk eXchange v2)

IAX es un protocolo simple y robusto diseñado para interconectar servidores asterisk, soporta la mayoría de flujos multimedia, sin embargo está optimizado para VoIP, donde la baja sobrecarga y el bajo consumo de ancho de banda son prioritarios. Esto lo hace más eficiente que protocolos complejos que soportan servicios más allá de las necesidades actuales.

IAX es un protocolo binario, diseñado para reducir la sobrecarga, especialmente en flujos de voz, por ejemplo cuando se transmite voz comprimida a 8 kbit/s con 20 mili segundos de paquetización IAX2 agrega 20% de sobrecarga a cada paquete de voz, mientras RTP añade 60% de sobrecarga [30].

El enfoque de diseño básico de IAX2 para la señalización y comunicación de varios flujos sobre una única asociación UDP entre dos hosts, se logra mediante el uso de un único puerto (4569) para todos los tipos de tráfico IAX. Los codecs mas utilizados para compresión de voz son G729, GSM, ILBL/Speech, G722/G723, G711a/G711u.

1.3 CALIDAD DE SERVICIO EN VOIP

1.3.1 Medidas de QoS

La medida tradicional de la percepción de calidad de un usuario es el puntaje promedio de opinión (MOS "*Mean Opinion Score"*) definido en "*Methods for Subjective Determination of Voice Quality"* en ITU-T P.800, un panel experto de oyentes tasan muestras de voces preseleccionadas de codecs y algoritmos compresores de voz bajo condiciones controladas, un puntaje de MOS puede tomar valores desde 1 (malo) hasta 5 (excelente). El algoritmo de modulación por pulso codificado (PCM *"Pulse Code Modulation"*) (ITU-T G.711) obtuvo un puntaje MOS de 4.4 [31].

ITU-T G.107 presenta un modelo matemático, conocido como E-model evalúa como afectan los parámetros de la red la calidad de la conversación, usando factores de deterioro más objetivos [1]. TIA/EIA TSB116 provee una comparación de valores tasados del E-model (R) y puntajes MOS [32].

1.3.2 Factores que afectan la QoS en VoIP

Los factores de deterioro más importantes para VoIP que se usan como parámetros de entrada de E-model son:

- Delay (retardo)
- Packet Loss (pérdida de paquetes)

- Speech Compression (compresión de la conversación)
- Echo (eco)

El objetivo de la calidad de servicio es mantener estos factores dentro de unos límites aceptables. Los requerimientos a cumplir para voz sobre IP son:

- Un retardo entre 150ms y 200ms afecta la calidad de la conversación [32].
- La recomendación de la ITU-T G113 [33] brinda guías del impacto de la pérdida de paquetes sobre la calidad de servicio de VoIP, en términos del factor de degradación de equipo Ie (Equipment impairment), un valor de Ie mayor significa un deterioro más severo,

  Para el códec G.711 el factor de deterioro es de 35 cuando hay un 2% de pérdida de paquetes, sin embargo este valor disminuye en G.711 con PLC (Packet Loss Concealment), los codecs G.729 y G.723.1 tienen un factor de deterioro diferente a cero aún cuando no hay pérdida de paquetes como se muestra en la tabla 1.1.

Códec	Ie(0 % de pérdidas)	Ie (2 % de pérdidas)	Ie (5 % de pérdidas)
G.711 sin PLC	0	35	55
G.711 con PLC	0	7	15
G.729ª	11	19	26
G.723.1	15	24	32

Tabla 1.1 Valores de Ie y pérdida de paquetes

- El jitter máximo no debe sobrepasar los 30 ms.

2. FUNDAMENTOS PARA EL DESARROLLO DE SISTEMA HTB ADAPTABLE

En este capítulo se presentan características del estándar 802.11, los esquemas de modulación utilizados, conceptos sobre la modulación adaptativa y la influencia de la SNR sobre el rendimiento de las redes inalámbricas, como soporte teórico del desarrollo del monitor de ancho de banda del canal inalámbrico. Posteriormente se analiza el código fuente de HTB, con el fin de entender su funcionamiento; y por último se realiza una descripción del hardware y diferentes tipos de software.

2.1 MODULACIÓN EN ESTANDAR 802.11

A continuación se realiza una breve descripción de algunas de las modulaciones de transmisión utilizadas en redes inalámbricas WiFi.

2.1.1 Modulación BPSK (Binary phase-Shift keying)

La modulación[5] BPSK es la más sencilla de las modulaciones por fase, ya que tiene dos fases de salida en la misma frecuencia de portadora y cada fase representa un bit de información [34], es la modulación que ofrece menor velocidad de transmisión, pero mayor inmunidad al ruido, ya que la distancia entre símbolos es máxima (180º), la ecuación 2.1 representa la modulación en fase:

$$S(t) = Am(t)\cos(2\pi f_c t) \qquad 2.1$$

Donde A es la amplitud de la portadora y fc su frecuencia.

m(t) es igual a 1 para el bit 1 y -1 para el bit 0, es decir que el 1 lógico se representa con 0º y el 0 lógico se representa con 180º (figura 2.1 [34]) en la misma frecuencia de [11]portadora [34] [35[5]].

[5] Conjunto de técnicas para transportar información sobre una onda portadora.

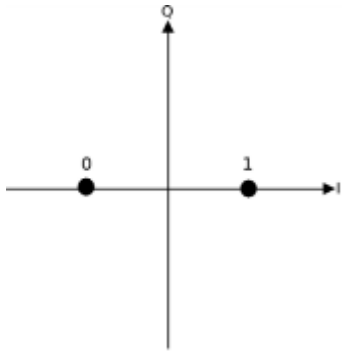


Figura 2.1 Constelación de la modulación BPSK

En la figura 2.2 [36], se presenta la fase de salida con relación al tiempo para una señal modulada BPSK.

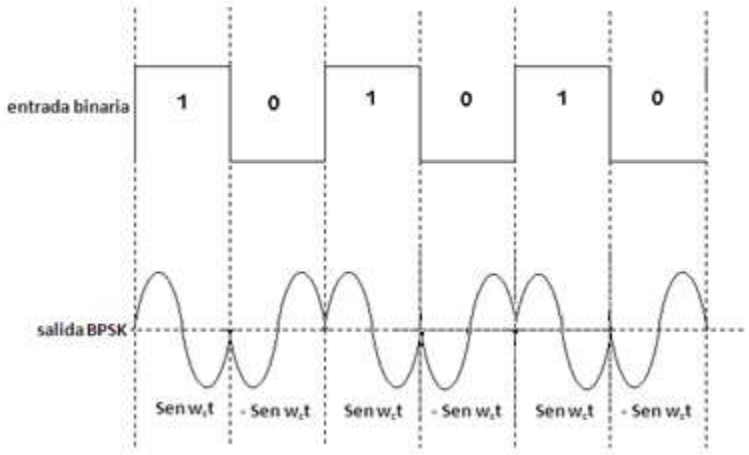
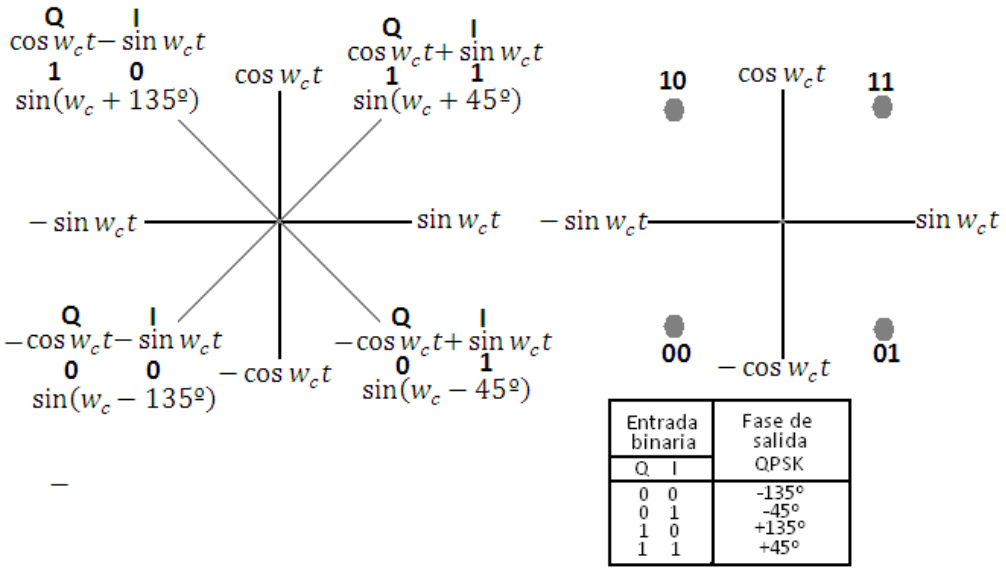


Figura 2.2 Fase de salida BPSK

2.1.2 Modulación QPSK (Quadrature phase Shift Keying)

QPSK es una esquema de modulación de 4 estados, donde son posibles 4 fases de salida para una única frecuencia de portadora, son necesarias 4 entradas distintas. Para producir dichas entradas se necesitan 2 bits, que producen las combinaciones 00, 01, 10,11, los bits de entrada al modulador QPSK se ordenan en grupos de 2 bits [36].

En la figura 2.3 [36] se aprecia que cada uno de los grupos de dos bits está representado por una señal con la misma amplitud y fase diferente (desfase de 90º), es decir que la información está codificada únicamente en la fase de la señal de salida.



Entrada binaria Q I	Fase de salida QPSK
0 0	-135°
0 1	-45°
1 0	+135°
1 1	+45°

Figura 2.3 Codificación de QPSK

2.1.3 Modulación QAM (Quadrature Amplitude Modulation)

La modulación de amplitud en cuadratura (QAM) es un esquema donde la información está representada tanto en la amplitud como en la fase de la señal transmitida, esto se consigue desfasando 90 grados una misma portadora de un mensaje a otro [37], [38], disminuyendo las interferencias entre las señales. La señal QAM es el resultado de sumar dos señales moduladas en desplazamiento de amplitud, representadas en la ecuación 2.2.

$$a_n \cos(wt) + b_n \sin(wt) \qquad 2.2$$

Las amplitudes de las señales moduladas en amplitud toman los valores discretos an y bn correspondientes a los N estados de la señal moduladora.

Una señal QAM es la combinación de la modulación en amplitud (ASKn,m) y modulación en fase (PSKn,m) de una única portadora como se representa en la ecuación 2.3.

$$A_n(\cos wt) + B_m(\sin wt) = A_{n,m} \cos(wt - H_{n,m}) \qquad 2.3$$

Donde An,m es la señal modulada en amplitud y cos(wt – Hn,m) está modulada en fase.

Se han desarrollado esquemas de modulación como 16 QAM, 64 QAM y 256 QAM, donde usan 16, 64 y 256 combinaciones de amplitud y fase. Sin embargo los niveles altos de modulación son más sensibles al ruido, ya que los símbolos están más cercanos unos de otros, por esta razón es más fácil que el receptor confunda un símbolo con otro símbolo cercano, incrementando la probabilidad de error [37].

Por lo tanto si se desea una probabilidad de error menor es necesaria una relación señal a ruido (SNR)[12] mayor, a medida que el orden de modulación aumenta. De la misma manera cuando la relación señal a ruido disminuye se requiere un orden de modulación menor, menos eficaz, pero menos sensible al ruido.

2.1.4 Modulación 16 QAM

En cada cuadrante existen 3 valores de fase y amplitud diferentes, y cada punto de la constelación representa 4bits, codificados con base en código gray, por lo que en un punto solo cambia un bit con respecto a sus puntos adyacentes (figura 2.4) [37], de esta forma se reduce la probabilidad de error, ya que si la señal se distorsiona solo se tendrá un bit erróneo.

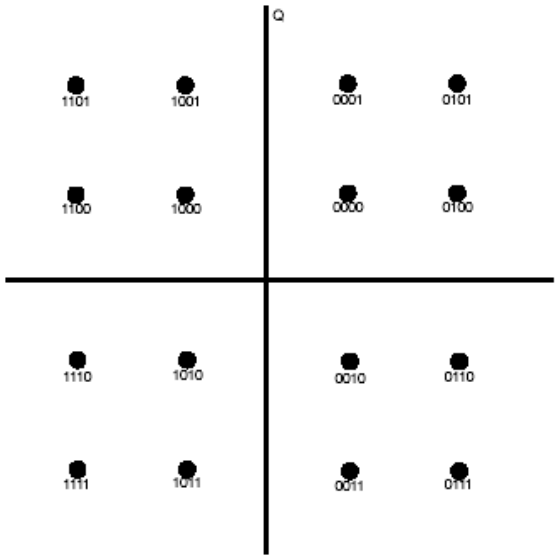


Figura 2.4 Constelación de 16 QAM

2.2 MODULACIÓN ADAPTATIVA

Diferentes órdenes de modulación permiten enviar más bits por símbolo y así alcanzar altos *"throughputs"* o mejorar la eficiencia espectral. Sin embargo, cuando se usa una modulación como 64 QAM, es necesaria una mejor relación señal a ruido (SNR), para superar cualquier interferencia y mantener un nivel seguro de la tasa de error de bit (BER[13]) [39], [40].

El uso de modulación adaptativa permite a los sistemas inalámbricos escoger esquemas de modulación de orden superior dependiendo de las condiciones del canal. En la figura

[12]Se define como el margen que hay entre la potencia de la señal que se transmite y la potencia del ruido que la corrompe.
[13] Medida de la exactitud de transmisión que representa los bits recibidos en error a bits enviados.

2.5 [41], se presenta el cambio en el esquema de modulación dependiendo de la distancia de la estación base al cliente. Cuando se incrementa esta distancia, se cambia a una modulación más lenta (como BPSK) para mantener un nivel de BER aceptable, pero cuando el cliente está cerca de la estación base se utilizan modulaciones de orden alto como 64 QAM para incrementar el *"throughput"* [39]. Adicionalmente la modulación adaptativa permite que el sistema supere desvanecimientos y otras interferencias.

Figura 2.5 Modulación adaptativa

QAM y QPSK son técnicas de modulación usadas en tecnologías inalámbricas como IEEE 802.11 (Wi-Fi), IEEE 802.16 (WiMAX) y 3G (WCDMA/HSDPA) [42]. El uso de la modulación adaptativa permite a las tecnologías inalámbricas optimizar el *"throughput"*, mejorando la velocidad de transmisión mientras se cubren largas distancias.

La modulación adaptativa es la forma de mejorar la relación entre la eficiencia espectral y la tasa de error de bit, pero es necesario decidir cuál sistema de modulación es el más efectivo para las condiciones del canal con ruido Gaussian blanco aditivo (AWGN, *"Additive White Gaussian Noise"*), con el fin de mantener un nivel de BER aceptable y la mejor eficiencia espectral posible, para esto es preciso establecer un rango de SNR para cada esquema de modulación del sistema.

Ahora se considera el desempeño de la BER empleando tres esquemas de modulación, las ecuaciones 2.4, 2.5 y 2.6 representan la probabilidad de error de bit en un canal con ruido blanco gaussiano [43].

$$P_{QPSK(\gamma)} = Q(\sqrt{\gamma}) \qquad 2.4$$

$$P_{16QAM}(\gamma) = \frac{1}{4}\left[Q\left(\sqrt{\frac{\gamma}{5}}\right) + Q\left(3\sqrt{\frac{\gamma}{5}}\right) + \right] + \frac{1}{2}Q\left(\sqrt{\frac{\gamma}{5}}\right) \qquad 2.5$$

$$P_{64QAM}(\gamma) = \frac{1}{12}\left[Q\left(\sqrt{\frac{\gamma}{21}}\right) + Q\left(3\sqrt{\frac{\gamma}{21}}\right) + Q\left(5\sqrt{\frac{\gamma}{21}}\right) + Q\left(7\sqrt{\frac{\gamma}{21}}\right)\right] + \frac{1}{6}Q\left(\sqrt{\frac{\gamma}{21}}\right) + \frac{1}{6}Q\left(3\sqrt{\frac{\gamma}{21}}\right) + \frac{1}{12}Q\left(5\sqrt{\frac{\gamma}{21}}\right) + \frac{1}{12}Q\left(7\sqrt{\frac{\gamma}{21}}\right) + \frac{1}{3}Q\left(\sqrt{\frac{\gamma}{21}}\right) + \frac{1}{4}Q\left(3\sqrt{\frac{\gamma}{21}}\right) - \frac{1}{4}Q\left(5\sqrt{\frac{\gamma}{21}}\right) - \frac{1}{6}Q\left(7\sqrt{\frac{\gamma}{21}}\right) + \frac{1}{6}Q\left(9\sqrt{\frac{\gamma}{21}}\right) + \frac{1}{12} + Q\left(11\sqrt{\frac{\gamma}{21}}\right) - \frac{1}{12}Q\left(13\sqrt{\frac{\gamma}{21}}\right) \qquad 2.6$$

En las ecuaciones (2.4), (2.5) y (2.6) γ es la relación señal a ruido, y Q es la función de densidad de probabilidad gaussiana y está dada por la ecuación 2.7:

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{oo} e^{-\frac{x^2}{2}} dx \qquad 2.7$$

Usando las ecuaciones (2.4), (2.5) y (2.6) se pueden obtener gráficas de desempeño de BER contra SNR [44], para modulación QPSK, 16QAM y 64QAM (figura 2.6) [44].

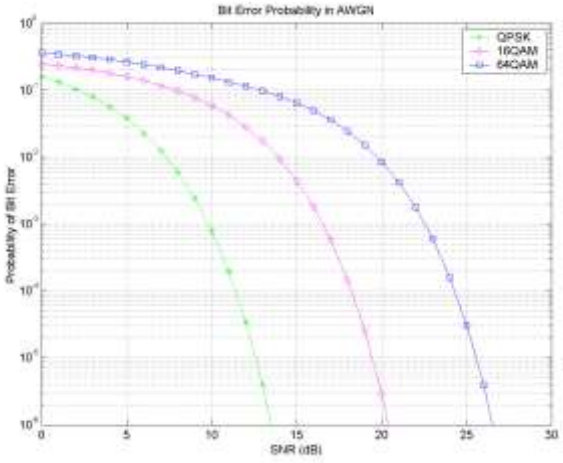


Figura 2.6 BER en función de SNR para QPSK, 16 QAM y 64 QAM

En la figura las curvas de izquierda a derecha representan la BER de QPSK, 16 QAM, y 64QAM en un canal con ruido blanco. Para decidir el esquema de modulación apropiado, es necesario decidir un punto de operación o una BER. Normalmente se escoge una BER de 10^{-3} como punto de operación, esto significa que el sistema tratará de mantener un nivel de BER más bajo que el nivel de operación con el esquema de modulación de mayor eficiencia espectral siempre que sea posible.

Con el punto de operación definido y las gráficas de BER se establecen los siguientes rangos de SNR para cada esquema de modulación:

QPSK	SNR < 17dB
16QAM	17 <= SNR <= 23dB
64QAM	SNR > 23 dB

Tabla 2.1 Rangos de SNR para esquemas de modulación

Estos valores se obtienen así: en un punto de operación de BER de 10^{-3}, no hay ningún esquema de modulación que proporcione el desempeño deseado con un SNR debajo de 10dB. Por lo tanto, se escoge QPSK por ser el esquema más robusto, entre 10 y 17 dB este es el único que brinda un nivel por debajo de 10^{-3}, entre 17 y 23 dB 16 QAM ofrece la BER deseada con una mejor eficiencia espectral que QPSK. Y con un SNR mayor a 23 dB, 64QAM brinda una mejor eficiencia espectral mientras provee el nivel de BER deseado.

2.3 ANÁLISIS DEL CÓDIGO FUENTE HTB

El código utilizado para realizar las modificaciones en el algoritmo HTB, está escrito en lenguaje C, puede descargarse de [26] y se llama htb3.6_2.4.17.

Además de los parámetros mencionados en capítulos anteriores es importante definir algunos que se encuentran en el código HTB:

El nivel: determina su posición en la jerarquía. Las clases hojas tienen nivel 0, las clases raíz -1 y cada clase interna tiene nivel uno menos de su clase padre [45].

El modo: el modo de la clase es un valor que puede calcularse de **R** (tasa real), **AR** (tasa asegurada), **CR** (tasa *"ceil"*). Los posibles modos son:

- **Red**: **R** > **CR**
- **Yellow**: **R** <= **CR** y **R** > **AR**
- **Green** en otro caso.

Estos modos se definen en el código en la estructura de enumeración [46]:

```
enum htb_cmode {
HTB_CANT_SEND,     /* la clase no puede enviar y no puede prestar */
HTB_MAY_BORROW,   /* la clase no puede enviar pero puede prestar */
HTB_CAN_SEND      /* la clase puede enviar*/
+};
```

Las clases de tráfico en el código HTB se representan con una estructura llamada **htb_class** en la cual se tienen estructuras internas para definir parámetros generales como la topología del árbol de clases.

```
+    struct htb_class *parent;      /* padre de la clase */
+    struct list_head hlist;        /* classid hash list item */
+    struct list_head sibling;      /* lista de item hermano */
+    struct list_head children      /* lista de hijas*/
```

Las estructuras y variables definidas dentro de la clase, más importantes para el proyecto son:

```
+    struct qdisc_rate_table *rate; /* tabla de tasa de la clase */
+    struct qdisc_rate_table *ceil; /* tasa ceiling (limites de */
                                    /* prestamos)*/
+    long buffer,cbuffer;           /* token bucket depth/rate */
+    long mbuffer;                  /* tiempo maximo de espera */
+    long tokens,ctokens;           /* actual numero de tokens */
+    psched_time_t t_c;             /* checkpoint time */

+    /* contadores de medida de tasa */
+    unsigned long rate_bytes,sum_bytes;
+    unsigned long rate_packets,sum_packets;
```

Se consideran estas estructuras por ser parámetros del token bucket e intervienen en la tasa de transmisión de las clases de tráfico.

El planificador HTB se declara en la estructura **htb_sched**, donde está definida una lista de alimentación **htb_sched::row,** compuesta por varios slot necesarios para desencolar paquetes, hay un slot para cada prioridad por nivel, los que se representan con los cuadros en azul y rojo (rojo-prioridad alta, azul-prioridad baja) a la derecha del árbol de clases en la figura 2.8 [45].
Las clases internas también tienen slots internos de alimentación, los que se representan por los cuadros rojos y azules debajo de las clases A y B.
El slot blanco pertenece a la cola de espera **wait_pq,** esta cola guarda una lista de las clases con modo rojo o amarillo en ese nivel. Las clases se clasifican en un límite de tiempo **pq_key** definido en la estructura **htb_class,** dentro de este límite las clases cambiarán de modo**.**

El tratamiento de los paquetes en la cola de entrada se resume en la figura 2.7 [47].

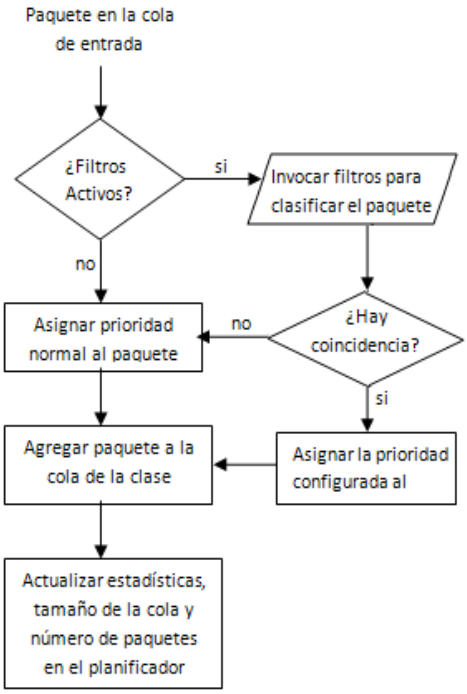


Figura 2.7 Diagrama de flujo del funcionamiento del planificador HTB

Cuando un paquete entra al planificador, se agrega a la cola de entrada del planificador con el método **htb_enqueue** el cual se encarga de clasificar los paquetes en su clase correspondiente con el método **htb_classify**, este método retorna la clase en la cual el paquete debe ser encolado, retornará NULL si el paquete debe ser descartado o -1 si el paquete se debe desencolar directamente [46].

Para la clasificación de los paquetes se examinan los filtros en qdisq y en los nodos internos, si después de examinar los filtros no se encuentra una clase apropiada para el paquete, se intenta usar la clase *"default"*. Cuando se desencola un paquete utilizando el método **htb_dequeue_tree,** se actualizan las estadísticas, tamaño de la cola, y el número de paquetes en el planificador con el método **htb_charge_class**.

A continuación se describe el procedimiento para desencolar un paquete de la cola de la clase de tráfico.

En la figura 2.8 [46] se observa que en el árbol de la izquierda no ha llegado ningún paquete a las clases, por lo que cualquier clase podría enviar paquetes, en el árbol de la derecha han llegado paquetes a las clases C y D, así que es necesario activar esas clases con el método **htb_activate** y asignar los slot apropiados (ya que las dos clases están en verde), luego se desencola el paquete de la clase D con el método **htb_dequeue** ya que tiene mayor prioridad, y posteriormente se desencola el paquete de la clase C.

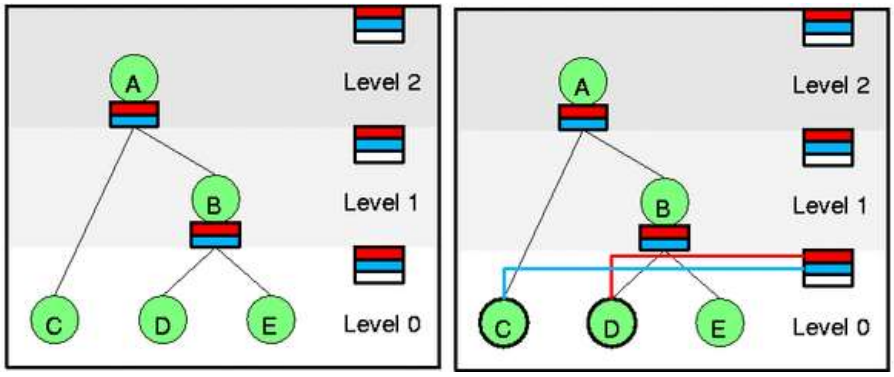


Figura 2.8 Arboles de clases sin paquetes y desencolado de paquetes desde C y D

Una vez el paquete se desencola desde la clase D con el método **htb_dequeue_tree** se actualizan las estadísticas, tamaño de la cola, y el número de paquetes en el planificador con el método **htb_charge_class**, para que la clase D verifique su tasa de transmisión y cambie su modo a amarillo (si es necesario) con el método **htb_change_class_mode.** Eventualmente la clase D recuperará el modo verde por lo que es necesario agregarla a la cola de evento de prioridad con el método **htb_add_to_wait_tree**

Debido a que la clase D ha cambiado a modo amarillo es necesario removerla de la lista de alimentación con los métodos **htb_deactive_prios** y **htb_remove_class_from_row** y agregarla a la lista interna de alimentación de la clase B. Se agrega la clase B a la lista de alimentación con los métodos **htb_active_prios** y **htb_add_class_to_row** como se muestra en el árbol de la izquierda en la figura 2.9 [46].

Suponiendo que se han desencolado paquetes de las clases C y esta ha cambiado a modo rojo, también que las clases C y B han desencolado paquetes cambiando a modo amarillo, entonces es necesario quitar a la clase B de la lista de alimentación y agregarla a la lista interna de alimentación de la clase A y agregar A a la lista de alimentación como se muestra en el árbol de la derecha en la figura 2.9 [46].

De esta forma se ilustra como la lista de alimentación y las listas internas de las clases crean una manera de pedir prestado tasas de transmisión, ya que cada vez que se cambia de modo verde a amarillo se debe remover la clase de la lista de alimentación y agregarla a la lista interna de alimentación de la clase padre. Lo que permite que la clase en modo amarillo transmita por medio de su clase padre con una tasa mayor que la permitida.

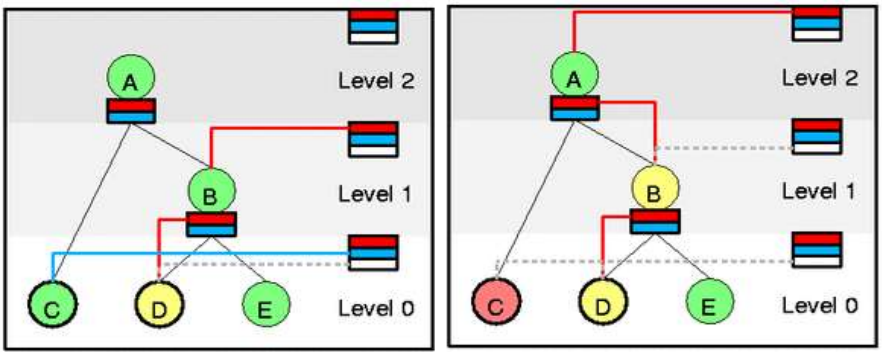


Figura 2.9 Cambio de modo de las clases C, D y B

2.4 DESCRIPCIÓN DEL SOFTWARE Y HARDWARE

2.4.1 Hardware

Para el desarrollo del proyecto se escogió el enrutador inalámbrico Linksys WRT54GL ya que soporta *"software"* con licencia GPL, permite modificar el *"firmware"*, cambiar funcionalidades del enrutador, además es uno de los enrutadores inalámbricos más utilizados por su estabilidad, buen desempeño y cuenta con la documentación necesaria para el desarrollo del proyecto.

2.4.1.1 Linksys WRT54GL

El modelo WRT54GL [48] (figura 2.10) es uno de los más completos y confiables, es una variación del enrutador linksys WTG54G, la principal característica de este equipo es que el código fuente del *"firmware"* fue liberado, los usuarios pueden modificarlo para añadir o modificar funcionalidades, y cambiar cualquier forma de funcionamiento del enrutador. Actualmente existen multitud de proyectos de firmwares de terceros, compatibles con este enrutador [49] entre ellos:

- DD.WRT
- Earthlink (IPv6 Firmware)
- Freifunk
- OpenWRT
- HyperWRT

Figura.2.10 Enrutador Linksys WRT54GL ver. 1.1

La versión 1.1 del WRT54GL tiene dos antenas externas conectadas al dispositivo a través de conectores de polaridad inversa y un chip controlador inalámbrico Broadcom, una velocidad de CPU de 200Mhz que se puede incrementar a 250 Mhz, 16Mb de RAM, 4MB de memoria flash para almacenamiento del *"firmware"* [49]. Este equipo cuenta con tres interfaces de red:

- Para enlazar con el operador: **WAN**
- Un "*switch"* de 4 puertos: **LAN**
- Interfaz Wireless: **WLAN**

La potencia de transmisión de la interfaz Wireless puede llegar hasta los 18dBm y es compatible con los protocolos 802.11, 802.11b, 802.11g y 802.11u.

2.4.2 Software

En el enrutador WRT54GL se pueden cargar *"firmwares"* desarrollados por terceros para añadir nuevas funcionalidades. Las distribuciones probadas están basadas en el *"kernel"* de Linux, por esta razón existen diversas versiones del *"firmware"* para cada modelo del enrutador linksys.

Para seleccionar el *"firmware"* para el desarrollo del proyecto se consideró el tamaño del archivo en el que se encapsulan las funciones, buscando contar con suficiente espacio en la memoria flash del enrutador, para la instalación de paquetes necesarios para el desarrollo del proyecto. Otra característica importante en la elección es la documentación existente y la facilidad de modificación del *"firmware"*.

2.4.2.1 Firmware OpenWRT

OpenWRT es una distribución de Linux para enrutadores que soportan licencia GPL. Esta distribución no está enfocada a tener muchas funcionalidades, por esto es necesario instalar paquetes para implementar las funciones requeridas. Con este firmware los usuarios desarrollan aplicaciones software bajo licencia GPL, de manera que existen muchos paquetes que brindan funcionalidades como: VLANs, VoIP, encriptación, etc. [50].

A lo largo de este estudio para la escogencia del *"firmware"* adecuado se han usado dos de las distribuciones de OpenWRT: Kamikaze y Whiterussian.

2.4.2.2 Firmware Kamikaze

Distribución de OpenWRT que tiene licencia GPL, cuenta con diversas características como configuración por medio de la interfaz web, control de potencia de transmisión y posibilidad de crear varias redes virtuales. La principal diferencia con el *"firmware"* WhiteRussian es el cambio de sistema de almacenamiento de variables, ya que no se guardan en la NVRAM (memoria de acceso aleatorio no volátil) sino en archivos de configuración en el directorio /etc/config lo cual dificulta la consulta de variables necesarias para el desarrollo del trabajo [51].

2.4.2.3 Firmware WhiteRussian

Cuenta con pocas características de configuración, entre ellas la configuración de conexiones WDS[14], seguridad WEP, WPA WPA2[15], además de las características fundamentales del enrutador como servidores DHCP, SSH cliente y servidor, telnet cliente y servidor. Es el *"firmware"* de menor tamaño entre los que se estudiaron, 1.5M dejando 2.5M de espacio libre en la memoria para la implementación, lo que es una ventaja, considerando el limitado tamaño de la memoria flash del enrutador, este cuenta con la mejor documentación, además ya se han desarrollado proyectos relacionados, por lo que se tiene mayor posibilidad de implementar este trabajo sin dificultades relacionadas con el *"firmware"* [49].

[14] WDS es un sistema que permite la interconexión inalámbrica de puntos de acceso en una red IEEE 802.11.
[15] Son sistemas de cifrado incluidos en el estándar IEEE 802.11 como protocolo para redes Wireless que permite cifrar la información que se transmite.

3. DISEÑO DEL SISTEMA HTB ADAPTABLE

En este capítulo se presenta el diseño del monitor de ancho de banda, la modificación del algoritmo HTB para que asigne dinámicamente el ancho de banda a cada una de las clases de tráfico y por último se describe de forma general el funcionamiento del sistema HTB adaptable a variaciones del ancho de banda.

3.1 DISEÑO DEL MONITOR DE ANCHO DE BANDA.

En el diseño del monitor de ancho de banda se identifican tres pasos para obtener una estimación del throughput del canal, y que esta sea consultada por el algoritmo HTB modificado.

1. Seleccionar el parámetro físico del canal inalámbrico que permita realizar una estimación del throughput.

2. Encontrar la relación de dicho parámetro con el throughput del canal.

3. Almacenar la estimación del throughput del canal en una variable del sistema para que esta sea consultada por el algoritmo HTB modificado.

El estándar 802.11 WLAN provee un número de posibilidades de modulación en la capa física, existe una relación directa entre el *"throughput"* y la relación señal a ruido (SNR), ya que en los sistemas inalámbricos se cambia de esquema de modulación con el fin de mantener un nivel de BER estable, dependiendo de la calidad del canal representada por la SNR.

La SNR y la tasa de error de bit (BER) están relacionadas, en la figura 2.6 se observa que un decremento de la SNR provoca un incremento en la BER, en consecuencia la degradación de calidad de canal implica un cambio en el esquema de modulación y una reducción del *"throughput"* alcanzable del canal.

El objetivo del sistema de monitoreo del canal inalámbrico es brindar una estimación del *"throughput"* alcanzable del canal, a partir de medidas de SNR, basándose en parámetros obtenidos del módulo de control de ganancia automático (ACG), presente en la mayoría de los chipsets inalámbricos. La estimación que el monitor realice debe actualizarse periódicamente.

El módulo ACG monitorea la condición de la señal y adapta el circuito RF de la tarjeta. Esta información se usa para calcular el nivel de señal, el nivel de ruido y la calidad de la

señal, valores entregados por el driver de la tarjeta inalámbrica en dB, permitiendo calcular la relación SNR del canal.

Para obtener el valor del throughput a partir de la SNR es necesario relacionar estos dos parámetros mediante una tabla, la cual debe ser construida previamente. Posteriormente se escribirá el valor de throughput en una variable del sistema, para que esta sea consultada por el algoritmo HTB modificado.

En la figura 3.1 se resumen los pasos a seguir para obtener el throughput del canal.

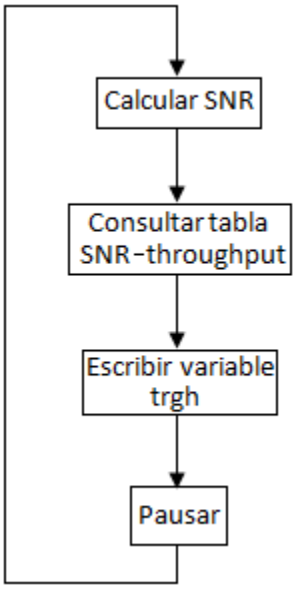


Figura 3.1 Diagrama de flujo del monitor de ancho de banda.

3.2 MODIFICACIÓN DEL CÓDIGO FUENTE HTB

En el diseño de las modificaciones a realizar al algoritmo HTB se identifican dos requerimientos esenciales.

1. capturar periódicamente el valor de la estimación de throughput realizada por el monitor de ancho de banda.

2. modificar constantemente la tasa de transmisión de todas las clases dependiendo del valor del throughput del canal.

En la etapa de análisis del código fuente del algoritmo HTB se identifican secciones que pueden ser útiles para el desarrollo del proyecto, concretamente los métodos:

- **htb_dequeue:** se encarga de desencolar paquetes desde una clase de tráfico.

- **htb_charge_class:** actualiza parámetros de la clase de tráfico y de su clase padre de esta, cada vez que se desencola un paquete.

También se identificaron parámetros del buket como la estructura *"rate"* y *"ceil"* de tipo qdisc_rate_table que tendrán que variar dependiendo del thruoghput del canal.

Para que el algoritmo capture periódicamente el valor de estimación de throughput del monitor de ancho de banda, se cuentan una cantidad definida de paquetes que salen de la cola de la clase, con este propósito se define una variable global y se agregan algunas líneas de código al método **htb_dequeue** (figura 3.2).

```
+static struct sk_buff *
+htb_dequeue_tree(struct htb_sched *q,int prio,int level)
+{
+     struct sk_buff *skb = NULL;
+     //struct htb_sched *q = (struct htb_sched *)sch->data;
+     struct htb_class *cl,*start;
+     /* look initial class up in the row */
+     DEVIK_MSTART(6);
+     start = cl = htb_lookup_leaf (q->row[level]+prio,prio,q->ptr[level]+prio);
+
```

Figura 3.2 Código htb_dequeue

El número de paquetes se asigna buscando que las clases actualicen su tasa de transmisión dentro de un periodo de tiempo, en este caso se busca un periodo de 6 segundos.

Con una velocidad de transmisión de 1Mbps, con la que se alcanza un throughput de aproximadamente 600kbps, se calcula el número de paquetes necesarios para el periodo de tiempo deseado de la siguiente forma:

Considerando un MTU de 1500 Bytes.

$$1500\frac{Bytes}{paquete} \times 8\frac{bits}{paquete} = 12000\frac{bits}{paquete}$$

$$6\,seg \times 600000\frac{bits}{seg} = 3600000\,bits$$

$$\frac{3600000\,bits}{12000\frac{bits}{paquete}} = 300\,paquetes$$

Por lo tanto, es necesario contar 300 paquetes para que las clases actualicen su tasa de transmisión cada 6 segundos, cuando el enlace tiene una velocidad de transmisión de 1Mbps, este tiempo se reduce a la mitad cuando la velocidad de transmisión del canal se incrementa a 2Mbps.

En el método **htb_charge_class** (figura 3.3) se agregan líneas de código donde indica la flecha, para que varíe la tasa de transmisión de la clase que desencola un paquete y de la clase padre, dependiendo del throughput del canal.

```
+#ifdef HTB_RATECM
+           /* update rate counters */
+           cl->sum_bytes += bytes; cl->sum_packets++;
+#endif
+
+           /* update byte stats except for leaves which are already updated */
+           if (cl->level) {
+                 cl->stats.bytes += bytes;
+                 cl->stats.packets++;
+           }
+
+           cl = cl->parent;
+     }
+}
```

Figura 3.3 Código htb_charge_class

Las líneas agregadas al código fuente modifican los valores de las estructuras *"rate"* y *"ceil"*, la figura 3.4 ilustra el funcionamiento del algoritmo HTB con las modificaciones planteadas.

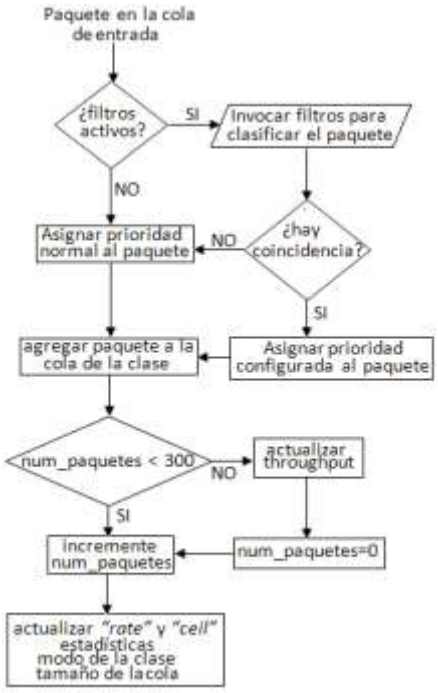


Figura 3.4 Diagrama de flujo del algoritmo HTB modificado

3.3 DESCRIPCIÓN GENERAL DEL SISTEMA

El sistema diseñado cuenta con dos módulos, el módulo planificador de tráfico que corresponde al algoritmo HTB modificado y el módulo de monitoreo de ancho de banda que incluye el bloque de estimación de *throughput,* en la figura 3.5 se presenta el sistema completo.

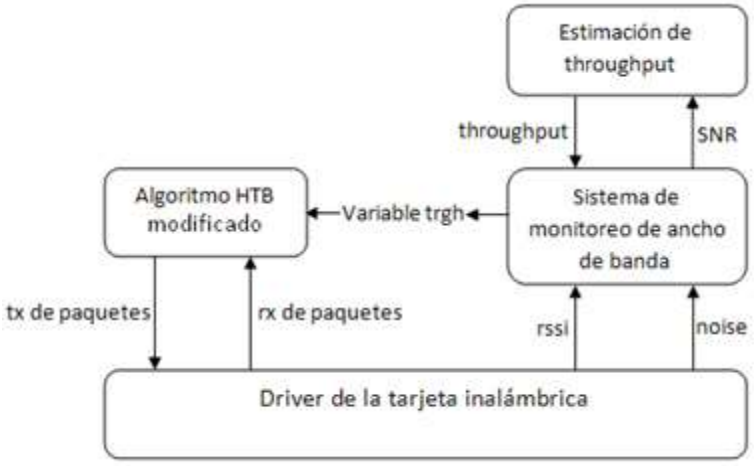


Figura 3.5 Descripción del sistema de monitoreo de ancho de banda

Los módulos funcionan de forma separada, y se relacionan por medio de una variable que almacena un valor estimado del *throughput* del canal, para calcular este valor el sistema de monitoreo de ancho de banda calcula la SNR a partir de datos obtenidos del driver de la tarjeta inalámbrica y consulta periódicamente un archivo de texto que contiene valores de SNR contra *"throughput*".

El sistema de monitoreo de ancho de banda guarda el valor calculado de *"throughput"* en la variable *"trgh"*, la cual es consultada periódicamente por el algoritmo HTB. El sistema de monitoreo y el algoritmo HTB se describen detalladamente en el numeral 3.1 y 3.2 respectivamente.

En el diagrama de secuencia del sistema HTB adaptable (figura 3.6), muestra cómo interactúan los diferentes subsistemas. El driver de la tarjeta inalámbrica proporciona al sistema de monitoreo de ancho de banda, valores de ruido y potencia de la señal recibida. Este calcula la SNR para estimar el throughput que será almacenado en una variable del sistema (variable "trgh"). El algoritmo HTB modificado consulta esta variable para calcular la tasa de transmisión de las clases de tráfico.

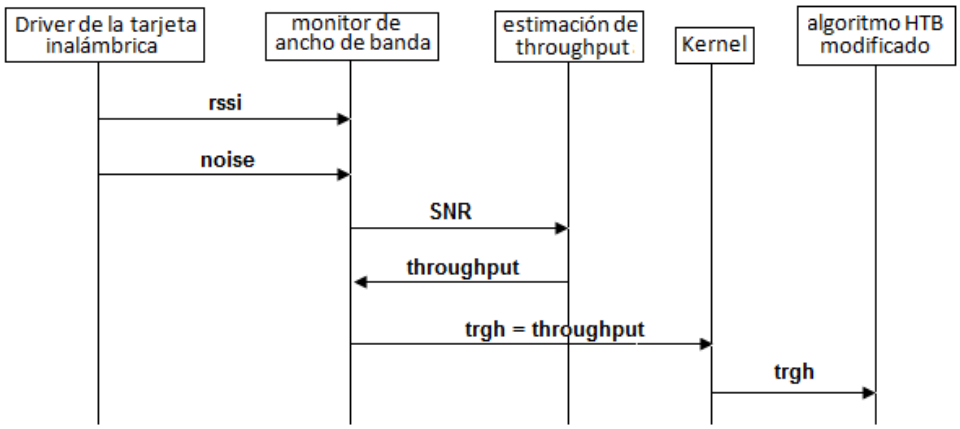


Figura 3.6 Diagrama de secuencia del sistema HTB adaptable

4. DESARROLLO DEL SISTEMA HTB ADAPTABLE

En este capítulo se presenta el desarrollo del sistema de monitoreo de ancho de banda y del algoritmo HTB modificado, además los pasos necesarios para la compilación del nuevo sistema HTB adaptable.

4.1 DESARROLLO DEL SISTEMA DE MONITOREO DE ANCHO DE BANDA

El sistema de monitoreo diseñado, primero obtiene los valores de nivel de señal y nivel de ruido, mediante los comandos *wl rssi* (Received Signal Strength Indicator, indicador de potencia de señal recibida) y *wl noise* respectivamente, posteriormente utiliza estos valores para calcular la SNR del canal.

Con el valor de SNR, se calcula (por medio de la tabla 4.1) el *throughput* esperado, el monitor de ancho de banda guarda este valor en una variable del sistema que será consultada por el algoritmo HTB modificado, el código implementado para este propósito se muestra a continuación.

```
#!bin/bash
while true;
do
signal=$(wl rssi)
noise=$(wl noise)
levelsignal=${signal#r*-}
levelnoise=${noise#n*-}
snr=`expr $levelnoise - $levelsignal`
troughput=`awk  -F " " '/'$snr'/ { print  $2 }'/etc/tabla`
nvram set trgh=$troughput
sleep 10
done;
```

En el script del enrutador en modo AP se debe especificar la dirección física del cliente, en consecuencia la línea número 4 del código en el enrutador modo AP es:

signal=$(wl rssi xx:xx:xx:xx:xx:xx)

El requisito de este código es tener instalado el paquete *"wl"*, este puede instalarse mediante los comandos:

ipkg update

ipkg install wl

Este script se guarda en el directorio etc/init.d/ bajo el nombre de S40monitor y se le dan permisos de ejecución mediante el comando:

chmod +x /etc/init.d/S40monitor

De esta forma el script se ejecutará cuando se encienda el enrutador inalámbrico

La desventaja del sistema de monitoreo diseñado, es la necesidad de construir de forma manual la tabla para estimar el *"throughput"*, ya que es necesario tomar medidas de la SNR del canal y medir el *"throughput"* alcanzado en ese instante, este procedimiento se realizó con la herramienta *IPERF*.

Posteriormente se registran los datos obtenidos en las mediciones, en una tabla que se guarda como un archivo de texto en el directorio */etc* bajo el nombre "tabla". De esta forma cuantos más datos se tomen relacionando la SNR con el *"throughput"* mejor será la estimación de este parámetro. Los datos obtenidos en las mediciones se presentan en la tabla 4.1.

SNR (dB)	BW (Mbps)	rate (Mbps)	Estándar
27	20.9	54	802.11g
26	19.7	54	
25	17.2	54	
24	15.8	54	
23	15.3	54	
22	14.9	54	
21	14.4	54	
20	13.7	36	
19	12.4	36	
18	11.2	24	
17	10.2	24	
16	7.83	18	
15	6.92	18	
14	6.31	18	
13	4.73	12	
12	4.12	12	
11	3.81	9	
10	3.61	9	
9	2.87	9	
8	2.48	5.5	
7	2.15	5.5	
6	1.93	4	

5	1.78	4	802.11b
4	1.34	2	
3	1.23	2	
2	0.694	1	
1	0.640	1	

Tabla 4.1 *Throughput* vs SNR

En la tabla 4.1 se observa un valor de *"throughput"* para cada valor de SNR, aunque estos valores pueden no ser exactos si son muy aproximados al *"throughput"* del canal alcanzado para una SNR dada.

Se aprecia que para las velocidades que corresponden al estándar 802.11b la velocidad real alcanzada es aproximadamente el 50% de la velocidad nominal del canal, y para las velocidades del estándar 802.11g es aproximadamente el 30% de la velocidad nominal del canal. Estos valores de *"throughput"* pueden variar dependiendo de los equipos utilizados y del número de clientes que estén asociados a la estación.

Teniendo en cuenta esta desventaja se optó por implementar el enlace inalámbrico con equipos idénticos en ambos extremos, ya que aunque todos los fabricantes construyen sus productos conforme al estándar IEEE 802.11, algunas aplicaciones pueden diferir de varias maneras en el *"hardware"* y *"software"*. Como consecuencia, la sensibilidad del receptor puede cambiar, así equipos diferentes pueden ofrecer una velocidad de transmisión diferente para una SNR dada, lo cual dificulta la construcción de la tabla SNR-*"throughput"* siendo necesaria una tabla para cada equipo diferente.

4.2 MODIFICACIÓN DEL ALGORITMO HTB

Para realizar las modificaciones mencionadas en el capítulo de diseño, es necesario declarar dos nuevas variables, la variable *"throughput"* donde se guardará el valor del throughput del canal inalámbrico, obtenido a través del monitor de ancho de banda, y la variable "num_pck" donde se contarán el numero de paquetes desencolados por las clases de tráfico (figura 4.1).

```
+int troughput = System("nvram get trgh");
+int num_pck = 0;
```

Figura 4.1 Variables globales

Según lo establecido en el capítulo 3, es necesario contar 300 paquetes para que las clases actualicen su tasa de transmisión cada 6 segundos, cuando el enlace tiene una velocidad de transmisión de 1Mbps, este tiempo se reduce a la mitad cuando la velocidad de transmisión del canal se incrementa a 2Mbps, el código agregado en el método **htb_dequeue** se muestra en la figura 4.2:

```
+htb_dequeue_tree(struct htb_sched *q,int prio,int level)
+{
+int num_pck=0;
+     if (num_pkc < 300)
+     {
+     num_pck = num_pck +1 // incrementa el  numero de paquetes
enviados
+     }
+     else
+     {
+     troughput= exec("nvram get trgh"); //actualiza la variable global
//trougput
+     num_pck=0;
+     }
+
```

Figura 4.2 Modificación método htb_dequeue_tree

Con las líneas agregadas en el método **htb_dequeue**, se cuenta una cantidad definida de paquetes, una vez se llegue a esta cantidad (en este caso 300 paquetes) se actualiza la variable *throughput* con el comando `System("nvram get trgh"),` es aquí donde el algoritmo HTB consulta la variable "trgh", en la cual el sistema de monitoreo de ancho de banda ha guardado el valor del *throughput* del canal inalámbrico. Posteriormente la variable *"num_pck"* toma un valor igual a cero para que este proceso se repita cada vez que se desencolen 300 paquetes.

En la estructura de la clase (**htb_class**) se agregan dos estructuras auxiliares *"aux"* y *"auxc"* de tipo *"qdisc_rate_table"* que guardan el valor de *"rate"* y *"ceil"* asignados en el script de implementación de HTB (figura 4.3), con el fin de calcular la tasa de transmisión dependiendo del valor de estas estructuras.

```
+    struct qdisc_rate_table *auxc;  /* tabla de ceil auxiliar
+    long buffer,cbuffer;           /* token bucket depth/rate */
+    long mbuffer;                  /* max wait time */
+    long tokens,ctokens;           /* current number of tokens */
+    psched_time_t t_c;             /* checkpoint time */
+
+    aux = rate;                    /*TABLA DE TASA AUXILIAR*/ <----------
+    auxc = ceil;                   /*TABLA DE CEIL AUXILIAR*/ <----------
+};
```

Figura 4.3 Estructuras auxiliares.

En el método **htb_charge_class** se agregan líneas de código que modifican la tasa de transmisión y parámetro ceil (figura 4.4), cada vez que una clase envía un paquete, y posteriormente el algoritmo define cuantos *"tokens"* asigna a cada clase de tráfico.

```
cl->rate.rate.rate = (cl->aux.rate.rate) * (troughput / 100);
cl->ceil.rate.rate = (cl->auxc.rate.rate) * (troughput / 100);
```

Figura 4.4 Modificación de rate y ceil de la clase

En estas líneas se multiplica la variable *"aux"* por la variable *"throughput",* y se divide entre 100. El resultado de esta operación se almacena en una variable de la estructura "*rate".*

De esta manera se le asigna como velocidad de transmisión de la clase el porcentaje calculado del *throughput* del canal.

Con el mismo procedimiento se modifica el parámetro *"ceil"* de la clase, con la diferencia de que para el cálculo del parámetro ceil se utiliza la estructura auxiliar *"auxc"*.

Con esta modificación se cambia la forma en la cual se establece la tasa de transmisión a una clase en el código de implementación HTB, ya que no es necesario fijar la velocidad de la clase, sino el porcentaje del *"throughput"* del canal que será asignado, como se muestra a continuación.

```
$TC class add dev $DEV parent 1:1 classid 1:10 htb rate 50kbps ceil
100kbps burst 20k cburst 20k
```

En la línea anterior se le asigna a una clase con identificador 1:10 un 50% del *"throughput"*, aunque es necesario escribir las unidades en "bps" ya que es requisito de la aplicación de control de tráfico.
El valor asignado a los parámetros rate y ceil en la línea anterior queda copiado en las estructuras auxiliares *"aux"* y *"auxc",* con las que se calcula la tasa asegurada y la tasa máxima de la clase utilizando el procedimiento anteriormente descrito.

4.3 COMPILACIÓN DEL CÓDIGO HTB

Para incluir el código HTB modificado en el *"firmware"* OpenWrt, es necesario crear una nueva imagen del *"firmware"*, esto se realiza mediante el proceso de compilación cruzada, debido a que no es posible simplemente compilar el código en un *"kernel"*[16] de Linux de un computador personal, porqué es una plataforma hardware completamente diferente a la plataforma hardware en la que el código se ejecutará [52].

Para el desarrollo del proyecto se utilizó el *"firmware"* WhiteRussian de OpenWrt, el primer paso para la creación de la imagen del *"firmware"* es descargar su fuente desde el enlace http://downloads.openwrt.org/whiterussian/0.9/whiterussian-0.9.tar.bz2 y descomprimirlo [53].

En la fuente del firmware hay tres directorios claves:

- *"toolchain"*
- *"target"*
- *"package"*

El directorio *"toolchain"* hace referencia al compilador, la librería c y herramientas necesarias para la compilación de la imagen del *"firmware"*. Los resultados de la compilación son dos nuevos directorios, un directorio temporal llamado toolchain_build_mipsel, usado para construir el *"toolchain"* de una arquitectura específica, y staging_dir_mipsel donde el *"toolchain"* resultante es instalado.

[16] Kernel es un software que actúa como sistema operativo. Facilita a los distintos programas acceso seguro al hardware.

El directorio *"target"* hace referencia a la plataforma embebida, contiene items específicos para una plataforma. En el subdirectorio "target/linux" se almacena la configuración del *"kernel"* y los parches para una plataforma en particular, el subdirectorio "target/linux/image" contiene instrucciones para empaquetar el *"firmware"* para una plataforma ya sea *jffs2* o *Squashfs* los cuales se describen a continuación:

- *jffs2*: Contiene un sistema de archivos raíz re-escribible, que amplía la imagen del *"firmware"* al tamaño de la memoria flash disponible en el *"hardware"*, por lo que debe seleccionar la imagen correcta para el tamaño de memoria flash del equipo. La ventaja de utilizar jffs2 es que se pueden realizar modificaciones dentro de toda la jerarquía de archivos, aunque se corre el riesgo de dejar inservible el enrutador si se configura de manera inadecuada.

- *Squashfs*: es un sistema de solo lectura para unix, la versión estándar utiliza compresión gzip, aunque existe un sistema de archivos *squashfs* modificado el cual utiliza un algoritmo de compresión *LZMA* ("*Lempel-Ziv-Markov chain-Algorithm"*). Al iniciar, se puede crear un segundo sistema de archivos re-escribible, que contendrá las modificaciones al sistema de archivos raíz, incluyendo los paquetes que se instalan.

En el directorio *"Package"* se encuentran las configuraciones de los paquetes que serán incluidos en la imagen del firmware, incluyendo los parches para el paquete iproute2, que se encuentran en el directorio \whiterussian-0.9\package\iproute2\patches. En este directorio se copia el parche con el código HTB modificado con el nombre 004-htb3.6_2.4.17.patch, y de esta manera el código se instalará durante la creación de la imagen del firmware.

A continuación se detalla el procedimiento de configuración de la imagen del *"firmware"* WhiteRussian con el código HTB modificado.

Para empezar la creación de la imagen del firmware, se ingresa como súper usuario al directorio del \whiterussian-0.9 y se teclea el comando *"make menuconfig"* el cual desplegará el menú de configuración de OpenWrt (figura 4.1).

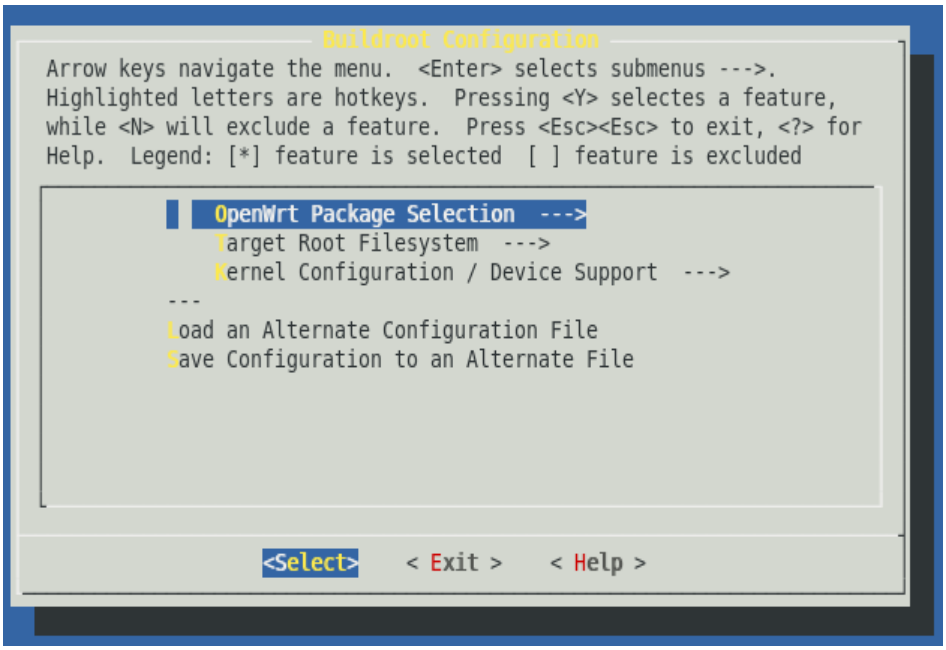


Figura 4.5 Menú principal de configuración OpenWrt

La primera opción de este menú permite escoger los paquetes que se instalarán dentro de la imagen del firmware (figura 4.2). Aquí se selecciona el paquete iproute2 y se deja el resto de paquetes como están, de esta forma se descarga el paquete iproute2 y se instalan sus parches incluido 004-htb3.6_2.4.17.patch.

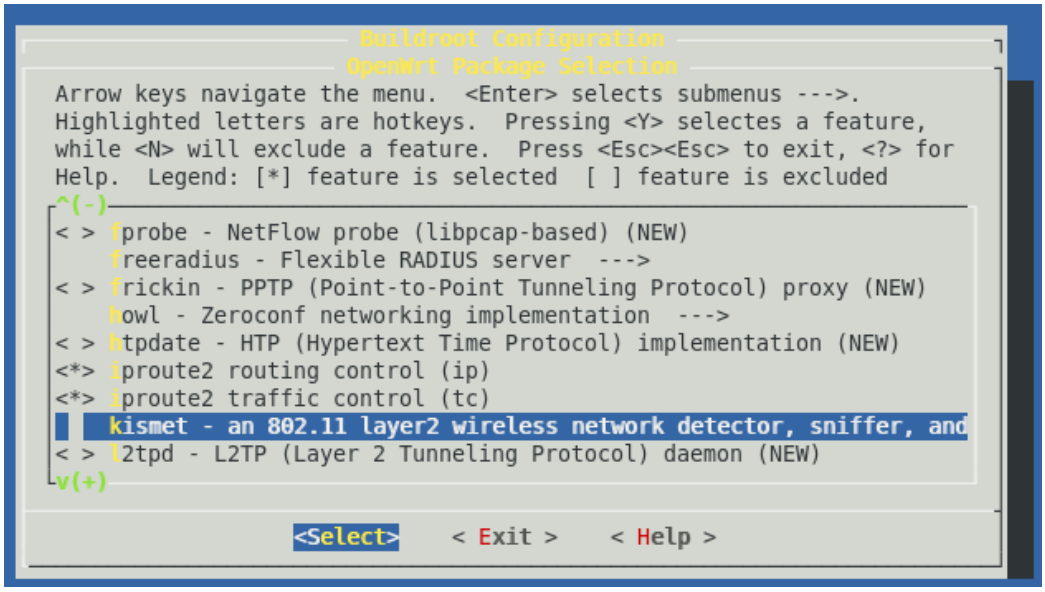


Figura 4.6 Menú de selección de paquetes

La segunda opción del menú principal, "*target root filesystem*" permite escoger el sistema de archivos en el que se generará la imagen del firmware (figura 4.3), se puede escoger

entre *jffs2* y *el squashfs* cuyas características se mencionaron anteriormente, en este caso se seleccionan las dos opciones.

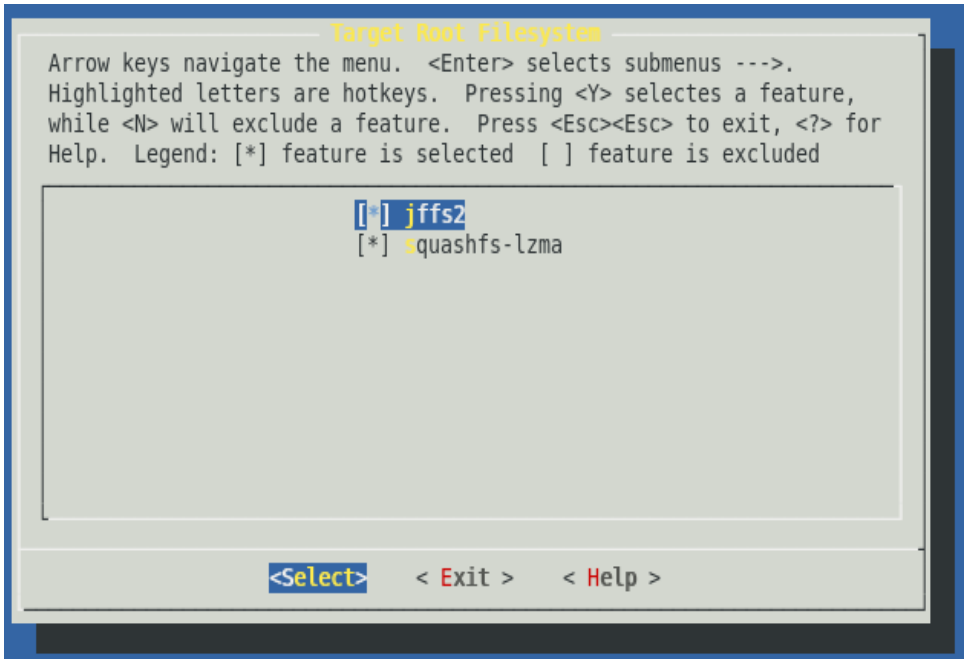


Figura 4.7 Menú de selección del sistema de archivos

La tercera opción del menú de configuración es la configuración del kernel "*Kernel Configuration / Device Support*" (figura 4.4), se puede seleccionar el tipo de procesador y algunos módulos adicionales.

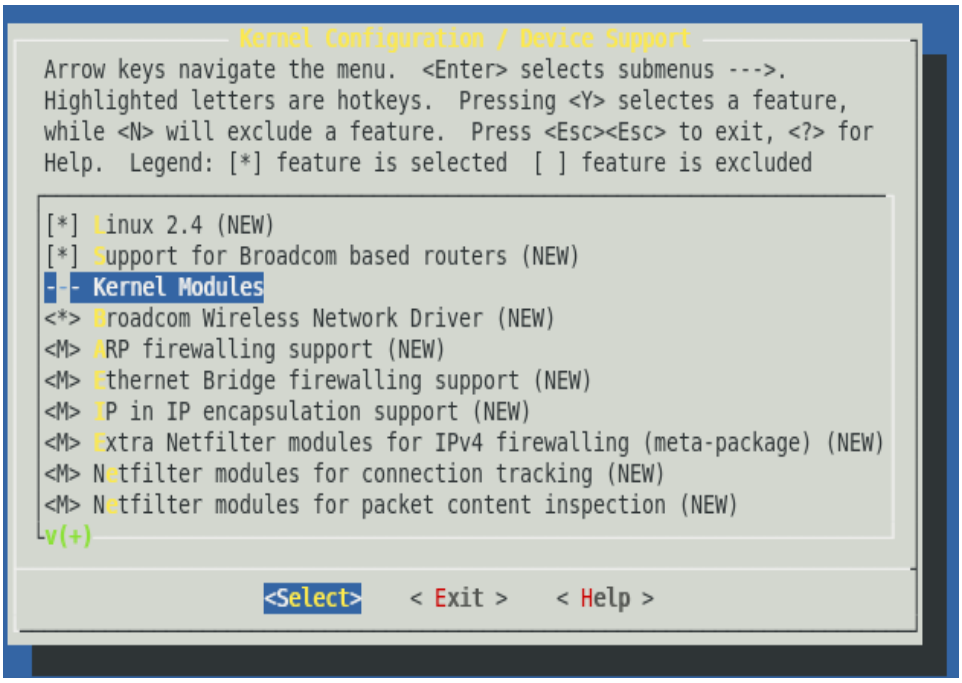


Figura 4.8 Menú de configuración del kernel

En la última opción del menú se nombra el archivo en el que se guardará la configuración realizada (figura 4.5).

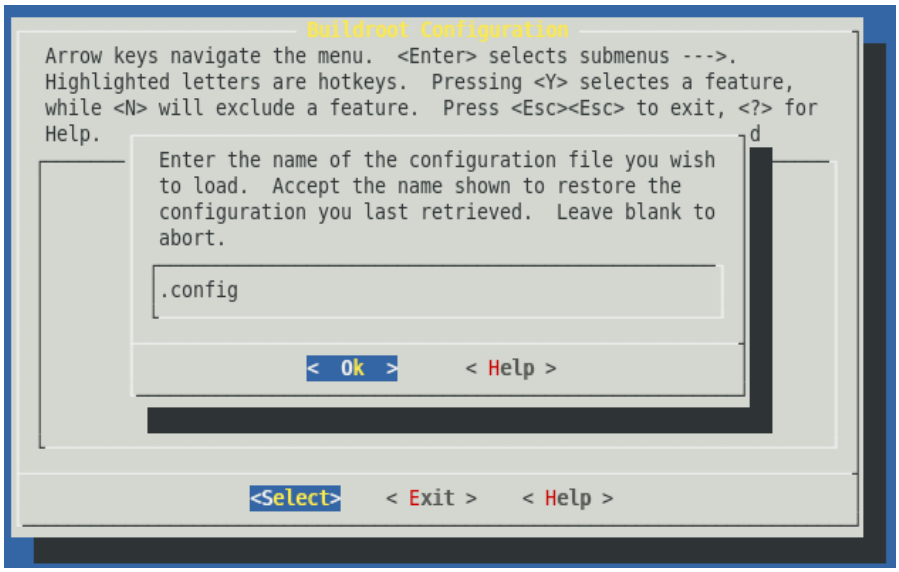


Figura 4.9 Nombrar el archivo de configuración

Al finalizar se guarda la configuración del *buildroot* en el archivo .config, esto se realiza cuando se sale del menú de configuración (figura 4.6).

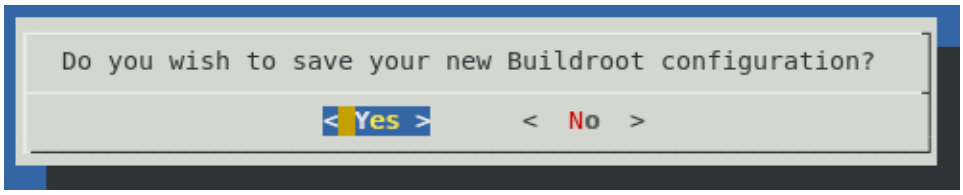


Figura 4.10 Guardar la configuración

Para compilar el *"firmware"*, se teclea "make", con este comando se inicia la compilación y construcción del *"Kernel"*, se descargan las fuentes necesarias desde Internet, se realiza la compilación cruzada mediante el "*toolchain"*, y la construcción del *"firmware"* con las aplicaciones seleccionadas.

Durante el proceso de construcción, el *"buildroot"* descargará todas las fuentes al directorio *"dl"* y empezará parchando y compilando estos en el directorio *"build_<arch>"*. Cuando finalice, el firmware resultante estará en el directorio *"bin"* y los paquetes estarán en el directorio *"bin/packages"* con una extensión .ipkg.

Una vez se tenga la imagen del *"firmware"* este se podrá instalar en los enrutadores, mediante web o mediante comandos de línea en la consola telnet.

En este caso realiza por línea de comando, para esto primero es necesario cargar la imagen del *"firmware"* en el enrutador con el siguiente comando
wget http://192.168.0.6/openwrt-wrt54g-squashfs.bin

5. PRUEBAS DEL SISTEMA HTB ADAPTABLE

En este capítulo se describen los requerimientos necesarios para implementar el sistema HTB adaptable, se muestra la configuración software y hardware empleados en el enlace inalámbrico y el tipo de pruebas que se realizan con el fin de evaluar el funcionamiento del algoritmo HTB adaptable a variaciones del ancho de banda.

5.1 CONFIGURACIÓN DEL ESCENARIO DE PRUEBAS

5.1.1 Requerimientos de implementación del sistema

Es necesario instalar paquetes adicionales para que los comandos del script de implementación del sistema de planificación de tráfico puedan ser interpretados correctamente.
Para la instalación de los paquetes necesarios se utilizan los siguientes comandos desde la consola del enrutador.

```
>Ipkg update    //actualiza la lista de paquetes disponibles en
                //para el firmware
>Ipkg install tc //contiene la utilidad de control de tráfico
```

5.1.2 Montaje del sistema

Para el montaje del sistema se configura un enlace punto a punto inalámbrico 802.11g como se presenta en la figura 5.1. Donde se conecta un cliente de voz y de datos HTTP desde el PC 1 al servidor de voz *"asterisk"* y el servidor de datos HTTP en el PC 2. El PC 1 está conectado por su interfaz Ethernet a un enrutador linksys WRT54GL en modo cliente que asocia a otro enrutador linksys WRT54GL en modo AP, que a su vez está conectado por uno de sus puertos LAN a la interfaz Ethernet del PC 2.

Figura 5.1 Topología del montaje del sistema

El sistema operativo es GNU/Linux en la distribución Ubuntu 8.10, en la tabla 5.1 se resumen las características *"hardware"* y *"software"* de los equipos utilizados.

Equipo	CPU	Memoria	Red	Sistema operativo	Software	Dirección IP
Cliente	Intel core 2 Duo 1.4 Ghz	2 GB	Broadcom NetLink fast Ethernet	GNU/Linux unbuntu 8.10	Asterisk PBX	192.168.1.6
Servidor	AMD Sempron 1.80 Ghz	512MB	VIA Rhine II Fast Ethernet Adapter	GNU/Linux unbuntu 8.10	Asterisk PBX Apache	192.168.0.6
Enrutador AP	BCM3302 – 216 Mhz	16MB	BCM3302	Linux OpenWRT	Wireless utils	192.168.0.2
Enrutador Cliente	BCM3302 – 216 Mhz	16MB	BCM3302	Linux OpenWRT	Wireless utils	192.168.1.3

Tabla 5.1 Características *"hardware y software"* del montaje

5.1.2.1 Configuración del enlace inalámbrico

La configuración de los enrutadores se realiza mediante consola Telnet, dada la facilidad para realizar las modificaciones necesarias a las variables del enrutador y el acceso a todos los archivos del *"firmware"*.

A continuación se presenta un ejemplo de la forma en la cual se modifica el valor de una variable por línea de comando.

Comando para consultar el valor de la variable:
nvram get nombre_variable

Comando para modificar el valor de una variable:
nvram set nombre_variable=valor_variable

Comando para guardar los cambios en las variables:
nvram commit

En la figura 5.2 se detalla la topología del enlace inalámbrico utilizado para verificar el funcionamiento del algoritmo HTB modificado.

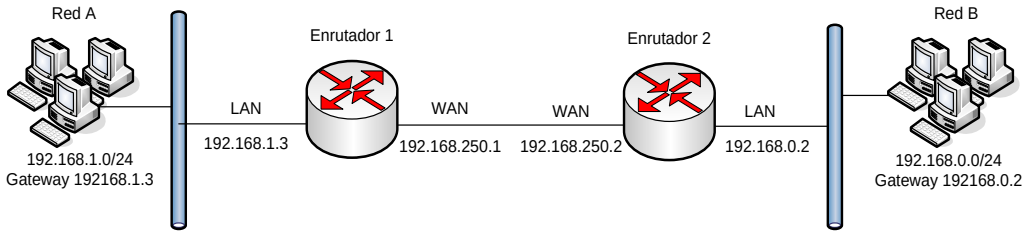


Figura 5.2 Topología del enlace punto a punto

La configuración del enrutador en modo AP según la figura 5.2 es:

```
lan_ifname=br0                #Nombre de la Interfaz LAN
lan_ifnames=vlan0 eth2  #Interfaces asociadas al Puerto LAN
lan_ipaddr=192.168.0.2  #Dirección IP del Puerto LAN
lan_netmask=255.255.255.0     #Mascara de Red LAN
lan_proto=static              #Configuración LAN Estática
wan_gateway=192.168.250.1     #Puerta de Enlace WAN
wan_hostname=estacion         #Nombre del Equipo
wan_ifname=eth1               #Interfaz Asociada al puerto WAN
wl0_ifname=eth1               #Nombre de la Interfaz Inalámbrica
wan_ipaddr=192.168.250.2      #Dirección IP del Puerto WAN
wan_proto=static              #Configuración WAN Estática
wan_netmask=255.255.255.252   #Mascara de red WAN
wl0_mode=ap                   #Modo de operación Master
wl0_radio=1                   #Tráfico inalámbrico Activado
wl0_ssid=redhtb               #SSID del Enlace
```

La configuración para el enrutador en modo cliente según la figura 5.2 es:

```
lan_ifname=br0
lan_ifnames=vlan0 eth2
lan_ipaddr=192.168.1.3
lan_netmask=255.255.255.0
lan_proto=static
wan_hostname=cliente
wan_ifname=eth1
wan_ipaddr=192.168.250.1
wan_netmask=255.255.255.252
wan_proto=static
wan_gateway=192.168.250.2
wl0_ifname=eth1
wl0_mode=sta                    #Modo de operación Cliente
wl0_radio=1
wl0_ssid=redhtb
```

Es necesario desactivar las políticas que el *"firewall"* tiene por defecto, quitando los permisos de ejecución al archivo S35firewall ubicado en el directorio /etc/init.d mediante el comando:

chmod –x /etc/init.d/S35firewall

5.1.2.2 Características físicas del enlace punto a punto

El enlace punto a punto se implementa en un medio urbano, con línea de vista entre los enrutadores y con algunos árboles entre ellos (figura 5.3), a continuación se describen los parámetros del enlace:

Distancia: 30mts.
Potencia de transmisión: 18dBm
Frecuencia de transmisión: 2412MHz (canal 1)
Nivel de sensibilidad: -90 dBm

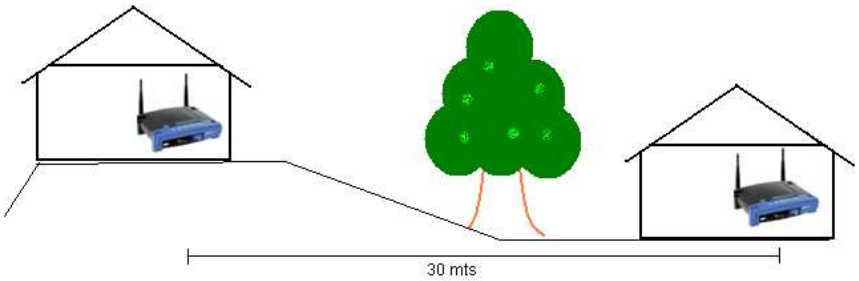


Figura 5.3 Enlace punto a punto 802.11

5.1.3 Implementación de HTB en OpenWRT

En el modelo de HTB implementado se analizan dos tipos de tráfico: voz y datos HTTP, para medir parámetros de calidad como el *"Jitter",* retardo para VoIP y el *"throughput"* para datos HTTP.

Para distribuir el ancho de banda disponible en el enlace es necesario crear dos clases HTB, una para cada tipo de tráfico, adicionalmente es necesario crear una tercera clase para asignar una porción de ancho de banda a otros tipos de tráfico, la figura 5.4 ilustra el modelo de clases HTB.

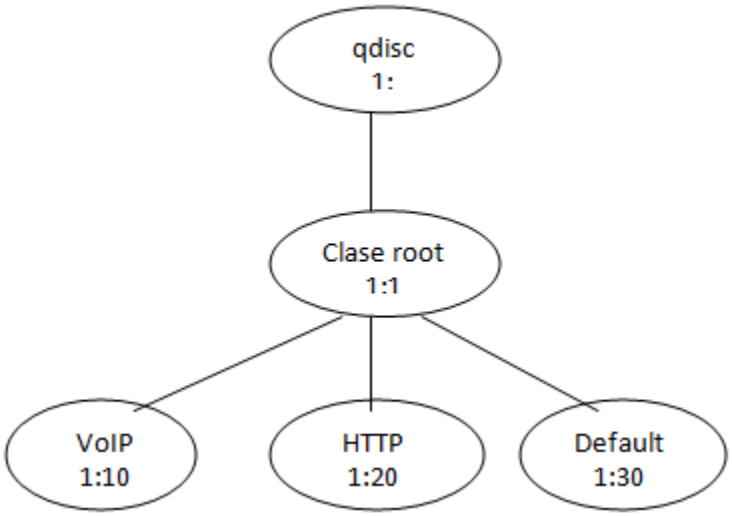


Figura 5.4 Modelo de clases HTB

Para crear este esquema, es necesario crear la clase qdisc, la cual representa la interfaz de salida, esto se realiza mediante el comando

```
$TC qdisc add dev $DEV root handle 1: htb default 30
```

La clase por defecto es la clase 30, es decir que todo el tráfico que no se envié por las clases 10 o 20 será enviado por esta clase.

Posteriormente se crea una clase en la cual se asigna el ancho de banda disponible, esta clase será la clase padre de las clases entre las que se distribuye el ancho de banda para cada uno de los tipos de tráfico.

```
$TC class add dev $DEV parent 1: classid 1:1 htb rate ${RATE}kbit burst
20k cburst 20k
```

En esta clase se le asigna al parámetro *"rate"* el total del ancho de banda del canal, esta tiene identificador 1:1 y como clase padre tiene a la clase qdisc, una vez creada se crean las clases de los tipos de tráfico con los siguientes comandos.

```
$TC class add dev $DEV parent 1:1 classid 1:10 htb rate $(($RATE*7/100))
ceil $RATE burst 20k cburst 20k
$TC class add dev $DEV parent 1:1 classid 1:20 htb rate $(($RATE*2/100))
ceil $RATE burst 20k cburst 20k
$TC class add dev $DEV parent 1:1 classid 1:30 htb rate $(($RATE*1/100))
ceil $RATE burst 20k cburst 20k
```

El siguiente paso es crear los filtros U32, encargados de identificar los tipos de tráfico y asignar los paquetes a las clases creadas anteriormente, los filtros se crean con los siguientes comandos.

```
$TC filter add dev $DEV protocol ip parent 1: prio 1 u32 match ip sport
4569 0xffff flowid 1:10
$TC filter add dev $DEV protocol ip parent 1: prio 1 u32 match ip sport
80 0xffff flowid 1:20
```

En estos comandos se identifica el tráfico por su puerto de salida, el puerto 80 para tráfico HTTP y 4569 para tráfico de voz, debido a el tráfico HTTP utiliza por defecto el puerto 80 TCP y el protocolo IAX2 (**I**nter-**A**sterisk e**X**change v**2**) utiliza el puerto 4569 UDP para tráfico de voz.

El código completo para la implementación de HTB según la figura 4.4 se encuentra en el anexo 1.

El script se guarda en el directorio etc/init.d/ bajo el nombre S41qos y es necesario asignarle permisos de ejecución mediante el comando:

chmod +x /etc/init.d/S41qos

De esta forma el *"script"* se ejecuta cuando se encienda el enrutador:

Para la implementación de las clases con el algoritmo HTB modificado, únicamente se debe cambiar la forma en la que definen las clases de tráfico, de manera que el código para crear las clases será:

```
#clase raiz HTB

$TC qdisc add dev $DEV root handle 1: htb default 10
$TC class add dev $DEV parent 1: classid 1:1 htb rate $100kbit burst 20k
cburst 20k

#clases HTB

$TC class add dev $DEV parent 1:1 classid 1:10 htb rate $70kbit ceil
$100kbit burst 15k cburst 15k
$TC class add dev $DEV parent 1:1 classid 1:20 htb rate $20kbit ceil
$100kbit burst 15k cburst 15k
$TC class add dev $DEV parent 1:1 classid 1:30 htb rate $10kbit ceil
$100kbit burst 15k cburst 15k
```

El código completo para la implementación de HTB modificado según la figura 5.4 se encuentra en el anexo 1.

5.1.4 Instalación y configuración del servidor asterisk

La instalación del servidor de VoIP *"asterisk"* se debe realizar en los dos PC, porque uno de ellos actuará como servidor y el otro como cliente, gracias a esto se pueden realizar varias llamadas desde un mismo equipo cliente hacia el equipo servidor, por canales diferentes.
Para la instalación del servidor *"asterisk"* se realizan los siguientes pasos, ingresando como súper usuario.

```
1. apt-get install g++

2. apt-get install libncurses5-dev

3. apt-get install libstdc++5

4. apt-get install ssh

5. uname -a
 Linux servidor 2.6.22-14-generic #1 SMP Sun Oct 14 23:05:12 GMT 2009 i686
GNU/Linux

6. apt-cache search inux-headers 2.6.22-14-generic
linux-headers-2.6.22-14-generic - Linux kernel headers for version 2.6.22
on x86/x86_64

7. apt-get install linux-headers-2.6.22-14-generic

8. ln -s /usr/src/linux-headers-2.6.22-14-generic /usr/src/linux-2.6

9. Descargar de http://www.asterisk.org/downloads:
   asterisk.tar.gz-1.4.17.tar.gz
   libpri-1.4.3.tar.gz
   zaptel-1.4.8.tar.gz

10. cp *.tar.gz /usr/src/

11. tar -zxvf zaptel-1.4.8.tar.gz
    cd /zaptel
    make clean
    make install

12. tar -zxvf libpri-1.4.3.tar.gz
    cd /libpri
    make clean
    make install
13. tar -zxvf asterisk-1.4.17.tar.gz
    cd /asterisk
    ./configure

14. make clean
    make install
    cd ..

15. make samples
```

5.1.4.1 Configuración del servidor asterisk

5.1.4.1.1 Configuración de los clientes en el archivo "IAX.conf"

Se declara un cliente con la dirección IP del PC 1.

```
[cliente1]
type=friend       ;Puede hacer y recibir llamadas
host=192.168.1.3  ;Se conectará desde esa IP
context=ciclo           ;contexto al que pertenece
qualify=yes       ;Se monitorea la conexión y el retardo
disallow=all            ;No permite un códec diferente g711u
allow=ulaw        ;utilizara el códec g711u
```

5.1.4.1.2 Configuración del plan de marcado en el archivo "extensions.conf"

En este archivo se configura el flujo de la llamada dentro del contexto al cual pertenece el cliente que se declaró anteriormente.

```
[ciclo]                                    ;Contexto Ciclo
exten=s,1,BackGround,demo-instruct         ;Reproduce el mensaje
;demo-instruct
exten=s,2,Goto(ciclo,s,1)                  ;Crea un ciclo que se
                                           ;repite indefinidamente
```

5.1.4.2 Configuración del cliente asterisk

En los clientes se debe declarar la dirección IP del servidor al cual se conectará en el archivo "iax.conf".

```
;Se declara la sección general
;Se declara el servidor que llamado ap

[ap]              ;nombre del servidor
type=friend
host=192.168.0.6  ;dirección ip del servidor al que se conectara
context=ciclo
qualify=yes
disallow=all
allow=ulaw
trunk=no         ;Para que cada llamada se haga como una llamada
                ;independiente
```

Para realizar un llamada se crea el siguiente script, el cual se copia en la dirección */var/spool/asterisk/outgoing* en el PC cliente. Cuando se copia el script en este directorio con extensión **.call* se genera un llamada, es necesario copiar tantos script como llamadas se deseen realizar, cada una de estas llamadas se realiza como una llamada independiente gracias a la configuración del cliente.

```
; Archivo sample.call
Channel: IAX2/ap     ; llama al servidor de nombre 'ap'
MaxRetries: 2          ; numero de reintentos antes de descartar la
        ; llamada
RetryTime: 60          ; Reintento cada 60 segundos
WaitTime: 30           ; Esperara 30 segundos la respuesta
Context: tutorial      ; contexto que buscara la llamada,
Extension: s           ; extension s y
Priority: 1            ; prioridad 1
```

La instalación del servidor apache se realizó mediante el comando:

apt-get install apache2

La instalación de la herramienta de monitoreo escogida para realizar las pruebas se realizó mediante el comando:

apt-get install wireshark

5.2 PRUEBAS

Para las pruebas de la implementación del algoritmo HTB y el algoritmo HTB modificado, se utilizó el montaje de la figura 5.2. El PC cliente (PC 1) recibe tráfico de voz y de datos, y genera gráficas del comportamiento de los mismos. Como herramienta de monitoreo de tráfico se utilizó el analizador de protocolos de red *"Wireshark"* versión 1.2.1. En los puntos de acceso o enrutadores inalámbricos se corrió el "*script"* del numeral 4.5, en el PC servidor (PC 2) fueron instalados servicios HTTP y VoIP, los cuales se acceden desde el PC cliente.

Para las pruebas se utilizó en el "*script"* una asignación del ancho de banda del 70% del enlace para el tráfico de VoIP, 20% para el tráfico HTTP, 10% para el tráfico por defecto, esta asignación de tasas de tráfico de realiza buscando darle la mayor parte de los recursos al tráfico de voz. El parámetro *"ceil"* para todas las clases es igual al 100 % de la capacidad del enlace, así cualquier clase puede utilizar toda la capacidad del canal en ausencia de otros tipos de tráfico.

Para la verificación del funcionamiento del algoritmo HTB modificado se realizan dos pruebas, una en el que la velocidad del canal inalámbrico se incrementa y otra en la que la velocidad disminuye, utilizando el algoritmo HTB y posteriormente el algoritmo HTB modificado. En estos casos se configura el algoritmo HTB y HTB modificado con una velocidad inicial que corresponde a la velocidad del canal y se analiza tráfico de voz y de datos.

Debido a que las gráficas de comportamiento de tráfico se generan monitoreando la interfaz Ethernet del cliente (PC1), se tendrá un comportamiento similar al del modelo de gráficas de tráfico (figura 6.1), en el que el tráfico de voz de subida tiene un tasa mayor que el tráfico de voz de bajada, debido a que este no ha sido limitado por el algoritmo HTB que se encuentra en el enrutador inalámbrico, si se monitorea la interfaz Ethernet del servidor se presenta el caso contrario en el que el tráfico de voz de bajada tiene un nivel mayor que el tráfico de voz de subida.

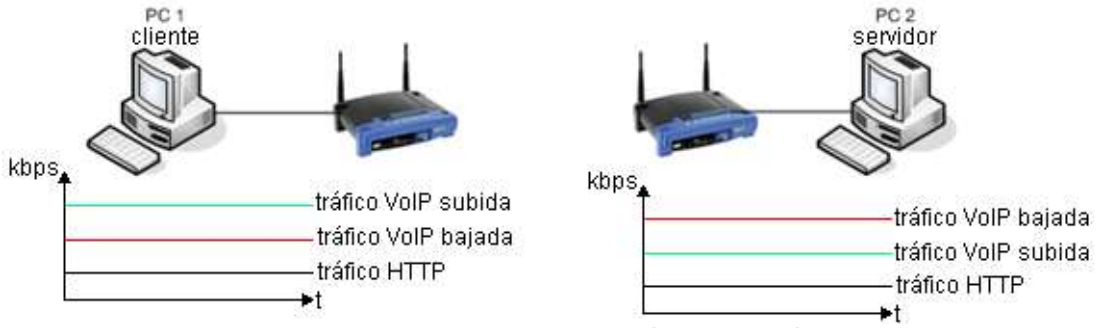


Figura 5.5 Modelo de gráficas de tráfico

5.2.1 Pruebas del algoritmo HTB modificado con incremento en la velocidad

En esta etapa se comparan las tasas de transferencia para tráfico de voz y de datos con el algoritmo HTB original y el algoritmo HTB modificado, con el fin de analizar cómo se comportan los tipos de tráfico ante un cambio en la velocidad del canal inalámbrico de 1Mbps a 2Mbps. Las variaciones en la velocidad de transmisión del canal se inducen cambiando la ubicación del enrutador cliente.

El procedimiento para realizar pruebas se describe a continuación.

1. Se inicia una descarga HTTP desde el cliente.

2. Se realizan algunas llamadas.

3. Aproximadamente en la mitad del transcurso de las llamadas se induce un cambio de velocidad del enlace.

4. Después de finalizado el primer grupo de llamadas se inicia una cantidad de llamadas mayor que la anterior.

5.2.2 Pruebas del algoritmo HTB modificado con un decremento en la velocidad

Para la segunda parte de las pruebas se inicia un único grupo de 7 llamadas con el algoritmo HTB y posteriormente con el algoritmo HTB modificado, y se realiza una variación en velocidad de transmisión de 2Mbps a 1Mbps inducida con el cambio de posición del enrutador en modo cliente.

6. ANÁLISIS DE RESULTADOS

En este capítulo se realiza un análisis de los resultados obtenidos con las pruebas descritas en el capítulo anterior. Teniendo en cuenta tasas de transmisión de los dos tipos de tráfico, retardo y jitter del tráfico de voz. Con el propósito de concluir que tan eficiente es el algoritmo HTB adaptable comparado con el algoritmo HTB original.

6.1 ANÁLISIS DE RESULTADOS DE LAS PRUEBAS DEL ALGORITMO HTB MODIFICADO CON INCREMENTO EN LA VELOCIDAD

A continuación se presenta un análisis de los resultados de la prueba descrita en el numeral 5.2.1 con un incremento en la velocidad de transmisión del canal, utilizando las gráficas de tasas de transmisión, retardo y jitter proporcionadas por la herramienta de monitoreo WireShark.

6.1.1 Análisis de throughput

Para la primera parte de las pruebas se sigue el procedimiento anterior con el algoritmo HTB y posteriormente con el algoritmo HTB modificado, con un cambio de velocidad de 1Mbps a 2Mbps, se inician las pruebas con 5 llamadas y posteriormente se inician 8 llamadas.

En las figuras 6.1 y 6.2 se presentan las gráficas de comportamiento del tráfico de voz y datos utilizando el algoritmo HTB y HTB modificado, el eje X representa el tiempo en segundos, el eje Y representa la tasa de transmisión en Kbps. La línea negra representa el tráfico HTTP, la verde el tráfico de voz desde el cliente hacia el servidor y la línea roja el tráfico de voz desde el servidor hacia el cliente.

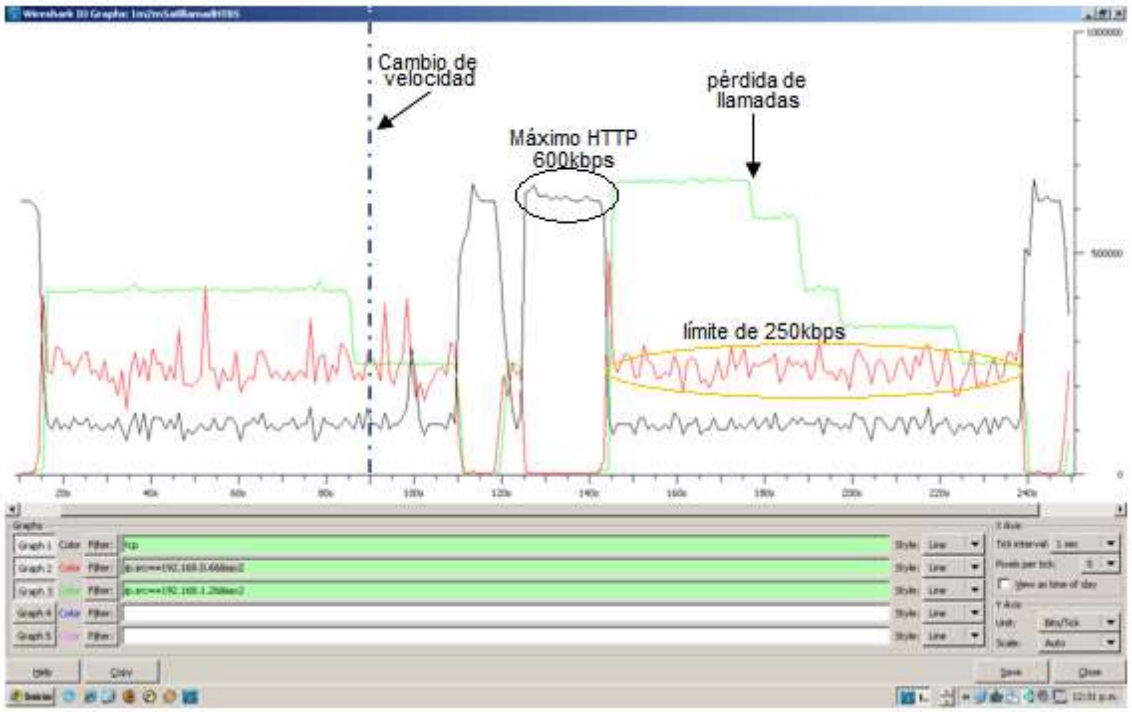


Figura 6.1 Tráfico de voz y HTTP con el algoritmo HTB

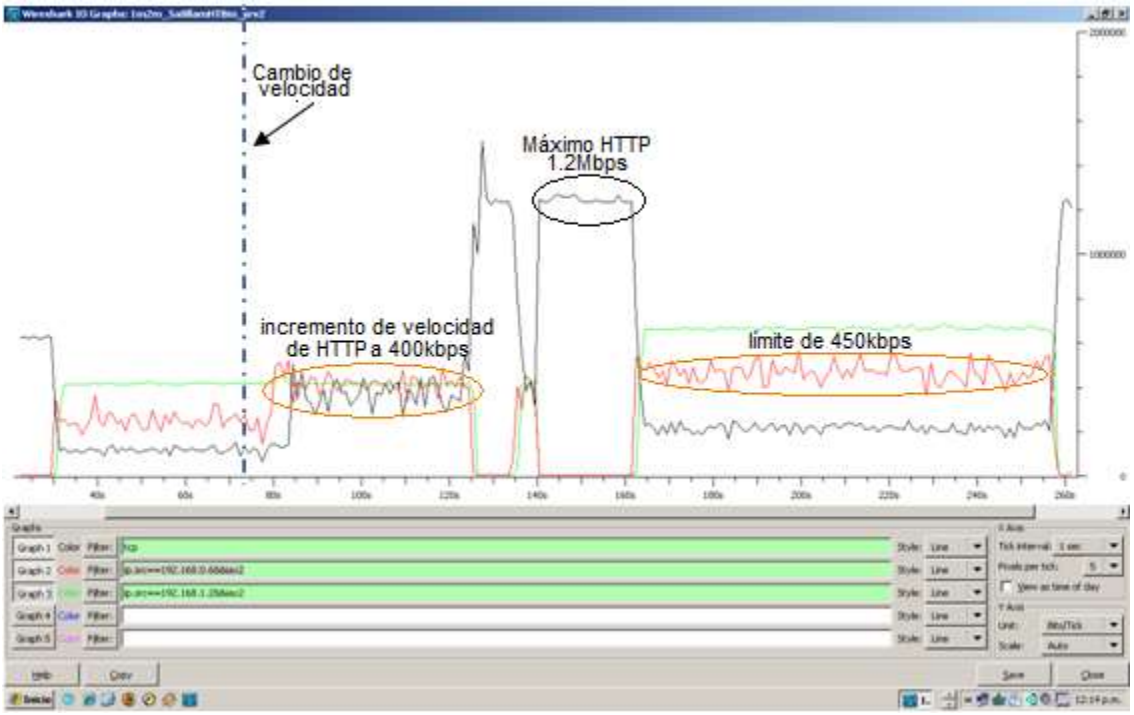


Figura 6.2 Tráfico de voz y HTTP con el algoritmo HTB modificado

- En las gráficas de tráfico de HTB y HTB modificado, El tráfico HTTP inicialmente alcanza una tasa de transmisión máxima cercana al *"throughput"*, que es aproximadamente un 60% de la velocidad del canal inalámbrico, ya que no hay otro tipo de tráfico.

- Cuando se inicia el tráfico de voz, el tráfico HTTP con HTB y HTB modificado, disminuye hasta una tasa de aproximadamente 120kbps, que es la tasa de transmisión establecida para ese tipo de tráfico.

- El cambio de velocidad del canal se indujo cerca de los 90 segundos para el algoritmo HTB y cerca de los 70 segundos para el algoritmo HTB modificado, a pesar de esto no se observan cambios en la tasa de transmisión del tráfico HTTP, tampoco en el tráfico entrante de voz en la gráfica del algoritmo HTB. En la gráfica de tráfico del algoritmo HTB modificado se puede notar un incremento en la tasa de transmisión de HTTP a aproximadamente 400kbps que corresponden a la tasa de transmisión de esta clase cuando la velocidad del canal es 2Mbps, más una cantidad que no está siendo utiliza por otras clases. También se observa un incremento en la tasa de transmisión del tráfico de voz desde cliente hacia el servidor.

- El tráfico de voz de subida empieza con un máximo aproximado a 400kbps en las dos gráficas de tráfico, con el algoritmo HTB esta tasa decae cerca del segundo 85 debido a la pérdida de algunas llamadas, lo que no sucede en la gráfica de tráfico de HTB modificado.

- Una vez se termina el primer grupo de llamadas el tráfico HTTP llega a un máximo de 600kbps en la gráfica de tráfico de HTB, a pesar del incremento de la velocidad, lo que significa que los recursos del canal adicionales, no están siendo aprovechados con este algoritmo. Con el algoritmo HTB modificado el tráfico HTTP llega a un máximo de aproximadamente 1.2Mbps, lo que significa que están utilizando los recursos del canal adicionales producto del incremento de velocidad.

- Cuando arranca el segundo grupo de llamadas el cliente transmite VoIP a aproximadamente 640kbps, esta es la velocidad necesaria para mantener 8 llamadas, aunque en la gráfica del tráfico de HTB esta tasa decae por la pérdida de algunas llamadas. En cambio con el algoritmo HTB modificado no se pierde ninguna llamada.

- El tráfico de voz que llega al cliente desde el servidor con HTB está alrededor de 250kbps, este sería el límite de tasa de transmisión de esta clase más una cantidad que no está siendo utilizada por las otras clases, considerando que los límites son establecidos cuando la velocidad del canal es de 1Mbps, con HTB modificado este tráfico está alrededor de 450kbps, esto supera el límite de este tipo de tráfico cuando la velocidad del canal es 1Mbps, lo que significa que las tasas de transmisión de las clases de tráfico están variando en función del *throughput* del canal.

En las tablas 6.1 y 6.2 se presentan los valores promedio de tráfico HTTP y de voz correspondiente a las pruebas realizadas

Algoritmo HTB original

	Antes del cambio de ancho de banda (1Mbps)				Después del cambio de ancho de banda (2Mbps)			
	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)
prueba 1	630	110	251	410	610	121	240	680
prueba 2	620	109	253	410	620	125	244	660
prueba 3	610	109	252	410	620	125	240	660
prueba 4	**620**	**115**	**245**	**415**	**620**	**115**	**245**	**660**
prueba 5	620	110	240	415	620	110	240	665
prueba 6	620	115	235	415	620	115	245	665

Tabla 6.1 Valores promedio de tráfico con HTB

Algoritmo HTB modificado

	Antes del cambio de ancho de banda (1Mbps)				Después del cambio de ancho de banda (2Mbps)			
	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)
prueba 1	640	119	246	410	1250	210	468	670
prueba 2	680	120	256	410	1270	215	470	670
prueba 3	650	122	254	410	1230	223	477	670
prueba 4	620	111	240	420	1280	219	460	660
prueba 5	**625**	**115**	**220**	**420**	**1240**	**220**	**490**	**670**
prueba 6	620	109	250	420	1250	220	470	660

Tabla 6.2 Valores promedio de tráfico con HTB modificado

En las tablas 6.1 y 6.2 se aprecia como en las pruebas realizadas utilizando el algoritmo HTB modificado se consigue un mejor uso de los recursos del canal inalámbrico, cuando este tiene un incremento en la velocidad de transmisión, ya que se incrementa la velocidad garantizada para cada clase. Contrario a los resultados obtenidos con el algoritmo HTB, en los que no se incrementó la tasa de transmisión de las clases de tráfico, a pesar del incremento en la velocidad de transmisión del canal inalámbrico, lo que significa que existen recursos de canal que no están siendo usados por las clases de tráfico.

Comparando la prueba número 4 de HTB y número 5 de HTB modificado, se puede notar que el promedio de tráfico de voz de bajada con el algoritmo HTB modificado (490kbps) es mucho mayor que el promedio de este mismo tráfico con el algoritmo HTB (245kbps). También se puede notar que la tasa máxima de transmisión de tráfico HTTP es mucho

mayor con el algoritmo HTB modificado (1240kbps) que el que se obtiene utilizando el algoritmo HTB (620kbps). En general, en todas la pruebas se observa que después del cambio de ancho de banda, el máximo de transmisión de datos se incrementa cuando se utiliza el algoritmo HTB modificado, demostrando que el parámetro “ceil” de las clases de tráfico se actualiza exitosamente, posibilitando una tasa de transmisión de datos mayor cuando no hay otro tipo de tráfico.

6.1.2 Análisis de retardo y jitter

En este análisis se compararán parámetros de calidad de servicio de voz, entre las pruebas realizadas con el algoritmo HTB y el algoritmo HTB modificado, para determinar las diferencias que se pueden encontrar entre los algoritmos con respecto al tratamiento de los paquetes de voz.

En las figuras 6.3 y 6.4 se presentan los valores de retardo y *“jitter”* de los paquetes de voz enviados desde el cliente al servidor, tomados de la prueba realizada para el algoritmo HTB y HTB modificado, en las gráficas las líneas rojas representan el retardo y las líneas negras representan el *“jitter”*.

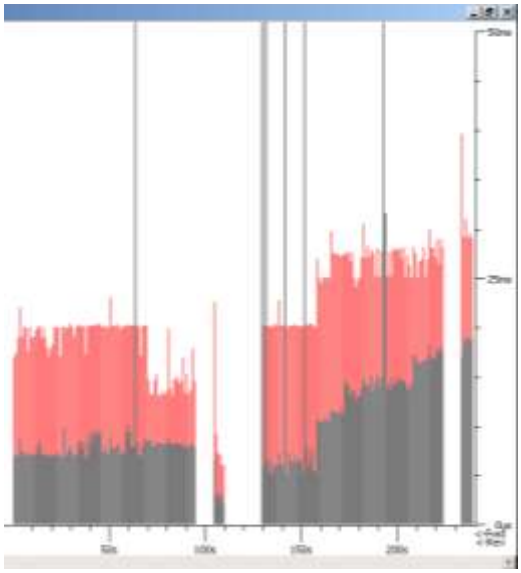


Figura 6.3 Jitter y retardo de cliente a servidor con HTB

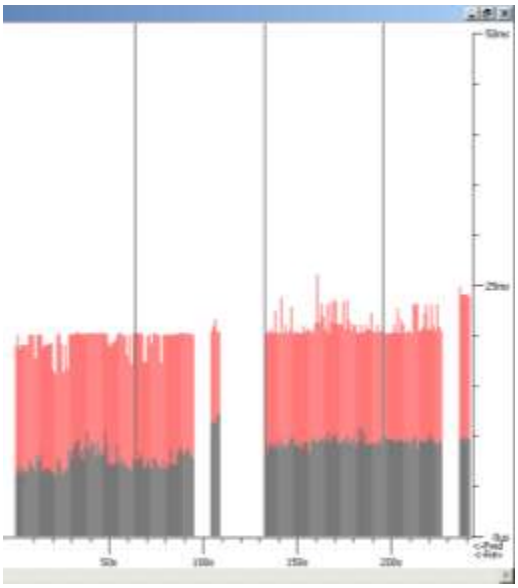


Figura 6.4 Jitter y retardo de cliente a servidor con HTB modificado

Como se observa en la gráfica 6.3 y 6.4, el retardo de los paquetes no es muy grande, en la primera parte de las pruebas es cercano a los 20 ms para los dos algoritmos, aunque en el segundo grupo de llamadas este valor se incrementa hasta aproximadamente 30 ms con HTB, mientras que con HTB modificado se mantiene en 20 ms. De igual manera el *"jitter"* en el segundo grupo de llamadas es menor con HTB modificado, siendo aproximadamente de 10 ms mientras que con HTB alcanza 17 ms.

Además, es necesario considerar que el cliente únicamente transmite voz y los ACK del tráfico de datos, produciendo un menor retardo de los paquetes de voz en las colas de transmisión.

En las figuras 6.5 y 6.6 se presentan los valores de retardo y *"jitter"* de los paquetes de voz enviados desde el servidor al cliente, tomados de la pruebas realizadas para el algoritmo HTB y HTB modificado respectivamente, en las gráficas las líneas azules representan el retardo y las líneas verdes representan el *"jitter"*.

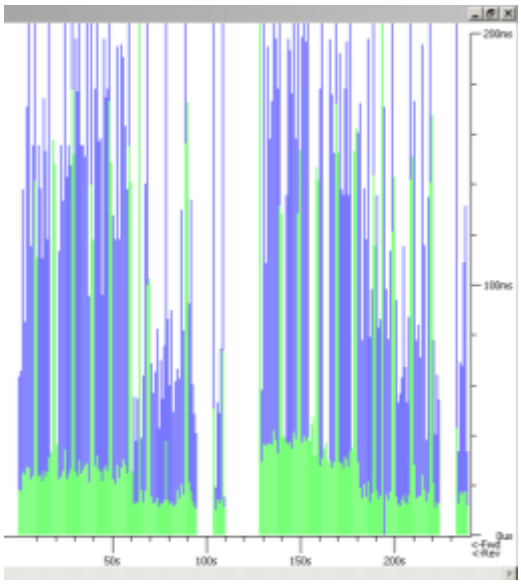


Figura 6.5 Jitter y retardo de servidor a cliente con HTB

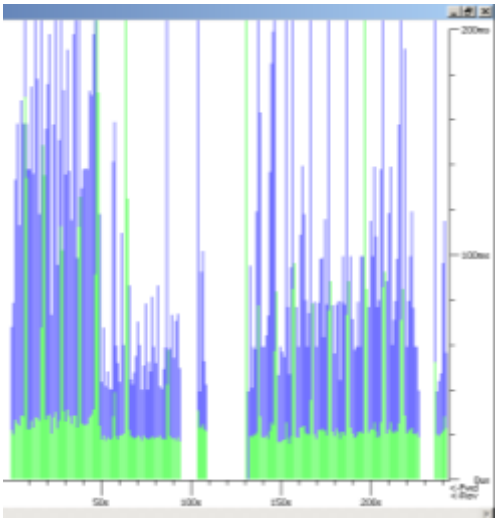


Figura 6.6 Jitter y retardo de servidor a cliente con HTB modificado

En la figura 6.5 y 6.6 se observa que el retardo del primer grupo de llamadas es mayor a 150ms, sobrepasando el límite para una buena calidad de voz, con HTB el segundo grupo de llamadas algunos paquetes superan los 150ms de retardo justo antes de que se pierdan algunas llamadas causando una disminución en el retardo.

Algunos paquetes del segundo grupo de llamadas alcanzan un *"jitter"* cercano a los 40ms lo que puede afectar la calidad del estas llamadas teniendo en cuenta el límite de 30ms mencionado en el numeral 1.2. Con HTB modificado se puede notar que el retardo de la mayoría de paquetes del segundo grupo de llamadas es menor a 150ms, lo que es importante considerando que en esta prueba se mantienen 8 llamadas hasta el final. De

igual forma el *"jitter"* se mantiene estable en aproximadamente 20ms, una diferencia importante con la gráfica de HTB donde el *"jitter"* alcanzó un valor cercano a los 40ms.

En general se observa una mejora en los valores de retardo y jitter de los paquetes de voz, sobre todo considerando que utilizando el algoritmo HTB modificado no se pierden llamadas y los valores de estos parámetros son aceptables, teniendo en cuenta los limites mencionados en el numeral 1.3.2.

6.2 ANÁLISIS DE RESULTADOS DE LAS PRUEBAS DEL ALGORITMO HTB MODIFICADO CON DECREMENTO EN LA VELOCIDAD

A continuación se presenta un análisis de los resultados de la prueba descrita en el numeral 5.2.2 con un decremento en la velocidad de transmisión del canal, utilizando las gráficas de tasas de transmisión, retardo y jitter proporcionadas por la herramienta de monitoreo WireShark.

6.2.1 Análisis de throughput

En las figuras 6.7 y 6.8 se presentan las gráficas de comportamiento del tráfico de voz y datos utilizando el algoritmo HTB y HTB modificado respectivamente.

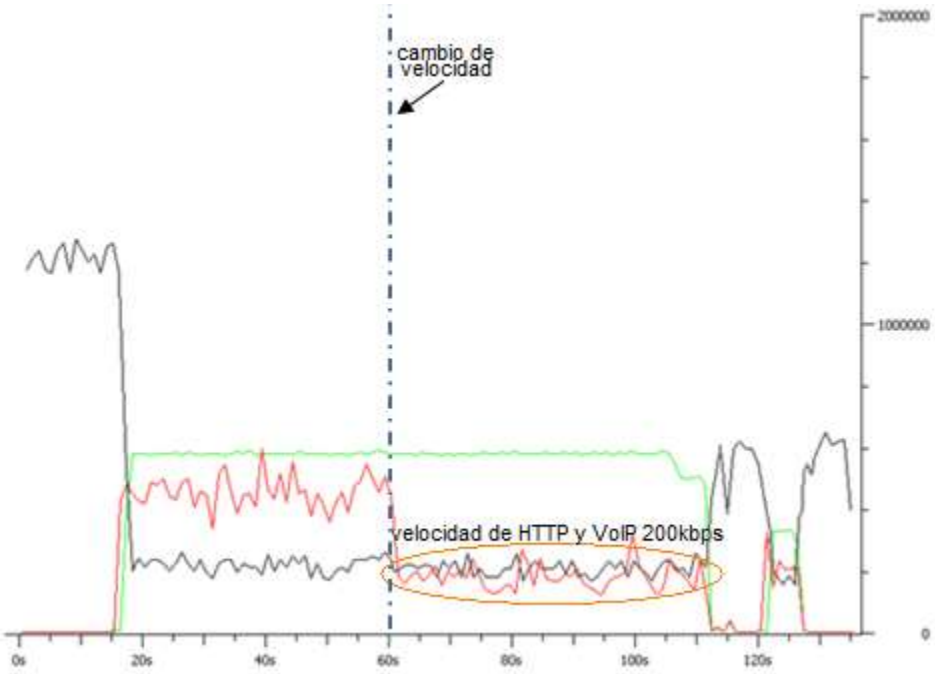


Figura 6.7 Tráfico de voz y HTTP con el algoritmo HTB

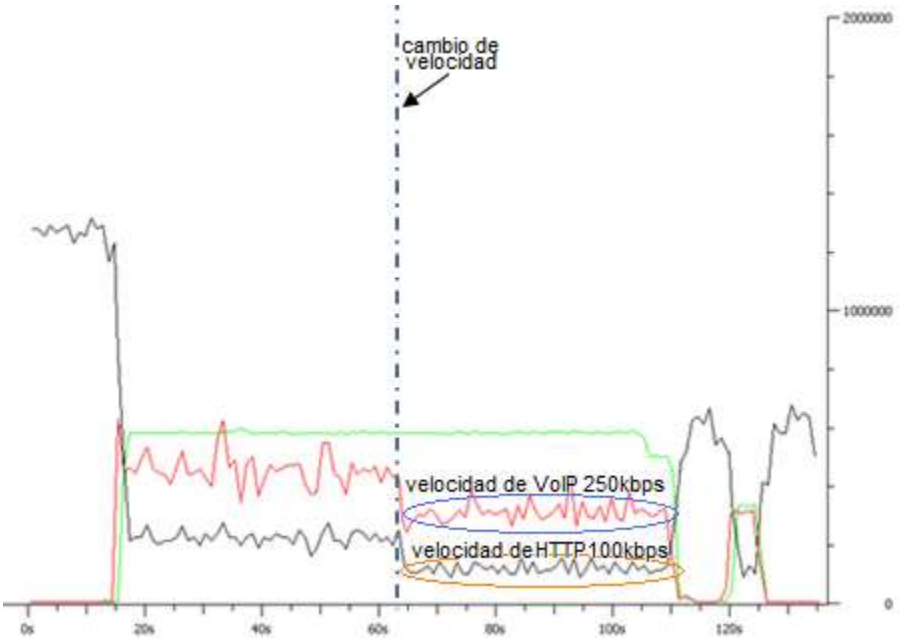


Figura 6.8 Tráfico de voz y HTTP con el algoritmo HTB modificado

- El tráfico HTTP inicialmente alcanza una tasa de transmisión máxima cercana 1.2Mbps para el HTB y HTB modificado, aproximadamente un 60% de la velocidad del canal inalámbrico, ya que no hay otro tipo de tráfico.

- Cuando se inicia el tráfico de voz, el tráfico HTTP disminuye hasta una tasa de 230kbps para el HTB y HTB modificado, que es la tasa de transmisión establecida para ese tipo de tráfico para un ancho de banda de 2Mbps.

- El cambio de velocidad del canal se indujo cerca de los 60 segundos para los dos algoritmos, a pesar de esto no se observan grandes cambios en la tasa de transmisión del tráfico HTTP con el algoritmo HTB, aunque si se ve un decremento en el tráfico de voz que llega desde el servidor. Con el algoritmo HTB modificado se aprecia que a partir del cambio de velocidad del canal, la tasa de transferencia del tráfico HTTP disminuyó aproximadamente 100kbps que sería el límite de tasa de transferencia de esta clase cuando la velocidad del canal es 1Mbps.

- Con el algoritmo HTB, una vez se induce el cambio de velocidad del canal se observa que el tráfico VoIP decrece hasta aproximadamente 200kbps a pesar de que esta clase tiene reservado 420kbps cuando la velocidad del canal es 1Mbps, lo que significa que el tráfico HTTP está utilizando más recursos perjudicando el tráfico VoIP. Con el algoritmo HTB modificado el tráfico de voz de decrece hasta 250 kbps aproximadamente

- El tráfico de voz de subida inicia con un máximo aproximado a 600kbps el cual se mantiene a pesar del cambio de velocidad del canal, debido a que esta es la tasa de transmisión necesaria para mantener 7 llamadas.

- Cuando el tráfico de voz termina, el tráfico HTTP alcanza una tasa de transmisión máxima cercana 600kbps para los dos algoritmos, lo máximo que se puede alcanzar considerando que la velocidad del canal es 1Mbps.

En las tablas 6.3 y 6.4 se presentan los valores promedio de tráfico HTTP y de voz correspondiente a las pruebas realizadas

Algoritmo HTB original

	Antes del cambio de ancho de banda (2Mbps)				Después del cambio de ancho de banda (1Mbps)			
	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)
prueba 1	1200	220	440	580	610	210	180	580
prueba 2	**1180**	**210**	**420**	**580**	**630**	**210**	**210**	**580**
prueba 3	1210	220	440	580	610	220	190	580
prueba 4	1200	220	450	580	620	210	200	580
prueba 5	1220	230	450	580	630	220	200	580
prueba 6	1200	210	420	580	620	210	200	580

Tabla 6.3 Valores promedio de tráfico con HTB

Algoritmo HTB modificado

	Antes del cambio de ancho de banda (2Mbps)				Después del cambio de ancho de banda (1Mbps)			
	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)	máximo tráfico HTTP (Kbps)	tráfico HTTP (Kbps)	tráfico de voz de bajada (Kbps)	tráfico de voz de subida (Kbps)
prueba 1	**1230**	**210**	**430**	**580**	**640**	**120**	**250**	**580**
prueba 2	1240	220	440	580	620	110	240	580
prueba 3	1220	210	415	580	640	120	250	580
prueba 4	1250	220	450	580	630	115	220	580
prueba 5	1230	220	420	580	610	110	240	580
prueba 6	1200	210	410	580	630	120	230	580

Tabla 6.4 Valores promedio de tráfico con HTB modificado

Comparando la prueba 2 y 1 de HTB y HTB modificado respectivamente, en las tablas 6.3 y 6.4 se aprecia como después del cambio de velocidad del canal, el promedio de la tasa de transmisión de tráfico de voz de bajada con el algoritmo HTB modificado (210kbps) es mayor que el promedio de este mismo tráfico con el algoritmo HTB (250kbps). También se puede notar que el promedio de la tasa de transmisión de tráfico HTTP es menor con el

algoritmo HTB modificado (120kbps) que el que se obtiene utilizando el algoritmo HTB (210kbps). En general se tiene una distribución de ancho de banda más apropiada cuando se utiliza el algoritmo HTB modificado, es decir que el tráfico de voz utiliza más recursos, afectando al tráfico de datos.

6.2.2 Análisis de jitter y retardo

En las figuras 6.9 y 6.10 se presentan los valores retardo y *"jitter"* de los paquetes de voz enviados desde el servidor al cliente, tomados de la prueba realizada para el algoritmo HTB y algoritmo HTB modificado, en las gráficas las líneas azules representan el retardo y las verdes representan el *"jitter"*.

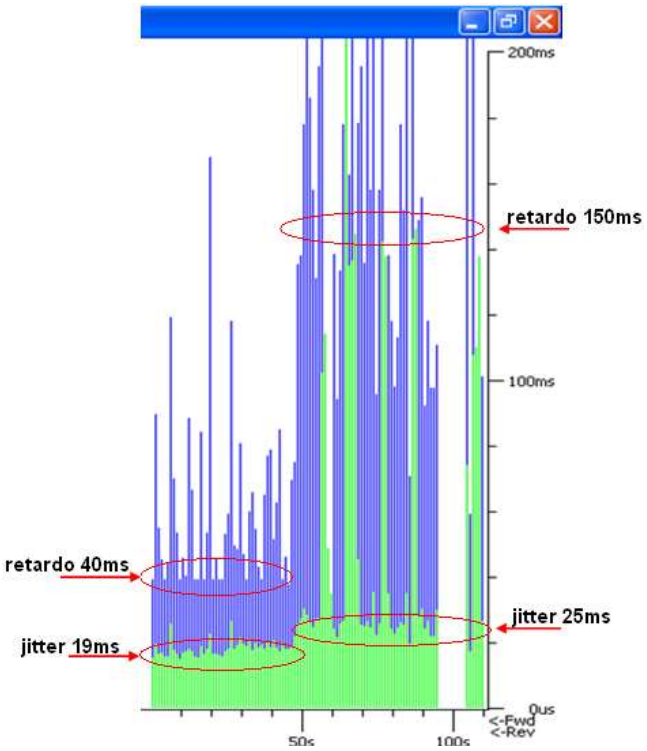


Figura 6.9 Jitter y retardo de servidor a cliente con HTB

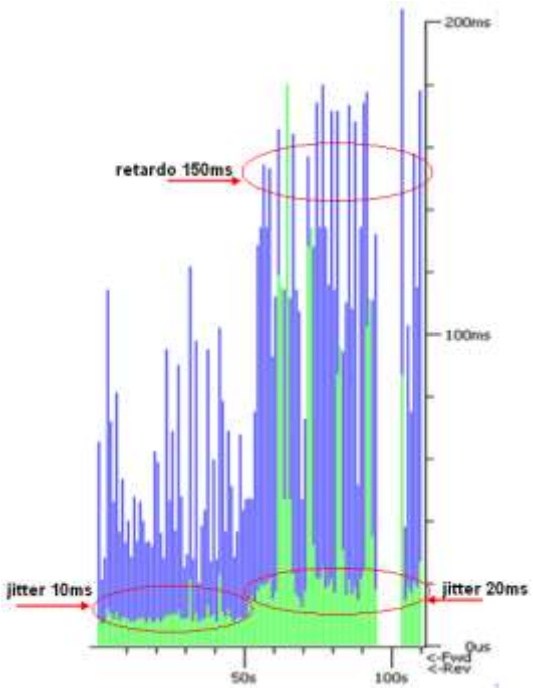


Figura 6.10 Jitter y retardo de servidor a cliente con HTB modificado

En la figura 6.9 el retardo y el *"jitter"* en la primera parte del grupo de llamadas no sobrepasa los 40ms y 20ms respectivamente, sin embargo después de inducir el cambio de velocidad del canal se incrementa el retardo hasta más de 110ms y el jitter aproximadamente 30ms.

En la figura 6.10 se observa que el retardo y el jitter de la mayoría de paquetes, en la primera parte del grupo de llamadas no sobrepasa los 40ms y 10ms respectivamente, incrementándose aproximadamente hasta 110ms y 30ms después de inducir el cambio de velocidad del canal, aunque en esta prueba es menor la cantidad de paquetes que alcanzan estos valores, lo que se nota en la cantidad de líneas que superan los 150ms.

En las figuras 6.11 y 6.12 se presentan los valores retardo y *"jitter"* de los paquetes de voz enviados desde el cliente al servidor, tomados de la prueba realizada para el algoritmo HTB y algoritmo HTB modificado, en las gráficas las líneas rojas representan el retardo y las negras representan el *"jitter"*.

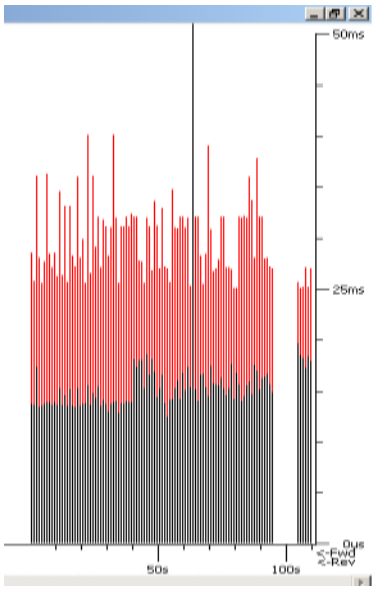


Figura 6.11 Jitter y retardo de cliente a servidor con HTB

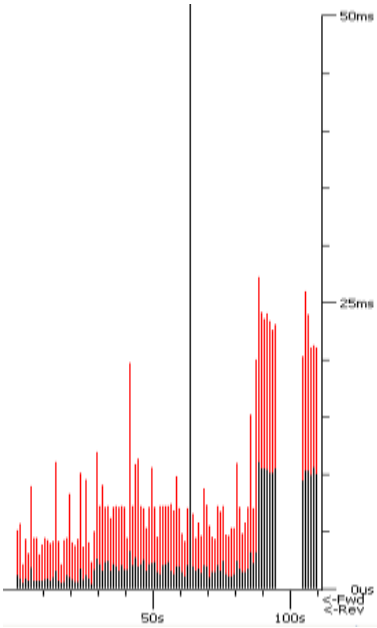


Figura 6.12 Jitter y retardo de cliente a servidor con HTB modificado

En la figura 6.11 y 6.12 el retardo y el *"jitter"* en el transcurso del grupo de llamadas no sobrepasa los 30ms y 20ms respectivamente. Estos valores se deben a que el cliente transmite tráfico de voz y ACK del tráfico HTTP, permitiendo que el tráfico de voz de subida utilice mayor ancho de banda que el tráfico de voz de bajada.

En este análisis de puede notar una leve mejoría en los valores de retardo y jitter de los paquetes de voz cuando se utiliza el algoritmo HTB modificado, a causa de que este

realiza una mejor distribución de recursos cuando se presenta una disminución en la velocidad de transmisión del canal.

7. CONCLUSIONES Y RECOMENDACIONES

7.1 CONCLUSIONES

Se demostró que el algoritmo HTB no es adecuado para enlaces inalámbricos, ya que las tasas de transmisión de las clases de tráfico no varían en función de la velocidad del canal, en consecuencia la asignación de recursos puede ser incorrecta. Esto evidencia la necesidad de modificar el algoritmo HTB para que se adapte a los cambios en la velocidad de transmisión del enlace inalámbrico.

Fue posible modificar el algoritmo HTB para que adapte las tasas de transmisión de las clases de tráfico, gracias a que se identificó la variable de la que dependía esta asignación de recursos, además se dispuso del código fuente y posibilidad de incluir este código en el firmware del enrutador inalámbrico.

En las pruebas realizadas se demostró que utilizando el algoritmo HTB modificado se pueden mantener un número mayor de llamadas, debido al reajuste de la tasa de transmisión de este tipo de tráfico, cuando la velocidad del canal inalámbrico varía.

En los resultados obtenidos se evidencia que con el algoritmo HTB modificado se aprovechan de manera más eficiente los recursos del canal, cuando este incrementa su tasa de transmisión, de igual forma se consigue una distribución más adecuada de los recursos del canal cuando se presenta una disminución en la tasa de transmisión del mismo. Aunque la modificación realizada al algoritmo HTB es aparentemente sencilla es el producto de varios meses de investigación y análisis del código fuente del algoritmo, y su posterior inclusión en el firmware del enrutador inalámbrico.

El algoritmo HTB modificado, implementado en este proyecto se convierte en una buena alternativa para optimizar el rendimiento de enlaces 802.11, haciendo posible dar soporte a nuevos servicios en este tipo de redes. Debido a que puede implementarse en enrutadores inalámbricos tengan firmware basado en el kernel de Linux.

Utilizando el algoritmo HTB en conjunto con el filtro U32 es posible implementar funciones como restricciones de ancho de banda, reserva de ancho de banda, distribución de ancho de banda, a nivel de redes subredes, direcciones IP y protocolos. Aunque cabe mencionar que el control de tráfico de Linux posee más disciplinas de colas y otros tipos de filtros.

Los enrutadores con firmware basado en el kernel de Linux son herramientas importantes en la implementación de calidad de servicio, ya que permiten implementar tareas basadas en el manejo de colas, como definir clases de tráfico, políticas y filtros. Además es posible modificar funciones, posibilitando el desarrollo de futuros proyectos de investigación en redes inalámbricas.

Las comunicaciones de VoIP en las redes inalámbricas tienen límites rigurosos, esto quiere decir que cuando se supera la capacidad del canal inalámbrico, no es posible incrementar el número de comunicaciones de voz. Por este motivo es preferible utilizar el algoritmo HTB modificado ya que este asigna los recursos del canal inalámbrico adaptándose a las variaciones de ancho de banda de las redes inalámbricas, asignando mayor capacidad al tráfico de voz si la velocidad del canal se incrementa posibilitando establecer un número mayor de comunicaciones.

Es importante utilizar un dispositivo que admita cambios de *"firmware".* Dadas las condiciones del mercado de infraestructura en redes, es conveniente adquirir hardware 802.11 que se enmarque bajo las políticas de software libre.

Finalmente, aunque en principio realizar el proceso de pruebas de este proyecto pareciera una tarea "sencilla", este requiere una minuciosa planeación, observación y recursividad, pues lo que más retardó la culminación del proyecto fue la necesidad de aprender a instalar, configurar y poner en funcionamiento los programas necesarios, además de asimilar el proceso de compilación del firmware utilizado.

7.2 RECOMENDACIONES

Se sugieren como trabajos complementarios a este:

Desarrollo de un sistema de monitoreo de ancho de banda, basado en los tiempos de transmisión de tramas, integrando lo con el algoritmo HTB modificado desarrollado en este proyecto, con el fin de confrontar los resultados obtenidos en este trabajo**.**

Realizar una modificación que permita una asignación dinámica de los recursos del canal inalámbrico para una red punto multipunto, en la que el monitor de ancho de banda calcule el *throughput* de cada estación y realice una asignación basada en estos cálculos.

REFERENCIAS

[1] Ferguson P., Huston G., "*Quality of Service: Delivering QoS on the Internet and in Corporate Networks"*. John Wiley & Sons, 1998

[2] Charny A., Le Boudec, Y., "Delay bounds in a network with aggregate scheduling. In *Quality of Future Internet Services", First COST 263 International Workshop(QofIS 2000)*, pages 1–13, Berlin, Germany. Springer, 2000

[3] de Veciana G., Walrand J., "*Bandwidths: Call admission, traffic policing and filtering for ATM networks".* University of California at Berkeley, Berkeley, 1994.

[4] Marcon M., Dischinger M., Gummadi K., Vahdat A., "*The Local and Global Effects of Traffic Shaping in the Internet".* University of California, San Diego.

[5] Park Kun I. Ph.D, "QoS in Packet Networks", Springer, 2005

[6] Kulkarni V.G., Gautam N., "*Leaky buckets: Sizing and Admission Control"*. University of North Carolina, Departament of Operations Research, 1996

[7] Fingerhut A.J.,Varghese G., "Randomized Token Buckets: Reducing the Buffers Required in Multiplexors". Washington University

[8] Cardei, Michaela. CARDEI, Iount. DU, Ding-Zhu. "Resource Management in Wireless Networking". Springer, 2005

[9] Daigle, John. "Queuing Theory with application to packet telecommunications". Springer Science+Business Media, 2005.

[10] Peterson L., Bruce S. D., "*Computer networks: a systems approach".* Springer, 2007.

[11] Shreedhar M., Varghese G., "Efficient fair queueing using deficit round robin". In *Proceedings of SIGCOMM '95*, pages 231–242, 1995.

[12] Kwak J., Nam J., Kim D., "A Modified Dynamic Weighted Round Robin Cell Scheduling Algorithm", 2004.

[13] Shimonishi H., Suzuki H., "Performance Analysis of Weighted Round Robin Cell Scheduling and its Improvement in ATM Networks", *IEICE Trans. Comm*., vol. E81 B, 1998.

[14] Mezger K., Petr D., "Bounded delay for Weighted Round Robin". University of Kansas, Departament of Electrical Engineering and Computers Sciences, 1995.

[15] Wesel E., "*Wireless Multimedia Communications - Networking Video, Voice, and Data"*. Addison-Wesley, 1997.

[16] Sierra R., "Fair queuing in data networks", 2002.

[17] Lu S., Bharghavan V., Srikant R., "Fair scheduling in wireless packet networks". *Proceedings of ACM SIGCOMM*, 1997.

[18] Nandagopal T., Lu S., Bharghavan V., "A unified architecture for thedesign and evaluation of wireless fair queueing algorithms". *MobiCom'99 - Proceedings of The Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 132–142, 1999.

[19] Eugene T., Stoica I., Zhang H., "Packet fair queueing algorithms for wireless networks with location-dependent errors". *Proceedings of IEEE INFOCOMM*, page 1103, 1998.

[20] Eugene T., Stoica I, Zhang H. "Packet fair queueing algorithms for wireless networks with location-dependent errors". Technical Report CMU-CS-00-112, School of Computer Science, Carnegie Mellon University, 2000.

[21] Goyal P., Vin H., Cheng H., "Start-time fair queuing: A scheduling algorithm for integrated servicespacket switching networks". Technical Report CS-TR-96-02, The University of Texas at Austin, Department of Computer Science, 1996.

[22] Lu S., Nandagopal T., Bharghavan V., "Design and analysis of an algorithm for fair service in error-prone wireless channels". *Wireless Networks Journal*, 1999.

[23] Parekh A., Gallager R., "A generalized processor sharing approach to flow control in integrated service networks: the multiple node case", 1994.

[24] Floyd S., Jacobsen V., "Link-sharing and resource management models for packet networks". *IEEE/ACM Transactions on Networking*, 3(4), 1995.

[25] Floyd S., "Notes on cbq and guaranteed service", 1995.

[26] Información disponible en el sitio Web: http://luxik.cdi.cz/~devik/qos/htb/.

[27] Brown M. A., "Traffic Control using tcng and HTB HowTo", Version 1.0, 2003.

[28] Información disponible en el sitio Web: http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm

[29] Wallingford T., "Switching to VoIP", 2005.

[30] Documento disponible en el sitio WEB: http://www.faqs.org/rfcs/rfc5456.html.

[31] Pietrosemoli E., VoIP, Escuela Latinoamericana de Redes, 2003.

[32] Documento disponible en el sitio web: http://ftp.tiaonline.org/TR-41/tr4112inactive/Public/Latest_Revision_of_PN-4689/PN4689LB.pdf.

[33] Documento disponible en el sitio web:
http://www.itu.int/rec/T-REC-G.113-200711-I/en

[34] Johnson D. “Binary Phase Shift Keying”.
Disponible en línea en: http://cnx.org/content/m10280/2.13/?format=pdf. 2007.

[35] Ruque J., Ruiz D., Carrión C., “Simulation and implementation of the BPSK modulation on a FPGA Xilinx Spartan 3 xcs200-4ftp256, using Simulink and the System Generator blockset for DSP/FPGA”. Technical University of Loja, Group of Electricity and Electronic Systems.

[36] “Modulación Digital FSK − PSK – QAM”. Disponible en línea en: http://www.mundodescargas.com/apuntestrabajos/electronica_electricidad_sonido/decargar_modulacion-digital.pdf.

[37] Faúndez M., “Sistemas de comunicaciones”, 2001.

[38] “Modulación de amplitud en cuadratura”. Disponible en línea en: http://es.wikipedia.org/wiki/Modulaci%C3%B3n_de_amplitud_en_cuadratura. 2010.

[39] Kowalski R., “The Benefits of Dynamic Adaptive Modulation for High Capacity Wireless Backhaul Solutions”, 2007.

[40] Mundra P., Kapur R., “The choise of a digital modulation scheme in a mobile radio system”. Electronics Research & Development Centre, SAS Nagar, India.

[41] Leiva N., Neira A., “WiMAx, *Worldwide Interoperability for Microwave Access”.*

[42] Sam W., “Adaptive Modulation (QPSK, QAM)”. 2004.

[43] “Upper bound performance of adaptive modulation in a slow Rayleigh fading channel”, *IEEE Electronics Letters*, Vol. 32, pg 718, 1996.

[44] Buehrer M., Woerner B., Pratt T., “Channel prediction for adaptative modulation in wireless communications. Blacksburg”, Virginia, 2003.

[45] Devera M., “Hierachical token bucket theory”. 2002.

[46] Guerrero L., Hurtado M., “Control de Tráfico Mediante Gestión Dinámica de Ancho de Banda en 802.11”. Universidad del Cauca, Departamento de Telecomunicaciones, 2007.

[47] OpenWrt Buildroot
Disponible en:http://downloads.openwrt.org/whiterussian/docs/buildroot-documentation.html.

[48] OpenWRT, TableOfHardware. Información disponible en sitio web:
http://wiki.openwrt.org/TableOfHardware.

[49] Asadoorian P., Pesce L., “Linksys WRT54G Ultimate Hacking”. Syngress, 2007.

[50] Pont F.,” OPENWRT. Linux per a dispositius embedded”, 2009.

[51] Información disponible en el sitio Web:
http://oldwiki.openwrt.org/OpenWrtDocs%282f%29KamikazeConfiguration.html

[52] Port tiggering with openwrt.
Disponible en. http://www.elbeno.com/openwrt/openwrt_porttrigger.html

[53] Squashfs LZMA.
Disponible en: http://www.squashfs-lzma.org/

[54] Devera M., “HTB Linux queuing discipline manual - user guide”. 2002.

ANEXO 1: CONFIGURACIÓN DEL PLANIFICADOR DE TRÁFICO HTB

El código de los *script* del algorimo HTB original se encuentra en [26]

1. HERRAMIENTA TC DE IPROUTE2

Iproute2 es una serie de comandos de línea para manipular las estructuras del kernel de Linux para enrutamiento IP dentro de equipo.

Los principales componentes del control de tráfico en Linux son [45]:

1.1 Qdisc

Un qdisc es un planificador. Cada interfaz de red necesita de un planificador de algún tipo, el planificador por defecto es FIFO. Otros tipos de planificadores disponibles en Linux organizan los paquetes entrantes de acuerdo a las políticas del planificador.

Los qdisc pueden contener clases, lo que permite un medio para asociar filtros. Aunque no es obligatorio utilizar un qdic con clases hijas, pero esto desperdiciará recursos del sistema.

1.2 Class

Las clases sólo existen dentro de un qdisc, son flexibles y pueden contener multiples clases hijas o un único qdisc hijo. También pueden tener un número arbitrario de filtros asociados, esto permite la selección de una clase hija o el uso de un filtro para reclasificar o transmitir tráfico entrante a una clase en particular.

Una clase de la hoja es una clase terminal. Contiene un qdisc (por defecto FIFO) y no tiene clases hijas. Las clases que tienen clases hijas son clases internas, a continuación se muestra un ejemplo de implementación de una clase HTB.

1.3 Filter

El filtro es el componente más complejo del control de tráfico en el Linux. La función más simple y más obvia del filtro es clasificar los paquetes. Los filtros de Linux le permiten al usuario clasificar los paquetes en una cola de salida con varios filtros diferentes o un solo filtro.

El filtro debe asociarse al qdisc o a las clases, sin embargo los paquetes encolados siempre entran primero al qdisc raíz. Después de que el filtro asociado al qdisc de la raíz se ha cruzado, el paquete puede dirigirse a cualquier subclase.
Un filtro debe tener una sentencia para clasificación y puede o no tener una sentencia para vigilancia. Debido a que este interactúa con el kernel para la creación, destrucción y modificación de las estructuras de control de tráfico, el binario de tc debe ser compilado con el soporte de todos los qdisc. La herramienta tc realiza toda la configuración de las estructuras del kernel requeridas para control de tráfico. La herramienta toma la primera opción como uno de los componentes del control de tráfico, "qdisc", "class" o "filter" a continuación se muestra un ejemplo de uso del comando tc.

#tc [opción] [objeto]

- Las opciones del comando tc son: -s(estadísticas), -d(detalles)

- Los objetos del comando tc son: qdisc, class, filter

Cada objeto acepta diferentes opciones las que se describirán a continuación.
Ejemplo de implementación de qdisc HTB
#tc qdisc add dev eth0 root handle 1:0 htb

- *"add"* añade una disciplina de colas, también se puede usar "del" para quitar una disciplina de colas.

- *"dev eth0"* especifica a que interfaz estará asociado el qdisc.

- *"root"* Esto significa salida al tc.

- *"handle 1:0"* se usa para especificar el número de identificación del objeto en formato mayor:menor.

- *"htb"* es la diciplina de colas asociada al qdisc.

2 CONFIGURACIÓN DE LAS CLASES DE TRÁFICO

A continuación se muestra en ejemplo para configurar una clase dentro de un planificador HTB, en la que se da una tasa asegurada de transmisión de 265kbit, y una tasa máxima de transmisión de 512kbit, con un identificador 1:1 [54].

#tc add dev eth0 parent 1:1 classid 1:6 htb rate 265kbit ceil 512kbit

- *"parent 1:1"* especifica la identificación de clase padre de esta clase.

- *"classid 1:6"* identificación de la clase.

- *"htb"* especifica la disciplina de colas, las clases deben tener la misma disciplina de la clase padre.

- *"rate 265kbit ceil 512kbit"* estos son los parámetros de configuración de la clase HTB

2.1 Parámetros de configuración de las clases HTB

Los parámetros de configuración de las clases HTB son [45], [54]:

2.1.1 Parámetro *"default"*

Es un parámetro opcional en el qdisc HTB, el valor de este parámetro por defecto es 0, que causa que cualquier tráfico no clasificado sea desencolado a la velocidad del hardware. Desviándose de cualquier clase asociada al qdisc raíz [45].

2.1.2 Parámetro *"rate"*

Se usa para establecer la tasa mínima de transmisión de la clase, esta puede considerarse equivalente a la tasa garantizada (CIR commited informtion rate), o el ancho de banda garantizado de una clase dada.

2.1.3 Parámetro *"ceil"*

Se usa para establecer el la tasa máxima de transmisión del la clase, el ancho de banda prestado por otras clases a la clase que transmite no debe superar el valor de ceil.

2.1.4 Parámetro *"burst"*

El hardware de red únicamente puede enviar un paquete al tiempo y depende solo de la tasa del hardware. Los software de enlaces compartidos pueden únicamente usar esta avilidad para aproximar los efectos de multiples enlaces corriendo a diferentes velocidades. Sin embargo rate y ceil no son en realidad medidas instantáneas sobre el tiempo que toma mandar muchos paquetes. Lo que realmente pasa es que el tráfico de una clase está enviando unos pocos paquetes al tiempo a la máxima velocidad y las otras clases esperan para enviar. Los parámetros burst y cburst controlan la cantidad de datos que pueden enviar a la máxima tasa del hardware sin intentar servir a otras clases. Si cburst es pequeño (preferiblemente del tamaño del paquete) se forman ráfagas para no exceder el ceil.

2.1.5 Parámetro *"quamtum"*

Este es un parámetro clave en el modelo de préstamos de ancho de banda, normalmente el quamtum es calculado por el algoritmo de HTB y no es especificado por el usuario, una variación pequeña en este parámetro puede tener grandes efectos en el préstamo de

ancho de banda, ya que este es usado para establecer la tasa de transmisión por encima de rate y por debajo de ceil.

3. CONFIGURACIÓN DE LOS FILTROS DE TRÁFICO

A continuación se definen dos elementos de los filtros de tráfico, y se muestra un ejemplo de implementación en el que se crea un filtro, el cual clasifica paquetes que utilizan el puerto 22 y está asociado a la clase con identificación 1:6.
Clasificador (classifier)
Los clasificadores son herramientas que pueden usarse como parte del filtro para identificar características de un paquete.
Los objetos del filtro que pueden manipularse usando el comando tc, se pueden usar varios diferente mecanismos de clasificación, el más común es el clasificador u32. El clasificador u32 permite al usuario seleccionar paquetes basados en características del paquete [54].

3.1 Vigilancia (policer)

Este mecanismo sólo se usa en el control de tráfico de Linux como parte de un filtro. Un policer llama una acción arriba y otra acción debajo de la tasa especificada. Es decir que se usa para limitar el uso de ancho de banda de una clase, aunque este no retrasa el tráfico, únicamente realiza una acción dependiendo de los criterios especificados.

El policer es un elemento básico del control de tráfico para limitar el uso de ancho de banda de mando de tráfico por limitar el uso del bandwidth un policer nunca tardará el tráfico

#tc filter add dev eth0 parent 1:0 protocol ip prio 5 match ip port 22 0xffff flowid 1:6.

- "*parent 1:0*": Especifica la identificación de la clase padre a la que se asociara el nuevo filtro

- "*prio 5*": este parámetro permite dar a un filtro más importancia sobre otro

- "*5 match ip port 22 0xffff*": estos son los parámetro con los que se clasificaran los paquetes, en este caso son los paquetes con el puerto 22 serán seleccionados por este comando.

- "*flowid 1:6*": este parámetro indica la identificación de la clase en la que serán encolados los paquetes que coincidan con los parámetros de clasificación.

4. CÓDIGO DE IMPLEMENTACIÓN HTB

```
#!/bin/bash

GREP=/bin/grep
INSMOD=/sbin/insmod
TC=/usr/sbin/tc
DEV=eth1
trg=1000kbps


RATE=$((($trg * 60) / 100))
porcentajeA=$(($RATE * 7))
porcentajeB=$(($RATE * 2))
porcentajeC=$(($RATE * 1))
# Cargar modulos delkernel
$GREP -q ^sch_htb /proc/modules || $INSMOD /lib/modules/`uname -r`/sch_htb.o
$GREP -q ^sch_sfq /proc/modules || $INSMOD /lib/modules/`uname -r`/sch_sfq.o
$GREP -q ^cls_u32 /proc/modules || $INSMOD /lib/modules/`uname -r`/cls_u32.o

#clase raiz HTB
$TC qdisc add dev $DEV root handle 1: htb default 10
$TC class add dev $DEV parent 1: classid 1:1 htb rate ${RATE}kbit burst 20k
cburst 20k

#clases HTB
$TC class add dev $DEV parent 1:1 classid 1:10 htb rate $(($porcentajeA /
10))kbit ceil ${RATE}kbit burst 15k cburst 15k
$TC class add dev $DEV parent 1:1 classid 1:20 htb rate $(($porcentajeB /
10))kbit ceil ${RATE}kbit burst 15k cburst 15k
$TC class add dev $DEV parent 1:1 classid 1:30 htb rate $(($porcentajeC /
10))kbit ceil ${RATE}kbit burst 15k cburst 15k

#filtros
$TC filter add dev $DEV protocol ip parent 1: prio 1 u32 match ip sport 4569
0xffff flowid 1:10
$TC filter add dev $DEV protocol ip parent 1: prio 1 u32 match ip sport 80
0xffff flowid 1:20
```

5. CÓDIGO DE IMPLEMENTACIÓN HTB MODIFICADO

```
#!/bin/bash

GREP=/bin/grep
INSMOD=/sbin/insmod
TC=/usr/sbin/tc
DEV=eth1
trg=1000kbps

RATE=$(($trg * 60) / 100))
porcentajeA=$($RATE * 7))
porcentajeB=$($RATE * 2))
porcentajeC=$($RATE * 1))

# Cargar modulos delkernel
$GREP -q ^sch_htb /proc/modules || $INSMOD /lib/modules/`uname -
r`/sch_htb.o
$GREP -q ^sch_sfq /proc/modules || $INSMOD /lib/modules/`uname -
r`/sch_sfq.o
$GREP -q ^cls_u32 /proc/modules || $INSMOD /lib/modules/`uname -
r`/cls_u32.o

#clase raiz HTB

$TC qdisc add dev $DEV root handle 1: htb default 10
$TC class add dev $DEV parent 1: classid 1:1 htb rate $100kbit burst 20k
cburst 20k

#clases HTB

$TC class add dev $DEV parent 1:1 classid 1:10 htb rate $70kbit ceil
$100kbit burst 15k cburst 15k
$TC class add dev $DEV parent 1:1 classid 1:20 htb rate $20kbit ceil
$100kbit burst 15k cburst 15k
$TC class add dev $DEV parent 1:1 classid 1:30 htb rate $10kbit ceil
$100kbit burst 15k cburst 15k

#filtros
$TC filter add dev $DEV protocol ip parent 1: prio 1 u32 match ip sport
4569 0xffff flowid 1:10
$TC filter add dev $DEV protocol ip parent 1: prio 1 u32 match ip sport 80
0xffff flowid 1:20
```