MODELO PARA LA OPTIMIZACIÓN DE LAS CAPAS DE TRANSPORTE EN LA ARQUITECTURA WAP 2.0

Trabajo de grado presentado como requisito para optar el título de Ingeniero en Electrónica y Telecomunicaciones

Guillermo Esteban Guerrero Rosero
Alejandro Cardozo Montes

Director: Ing. Javier Hurtado

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Popayán, 2005

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO I

1. INTRODUCCION

Las tecnologías inalámbricas han tenido mucho auge y desarrollo en estos últimos años. Una de las que ha tenido un gran desarrollo ha sido la telefonía celular, desde sus inicios a finales de los 70 ha revolucionado enormemente las actividades que realizamos diariamente. Los teléfonos celulares se han convertido en una herramienta primordial para las personas, a pesar de que la telefonía celular fue concebida estrictamente para la voz, la tecnología celular de hoy es capaz de brindar otro tipo de servicios, como datos, audio y video con algunas limitaciones, sin embargo, la telefonía inalámbrica del mañana hará posible aplicaciones que requieran un mayor consumo de ancho de banda.

La investigación de tecnologías de transporte de datos mas eficientes, para el transporte de datos a mayor velocidad, como es el caso de las tecnologías de transporte de segunda, tercera y cuarta generación (2G, 3G y 4G) han hecho posible mayor interactividad entre los usuarios de dispositivos móviles, dejando atrás la idea de utilizarlos únicamente para la comunicación de voz. Servicios de chat, correo electrónico, mensajes multimedia, y transmisión de datos desde y hacia Internet, hacen que las formas de comunicación sean variadas y personalizadas. Al explotar las capacidades de las tecnologías de transporte, se pueden prestar mejores servicios haciendo que los dispositivos móviles se conviertan en pequeños terminales computacionales al alcance de cualquier persona.

Para el soporte de todas estas características adicionales dentro de un dispositivo móvil, se han creado plataformas para el desarrollo de aplicaciones tanto multimedia como de la transferencia de archivos. WAP (Wireless Application Protocol) es una especificación para el desarrollo de aplicaciones, la cual permite crear, desplegar y transportar datos entre un dispositivo móvil y otro terminal ubicado en una red cableada, inalámbrica u otro terminal móvil.

En la actualidad, las compañías prestadoras de servicios de comunicaciones para dispositivos móviles en Colombia, han adoptado las recomendaciones emanadas por el WAP Forum, que es un grupo perteneciente a OMA (Open Mobile Alliance) [1] principal foro para el desarrollo y manejo del mercado de prestadores de servicios para las comunicaciones móviles,

recomendaciones que dan pautas para implementar servicios personalizados en dispositivos móviles.

WAP fue generado como un conjunto de especificaciones para el desarrollo de aplicaciones móviles con capacidades de interacción sobre redes de comunicaciones inalámbricas. Estas comunicaciones están basadas en el transporte de datos desde el dispositivo móvil hasta una red cableada, otro dispositivo móvil o una red inalámbrica, pero con la característica que todo el trafico de datos que envía y recibe el dispositivo móvil tiene que pasar a través de una gateway WAP, la cual tiene la tarea de convertir los protocolos utilizados en la comunicación inalámbrica con el móvil (enlace desde el móvil hasta la gateway) y la comunicación cableada con la red (cableada o inalámbrica) o equipo destino.

Debido a que WAP fue basado en la arquitectura de comunicaciones de Internet, ha heredado muchas de sus características, como es el caso del comportamiento de sus protocolos, los cuales fueron diseñados originalmente teniendo en cuenta los servicios prestados y el medio de transmisión utilizado hace ya algunos años.

El medio de transmisión al cual está ligada una comunicación en Internet tiene características completamente diferentes a las de una red inalámbrica. Condiciones como el ancho de banda, la tasa de bits erróneos (BER) y el tiempo de permanencia estable de la comunicación son características esenciales en una comunicación inalámbrica y por lo tanto, exigen un tratamiento diferente al que se da en redes cableadas, donde se utilizan modelos diferentes.

Para la implementación de servicios bajo la arquitectura WAP, se plantea un conjunto de protocolos organizados teniendo en cuenta el modelo de referencia OSI, en el cual las capas inferiores prestan servicios a las capas de nivel superior.

La arquitectura del conjunto de protocolos WAP se asemeja al modelo empleado por la mayoría de aplicaciones implementadas en Internet, en donde éstas siguen un modelo creado por ISO (Internacional Standard Organization) llamado el Modelo OSI (Open System Interconnect Model), el cual describe capas bien definidas para un sistema que opera dentro de una red. El propósito de las capas es proveer funciones claramente definidas que puedan mejorar la conectividad entre terminales con características heterogéneas. Las capas de nivel inferior prestan servicios a las capas de nivel superior.

En la figura 1.1 se muestra el modelo de referencia OSI.

Figura 1. 1. Modelo de Referencia OSI

La utilización del modelo de referencia OSI, como base de las aplicaciones para dispositivos móviles en la arquitectura WAP, permite que los desarrollos bajo esta plataforma tengan características como:

- Estandarización de tecnologías.
- Facilidad para el desarrollo de nuevas aplicaciones.
- Alto nivel de modularidad en cada capa.
- Aplicaciones escalables.

Las aplicaciones que trabajan con transferencia de datos, tienen un tipo de arquitectura que en su mayoría es estandarizado, ya que la globalización de las tecnologías y aplicaciones hace que sus modelos de arquitectura se asemejen a los que se emplean en Internet. El modelo de muchas aplicaciones que están conectadas y trabajan con Internet es el siguiente.

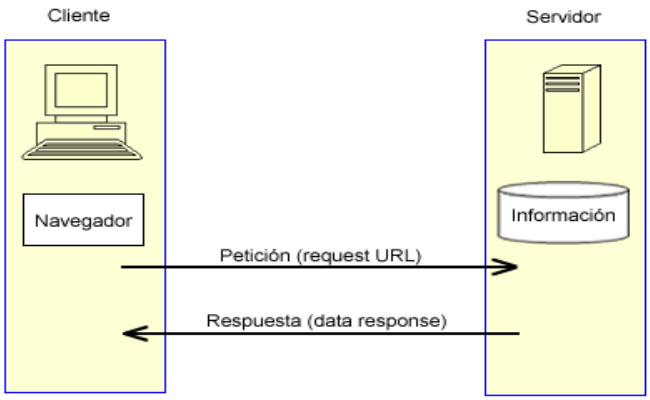


Figura 1. 2 Modelo de Aplicaciones Internet

Este modelo permite la interacción de dos terminales con diferentes características en un ambiente cableado (medio de transmisión rígido, fibra óptica, cable coaxial, par trenzado, etc.). La interacción entre capas se hace a partir de protocolos establecidos como estándar, como es el caso del Protocolo de Control de Transporte (TCP, RFC0793), y el Protocolo de Internet (IP, RFC0791).

La estructura de la especificación WAP, al igual que el modelo para aplicaciones de interconexión en Internet, esta compuesta por una serie de capas, las cuales están definidas por diferentes recomendaciones estandarizadas.

En la figura 1.3 se muestra la estructura en capas del modelo WAP.

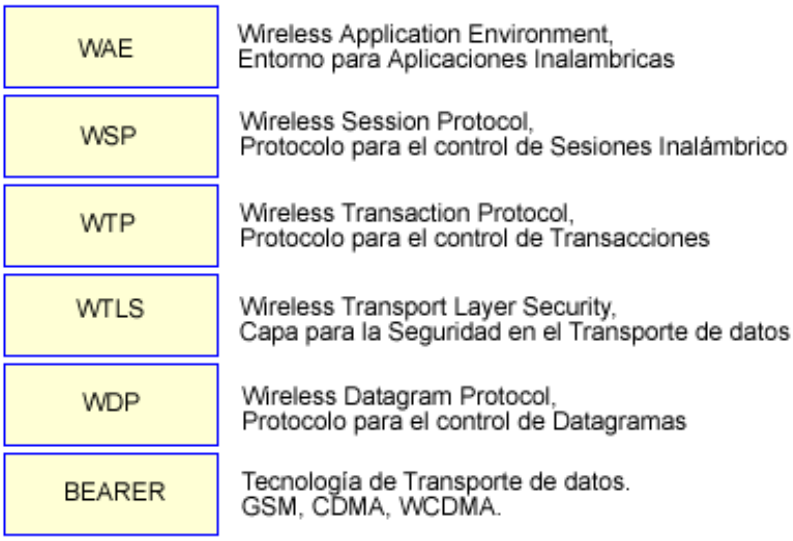


Figura 1. 3. Capas de la modelo WAP

Estas capas cumplen diferentes tareas de acuerdo a la aplicación que el usuario o el mismo dispositivo activen.

Para la interconexión entre una aplicación en un dispositivo móvil y otra aplicación en un Terminal sobre una red cableada, inalámbrica u otro dispositivo móvil, se implementan tecnologías en las capas WTP y WTLS. Estas capas pueden ser implementadas con diferentes recomendaciones como se muestra a continuación:

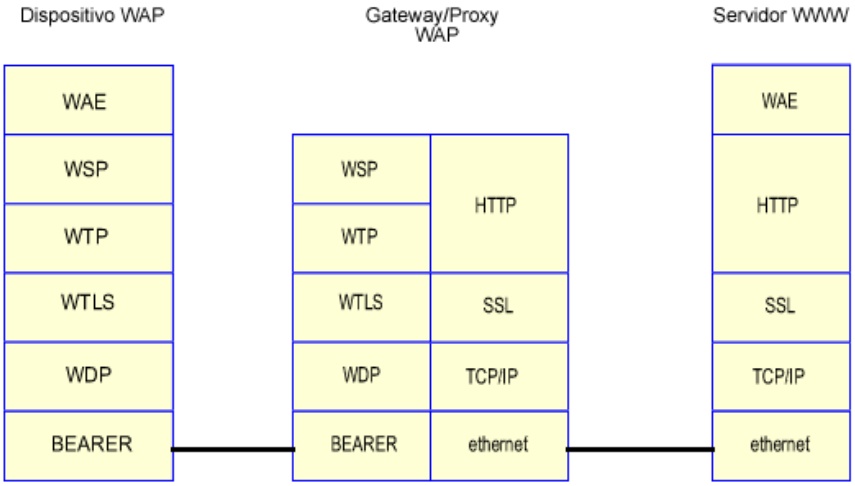


Figura 1. 4. Interconexión entre aplicaciones y su relación con la arquitectura WAP

Para la implementación en la capa mas alta de la arquitectura WAP se tiene un conjunto de herramientas, las cuales han sido heredadas de las aplicaciones, herramientas y tecnologías utilizados en Internet, en este caso tenemos por ejemplo, la posibilidad de utilizar tecnologías como XML, URL, *Scripting* y soporte para diversos tipos de datos multimedia.

De igual forma como se utiliza tecnologías para el desarrollo de servicios y/o aplicaciones para Internet se pueden utilizar similares tecnologías para el desarrollo en las capas por debajo de la capa de aplicación (WAE), un ejemplo de este es el soporte que se tiene para protocolos como HTTP para el control de sesiones en servicios o aplicaciones.

Para el caso de la implementación de políticas de seguridad en aplicaciones o servicios, se pueden utilizar soluciones como las utilizadas en Internet, como el Protocolo de Control de la Transmisión (TCP). La inclusión de TCP ha sido motivada por la necesidad de implementar servicios de alta velocidad en redes inalámbricas (2.5G y 3G). Algunos de los beneficios que provee TCP incluye, la transferencia de datos de gran tamaño, seguridad terminal a terminal (utilizando TLS) y la convergencia con los protocolos existentes en Internet (protocolos del IETF, RFC0793 y RFC1122).

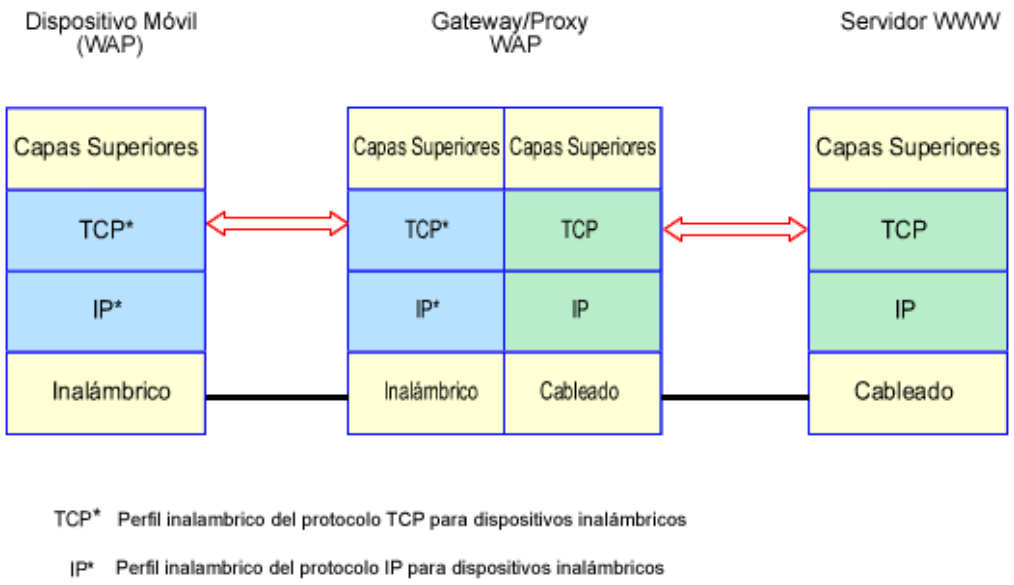


Figura 1. 5. Conexión fraccionada de la arquitectura WAP

En la figura 1.5 se representa una de las posibilidades para la configuración de las capas de la arquitectura WAP; en esta se muestra las posibles tecnologías empleadas en las capas que intervienen en la transmisión de los datos en el modelo WAP.

Para la implementación de las capas de el modelo WAP se han utilizado variaciones de los protocolos actualmente utilizados en Internet, como son TCP, IP y HTTP, lo cual garantiza una mejor eficiencia en la traducción de los datos en la Gateway/Proxy hacia Internet.
El propósito de esta arquitectura es el de poder facilitar a la Gateway/Proxy la conversión de los datos de un medio a otro.

Aunque la arquitectura anterior no es una implementación estricta, es ampliamente utilizada debido a que los protocolos que interactúan en esta arquitectura son conocidos y aceptados mundialmente, aunque tiene inconvenientes los cuales tienen que ver en su mayoría con el desempaquetamiento de la información en la Gateway/Proxy, esto en definitiva es una falla de seguridad, ya que se pierde la concepción de seguridad punto a punto (Terminal a terminal) que es fundamental cuando se envía información privada del usuario, como es contraseñas, números de cuentas e información privada del usuario.

Este tipo de arquitectura se planteó para brindar al usuario funciones personalizadas como es el envió de mensajería *push*, además de tener ventajas como la de la compresión y encriptación de datos, lo cual impacta en gran medida en la velocidad de la transmisión.

La utilización de protocolos en la arquitectura WAP, se hizo a partir del análisis de las redes de transmisión de datos que existen en la actualidad, y en las cuales predomina la utilización de protocolos como TCP e IP para el control y soporte de intercambio de datos entre terminales.

El WAP Forum, que es la entidad que genera las recomendaciones para las implementaciones de WAP a nivel mundial, ha generado recomendaciones que definen una Capa de Servicios de Transporte (Transport Services Layer TSL) que provee servicios orientados a la conexión y orientados a datagramas para las capas de protocolos que se encuentran mas arriba de la arquitectura; el perfil inalámbrico de TCP (Wireless Profiled TCP, WP-TCP) provee servicios orientados a la conexión. La inclusión de TCP ha sido motivada por las características que presta TCP en redes cableadas como es, la transferencia de datos de gran tamaño, seguridad terminal a terminal (utilizando TLS) y principalmente debido a la convergencia con los protocolos de la IETF. La especificación del WP-TCP es independiente de la versión IP soportada por las tecnologías de transporte que están por debajo de la capa de control de transporte. La recomendación del WAP Forum especifica las características que debería tener el perfil del protocolo TCP, como se muestra en la tabla 1.1 la cual fue tomada de la recomendación emanada del WAP Forum, **Wireless Profiled TCP,** Version 31-March-2001, WAP-225-TCP-20010331-a.

Items	Qualifier	Support level
Large window size based on BDP		SHOULD
Window Scale Option [RFC1323]	Window size >= 64KB	MUST
	Window size < 64KB	MAY
Timestamps Option [RFC1323] for RTTM	Window size >= 64KB	SHOULD
	Window size < 64KB	MAY
Large Initial Window (cwnd<=2) [RFC2581]		MUST
Large Initial Window (cwnd>2) [RFC2414]		MAY
Selective Acknowledgement Option (SACK) [RFC2018]		MUST
Path MTU Discovery [RFC1191, RFC1981]		SHOULD
MTU larger than default IP MTU	Path MTU Discovery NOT Supported	MAY
Explicit Congestion Notification (ECN) [RFC2481]		MAY

Tabla 1. 1 Características del WP-TCP WAP Forum.

La implementación de las recomendaciones mencionadas en la tabla anterior permiten una buena optimización del protocolo TCP, pero existen mecanismos de mejoramiento del perfil TCP que no han sido abordados en la recomendación generada por el WAP Forum, tales como la optimización de los paquetes SACK, la utilización de un método dinámico y mas confiable para la determinación de la ventana en caso de perdida de paquetes debido a las características del medio de transporte y no por congestión, además de los métodos de compresión de las cabeceras de los paquetes de control (banderas) como de los que llevan la información.

Los inconvenientes en el *perfil TCP* se presentan debido a su origen basado en las características del medio de transmisión cableado para el cual fue creado, el principal objetivo de este proyecto es mejorar el rendimiento del protocolo de transmisión en un ambiente inalámbrico, con la utilización de las funciones de optimización recomendadas por el WAP Forum y otras aun experimentales, sin perder las características que hacen de TCP uno de los mas utilizados protocolos en redes cableadas, tal optimización permite utilizar una arquitectura de los protocolos de WAP que sea más eficiente en las capas de los protocolos de transporte WTP (Wireless Transaction Protocol) y WDP (Wireless Datagram Protocol).

La conceptualización general sobre trabajos anteriores y sobre las diferentes temáticas que han sido el soporte para la implementación del proyecto se plantean en el capitulo II **MECANISMOS ALTERNATIVOS DE OPTIMIZACION DEL PERFIL TCP.**

En el Capitulo III de este proyecto de investigación, **PARAMETROS Y METRICAS PARA LA EVALUACION EXPERIMENTAL.** Se hace una descripción de las características del ambiente en el cual se desarrolló la investigación, los parámetros iniciales para los experimentos y las métricas para evaluar las optimizaciones del protocolo.

Debido a que la implementación de un protocolo de transporte esta íntimamente ligado al soporte software de un terminal (Sistema Operativo), no se puede especificar un código universal para la implementación de este tipo de soluciones, por lo tanto esta investigación finalizará con el planteamiento de los mecanismos de optimización que implementados dentro del dispositivo móvil y en la Gateway/Proxy WAP mejoraran el rendimiento de las comunicación de datos.

En el capitulo IV **EVALUACION DE LOS MECANISMOS DE OPTIMIZACION** se muestra en forma comparativa la evaluación de cada uno de los mecanismos para la optimización del protocolo TCP.

Las diferentes conclusiones que surgen al finalizar el proyecto además de los futuros trabajos que pueden ser soportados por el sistema desarrollado se muestran en el capitulo V **CONCLUSIONES Y TRABAJOS FUTUROS**.

CAPITULO II

2. MECANISMOS ALTERNATIVOS DE OPTIMIZACION DEL PERFIL TCP

2.1 Entorno para el estudio de los mecanismos de optimización del protocolo TCP inalámbrico.

El estudio de los mecanismos para la optimización del protocolo TCP para el perfil inalámbrico del conjunto de protocolos WAP 2.0, ha sido basado en los requisitos que necesitan algunas de las aplicaciones más utilizadas en los dispositivos móviles. Esta clasificación de las aplicaciones ha sido planteada en muchas investigaciones, en Towards Taxonomy For Mobile Applications With Adaptive Behavior. 2003 [ACBR 03] se propone un estudio taxonómico de las aplicaciones, basado en los elementos que pueden ser sujetos a la adaptación como son, los recursos (de la red y del dispositivo), de localización y el contexto.

Los recursos incluyen elementos computacionales tanto físicos como lógicos, relacionados con la red, el dispositivo móvil, los elementos estáticos, archivos, programas y otros. La localización es la posición física del usuario con su dispositivo móvil. El contexto es un concepto amplio y para algunos proyectos Towards Taxonomy For Mobile Applications With Adaptive Behavior. 2003 [ACBR 03] es definido como toda la información, relevante a la aplicación, que puede ser utilizada para definir su comportamiento. El contexto es determinado por la información de quién, cuándo, dónde, qué esta haciendo y con que recursos los esta haciendo (referido a la aplicación). Esta información puede ser obtenida en la parte móvil y en la parte estática del ambiente a través de algún modulo de monitoreo de la arquitectura del software.

En la figura 2.1 se muestra los elementos computacionales que son importantes para determinar el comportamiento de una aplicación. Este comportamiento esta representado por su grado de incorporación (por lo tanto de complejidad).

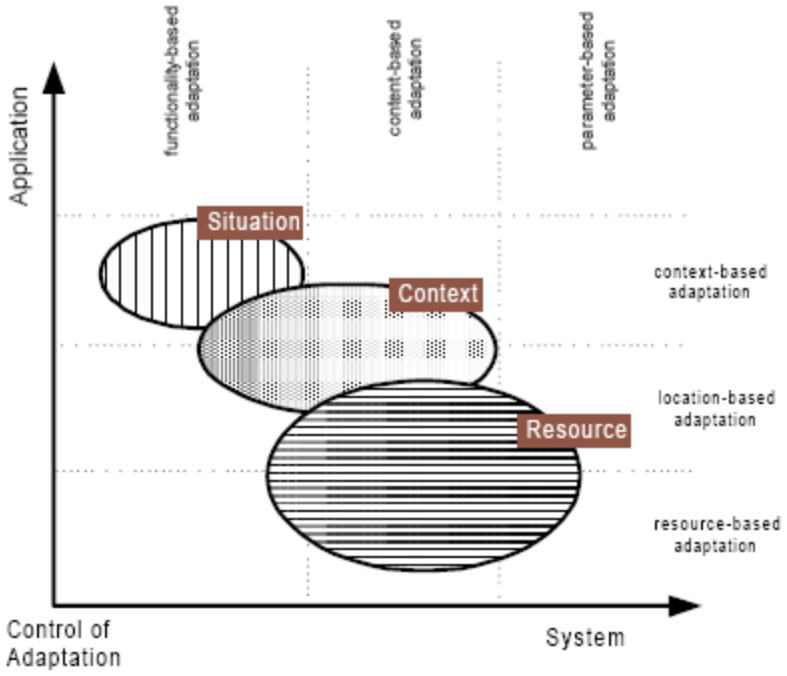


Figura 2. 1. Taxonomía de las aplicaciones móviles [ACBR 03].

En la figura 2.1 las elipses representan el tipo de aplicación, las divisiones horizontales representan la sensibilidad a los elementos computacionales, mientras las divisiones verticales representan las estrategias de adaptación.

La combinación de estos tres aspectos nos muestran las características de muchos dominios de las aplicaciones móviles, y sobre todo la importancia que tiene los recursos, no solo los recursos en cuanto a equipos sino también los recursos de las redes inalámbricas. Este último punto es el objetivo del proyecto de investigación presente, en el cual se quiere aumentar la eficiencia de los recursos de red en la interfaz aérea de una conexión inalámbrica, teniendo en cuenta que al hablar de *conexión inalámbrica* se entiende por una conexión de datos terminal a terminal en la cual se utiliza el protocolo de Control de Transporte (perfil TCP) para la transmisión confiable de los datos. La estructura de conexión entre los terminales de una *conexión inalámbrica* se muestra en la figura 2.2.

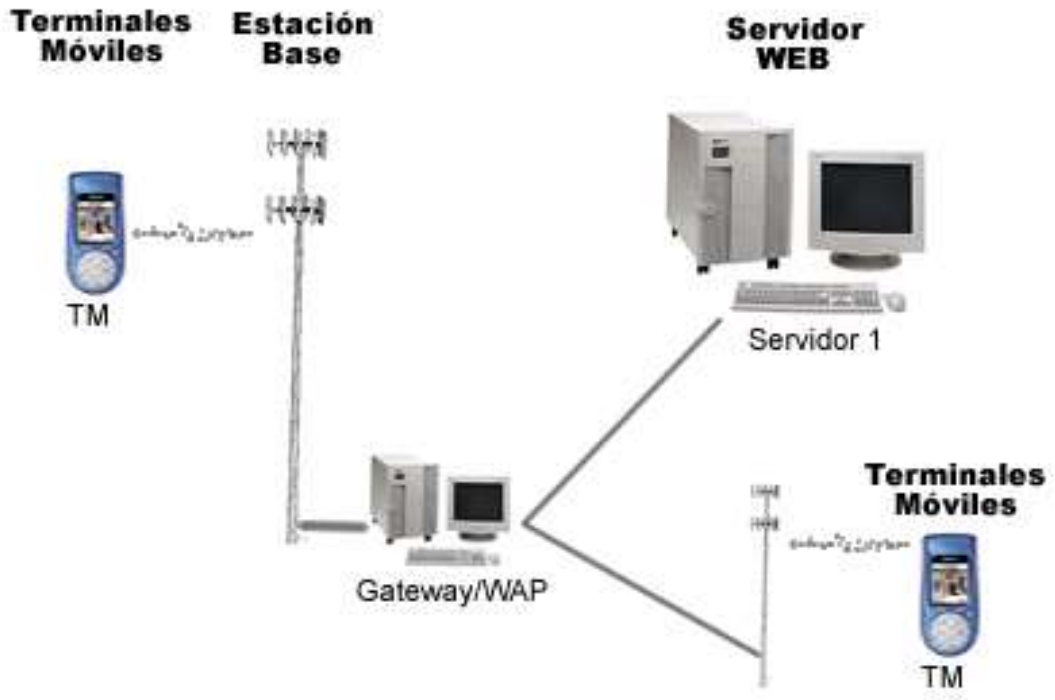


Figura 2. 2. Estructura de conexión entre los terminales de una *conexión inalámbrica.*

2.1.1 Proceso de encapsulado de datos del protocolo TCI e IP.

Las capas del modelo OSI se comunican entre sí utilizando las **PDU** (protocol data unit), que especifican que información debe agregarse como encabezado o final de los datos que ingresan a la capa. Analizamos el paso de los datos por las 4 últimas capas del modelo (*transporte, red, enlace de datos, física*).

Cuando los datos bajan de la capa sesión, la **PDU** de la capa de transporte exige el agregado del encabezado de protocolo **TCP**. La capa siguiente agrega el encabezado **IP**. Al bajar a la capa de Enlace, el encabezado que se agrega depende de la implementación de **Ethernet** que se esté utilizando. Si la implementación es **ETHERNET II**, se agrega solamente un encabezado **MAC**, si la implementación es **IEEE 802.3 802.2**, se agregan 2 encabezados: **LLC** de la subcapa superior (*Logical Link Control*) y **MAC** (*Media Access Control*) de la subcapa inferior, para luego pasar a la capa Física convertido en señales eléctricas.

En figura 2.3 vemos el proceso descrito, aplicado al caso del estándar **IEEE 802.3**.

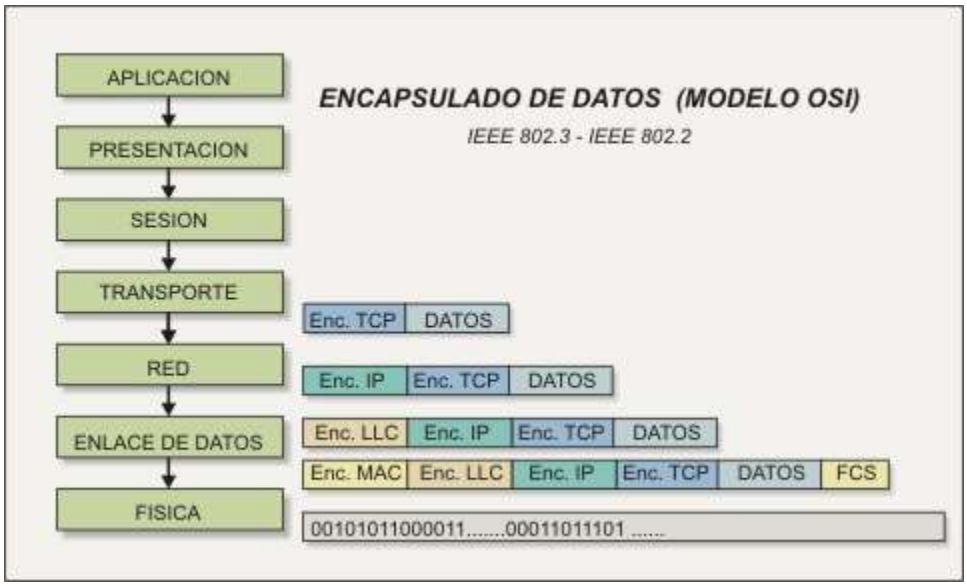


Figura 2. 3. Encapsulado de Datos

En figura 2.4 se detallan las tramas **ETHERNET**, notando las diferencias entre los 3 formatos posibles:

ETHERNET II: El cuarto campo contiene el tipo de protocolo que viaja en la parte de datos de la trama (*0x800* para **IP**).

IEEE 802.3 + 802.2 SAP (*Service Access Point*): el cuarto campo contiene el largo de la parte de datos más el encabezado **LLC** (no debe ser mayor de 2048 para mantener la compatibilidad con **ethernet II**). El encabezado **LLC** especifica el **DSAP** y **SSAP** (*Destination Service Access Point y Source Service Access Point*) que son utilizados por algunos protocolos para implementar la funcionalidad de la capa de transporte, y por último un byte de control.

IEEE 802.3 + 802.2 SNAP (Sub-Network Access Protocol): El cuarto campo es el mismo que en el caso anterior, pero el encabezado **LLC** cambia fijando los contenidos **DSAP (170), SSAP (170) y Control** y agregando 2 campos: **OUI ID** (Organization Unique Identifier ID seteado en 0 por defecto) y Tipo de protocolo (**IP 0x800, ARP 0x806 ó RARP 0x835**).

En todos los casos se agrega un campo **FCS** al final (*Frame Check Sequence*) para verificación (**CRC**).

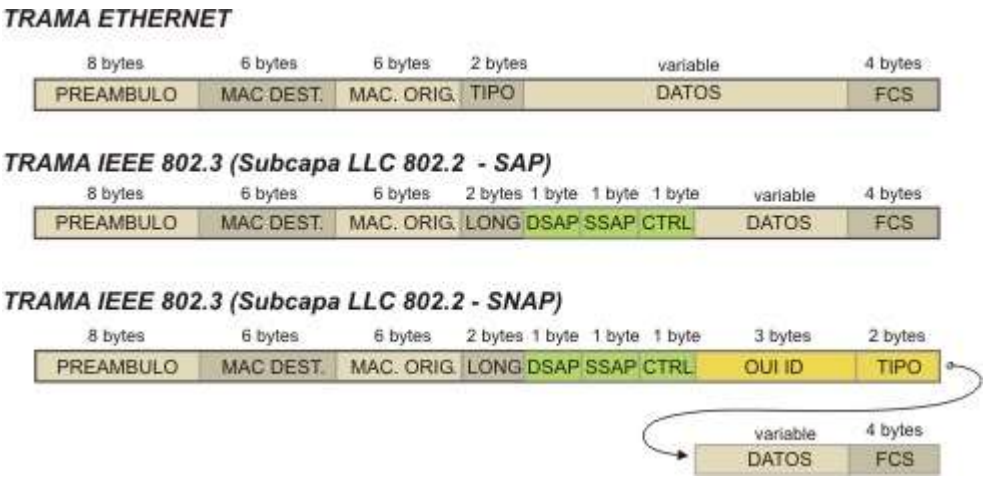


Figura 2. 4. Tramas de datos.

2.1.1.1 Encabezado IP

Ahora veremos de qué manera está compuesto un encabezado **IP**. En la figura 2.5 se representa el encabezado, separando los campos cada 32 bits.

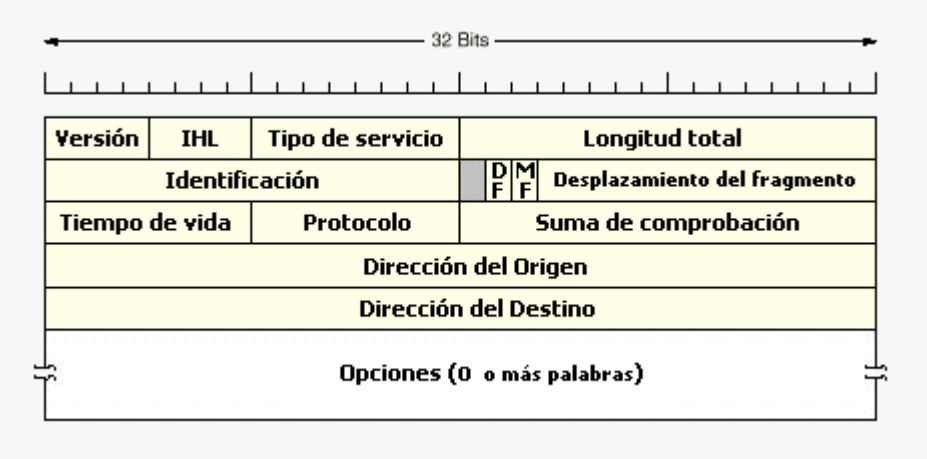


Figura 2. 5. Encabezado IP

A continuación se describen cada uno de los campos que forman la cabecera:

Versión (4 bits): Indica el número de versión del protocolo al que pertenece el datagrama, lo que permitirá la evolución futura del protocolo y que la transición entre las versiones se pueda hacer ejecutándose en unas máquinas la versión vieja y en otras la versión nueva.

IHL (Internet Header length) (4 bits): Indica la longitud de la cabecera en palabras de 32 bits (4 bytes). El valor mínimo es cinco (20/4=5). Este campo es necesario por no ser constante el tamaño de la cabecera como hemos comentado anteriormente. El valor máximo puede ser 15 (1111) lo que limita la cabecera a 60 bytes (15*4) y en consecuencia el campo de opciones a 40 (60-20). En el caso de que, por ejemplo, se quiera registrar la ruta de un paquete este valor puede ser insuficiente y ser totalmente inútil esta opción.

Tipo de servicio (8 bits): Permite que el host especifique que clase de servicio quiere, pudiéndose combinar confiabilidad y velocidad. Para la voz digitalizada es mas importante realizar la entrega de forma rápida que precisa, mientras que para la transferencia de ficheros no importa a que velocidad se realiza la transferencia pero si que esté libre de errores. De los 8 bits, 3 son para el campo de precedencia que en realidad es una prioridad de 0 (normal) a 7 (para los paquetes de control de red). A continuación aparecen los bits de seguridad (alta o baja), retardo (alto o bajo cuando se intenta minimizar el retardo) y rendimiento (normal o alto cuando se intenta maximizar el rendimiento durante la transmisión del datagrama).

Longitud total (16 bits) en bytes que tendrá todo el datagrama, considerando tanto la cabecera como los datos. Hay que tener en cuenta que el tamaño máximo de un datagrama es de 65535 bytes lo que puede ser insuficiente en las redes de alta velocidad.

Identificador (16 bits): es un número de secuencia que junto a la dirección origen, la dirección destino y el protocolo de usuario, sirven para que la máquina destino determine a que datagrama pertenece el fragmento que ha recibido. Todos los fragmentos de un datagrama contienen el mismo valor en el campo identificador y este número debe ser único para la dirección origen, la dirección destino y el protocolo de usuario durante el tiempo en el que el datagrama permanece en el conjunto de redes.

Indicadores (3 bits): El primer bit no se utiliza actualmente. El indicador de mas fragmentos (**MF**) cuando vale 1 indica que este datagrama tiene mas fragmentos y toma el valor 0 en el último fragmento. El indicador de no fragmentar (**DF**) prohíbe la fragmentación cuando vale 1. Es una orden que se le da a los encaminadores de que no fragmenten el datagrama cuando el destino es incapaz de reensamblarlo. Si este bit vale 1, el datagrama se descartará si se excede el tamaño máximo en una subred de la ruta. Por lo tanto, cuando este bit vale 1, es aconsejable

usar encaminamiento por la fuente para evitar subredes cuyo tamaño máximo de paquete sea menor que el tamaño del datagrama.

Desplazamiento del fragmento (13 bits): Indica en que posición del datagrama original, medido en unidades de 8 bytes (64 bits), va el fragmento actual. Debido a esto, todos los fragmentos excepto el último contienen un campo de datos con una longitud múltiplo de 8 bytes. Como se proporcionan 13 bits, puede haber un máximo de 8912 (2**13) fragmentos por datagrama, y por lo tanto el tamaño máximo de un datagrama es de 65536 bytes, uno mas que el campo de longitud total.

Tiempo de vida (8 bits): Es un contador que sirve para limitar la vida de un paquete. Aunque lo lógico sería pensar que cuenta el tiempo en segundos, en realidad lo que cuenta es el número de saltos de dispositivo de encaminamiento que realiza. Cuando el contador llega a cero, el paquete se descarta y se envía un paquete al computador origen avisándole. Con este mecanismo se consigue que los datagramas no permanezcan indefinidamente en la red si, por ejemplo, se dañan las tablas de encaminamiento.

Protocolo (8 bits): Se utiliza por la capa de red para saber a que protocolo de la capa de transporte le tiene que enviar el datagrama una vez lo ha reensamblado. Existen diferentes protocolos de transporte, entre ellos TCP y UDP. En el RFC 1700 se definen todos estos protocolos.

Suma de comprobación (16 bits): Sirve para verificar el contenido de la cabecera y es útil para la detección de errores generados durante la transmisión del datagrama. Como algunos de los campos de la cabecera pueden cambiar en alguno de los dispositivos de encaminamiento (por ejemplo, el tiempo de vida y algunos campos relacionados con la segmentación), este valor es verificado y recalculado en cada uno de los dispositivos de encaminamiento. El algoritmo empleado consiste en sumar todas las medias palabras de 16 bits a medida que van llegando, usando la aritmética de complemento a 1, y luego obtener el complemento a 1 del resultado. Se supone que la suma de comprobación de la cabecera es cero cuando llega. Este algoritmo es algo más robusto que una suma normal. Existen algunas técnicas para acelerar el cálculo.

Dirección origen (32 bits): Indica el número de red y el número del ordenador que envía el datagrama.

Dirección destino (32 bits): Indica el número de red y el número del ordenador al que se envía el datagrama.

Opciones (variable): Contiene las opciones solicitadas por el usuario que envía los datos y se diseñó para que las versiones posteriores del protocolo pudieran incluir información no considerada originalmente, para que los investigadores pudieran probar cosas nuevas y para que aquella información que es utilizada pocas veces no tuviera asignada unos bits determinados en la cabecera. Cada una de las opciones empieza en 1 byte que identifica la opción. Algunas de las opciones vienes seguidas de un campo de 1 byte para indicar la longitud de la opción y a continuación uno o más bytes de datos. Hay seis opciones (Seguridad, Encaminamiento estricto desde el origen, Encaminamiento libre desde el origen, Registrar la ruta, Identificación de secuencia, Marca de tiempo definidas actualmente pero no todas son reconocidas por todos los dispositivos de encaminamiento:

Seguridad: Permite añadir una etiqueta para indicar lo secreta que es la información que contiene el datagrama. Por ejemplo, se podría utilizar para que los dispositivos de encaminamiento no consideren redes en concreto. Pero en realidad esta etiqueta es ignorada y realmente para lo único que sirve es para ayudar a los espías a encontrar con mayor facilidad la información importante.

Encaminamiento estricto desde el origen: Es una secuencia de direcciones IP que sirve para indicar la trayectoria completa que debe seguir el datagrama desde el origen hasta el destino. Esta opción es usada sobre todo cuando los administradores de sistemas envían paquetes de emergencia porque las tablas de encaminamiento se han corrompido o para hacer mediciones de tiempo.

Encaminamiento libre desde el origen: Es una secuencia de direcciones IP que sirve para indicar que el datagrama debe pasar obligatoriamente por esos dispositivos de encaminamiento y en ese orden, pero también puede pasar por otros dispositivos de encaminamiento. Esta opción es útil cuando por diversas consideraciones se deben pasar por algunos dispositivos de encaminamiento en concreto.

Registrar la ruta: Sirve para indicar que los dispositivos de encaminamiento agreguen su dirección IP al campo de opción y de esta manera tener conocimiento de la ruta seguida por el datagrama. Se utiliza, por ejemplo, para poder determinar si los algoritmos de encaminamiento están funcionando correctamente. Los 40 bytes de tamaño máximo que puede tener el campo de opciones sólo permiten registrar 9 saltos, lo que puede ser en las redes actuales en muchos casos insuficiente.

Identificación de secuencia. Se utiliza cuando hay recursos reservados para un servicio, por ejemplo voz.

***Marca de tiempo*:** En este caso, además de registrar las direcciones de los dispositivos de encaminamiento como se hacia en la opción registrar la ruta, se utilizan 32 bits para guardar una marca de tiempo expresada en milisegundos. Esta marca es usada principalmente para buscar fallos en los algoritmos de encaminamiento.

Relleno (variable): El campo de opciones se rellena para que su tamaño sea múltiplo de 32 bits (4 bytes).

2.1.1.2 Encabezado TCP

La unidad de datos de este protocolo recibe el nombre de segmento TCP. Como la cabecera debe implementar todos los mecanismos del protocolo su tamaño es bastante grande, como mínimo 20 bytes.

En la figura 2.6 se puede ver el formato de la cabecera TCP.

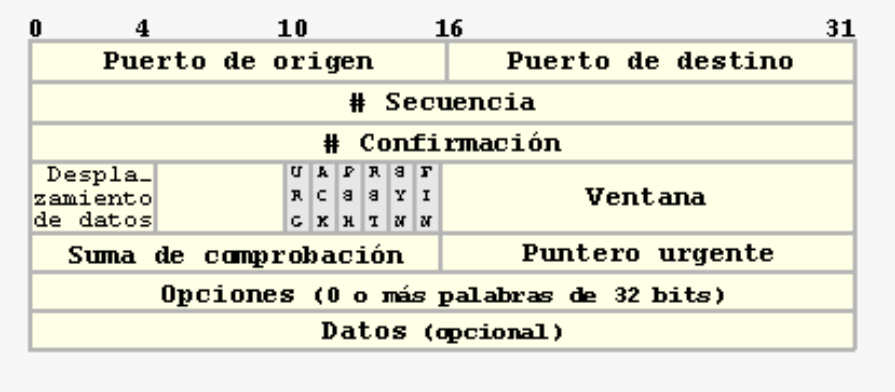


Figura 2. 6. Encabezado TCP

A continuación hay una descripción de cada uno de los campos que forman la cabecera:

Puerto origen (16 bits): Es el punto de acceso de la aplicación en el origen.

Puerto destino (16 bits): Es el punto de acceso de la aplicación en el destino.

Número de secuencia (32 bits): Identifica el primer byte del campo de datos. En este protocolo no se enumeran segmentos sino bytes, por lo que este número indica el primer byte de datos que hay en el segmento. Al principio de la conexión se asigna un número de secuencia inicial (ISN, Initial Sequence Number) y a continuación los bytes son numerados consecutivamente.

Número de confirmación (ACK) (32 bits): El protocolo TCP utiliza la técnica de *piggybacking* para reconocer los datos. Cuando el bit ACK está activo, este campo contiene el número de secuencia del primer byte que espera recibir. Dicho de otra manera, el número ACK - 1 indica el último bit reconocido.

Longitud de la cabecera (4 bits): Indica el número de palabras de 32 bits que hay en la cabecera. De esta manera el TCP puede saber donde se acaba la cabecera y por lo tanto donde empieza los datos. Normalmente el tamaño de la cabecera es de 20 bytes por lo que en este campo se almacenará el número 5. Si el TCP utiliza todos los campos de opciones la cabecera puede tener una longitud máxima de 60 bytes almacenándose en este campo el valor 15.

Reservado (6 bits): Se ha reservado para su uso futuro y se inicializa con ceros.

Indicadores o campo de control (6 bits): Cada uno de los bits recibe el nombre de indicador y cuando está a 1 indica una función específica del protocolo.

> **URG:** Hay datos urgentes y en el campo "puntero urgente" se indica el número de datos urgentes que hay en el segmento.
> **ACK:** Indica que tiene significado el número que hay almacenado en el campo "número de confirmación".
> **PSH:** Sirve para invocar la función de carga (push). Como se ha comentado anteriormente con esta función se indica al receptor que debe pasar a la aplicación todos los datos que tenga en la memoria intermedia sin esperar a que sean completados. De esta manera se consigue que los datos no esperen en la memoria receptora hasta completar un segmento de dimensión máxima. No se debe confundir con el indicador URG que sirve para señalar que la aplicación ha determinado una parte del segmento como urgente.
> **RST:** Sirve para hacer un reset de la conexión.

SYN: Sirve para sincronizar los números de secuencia.

FIN: Sirve para indicar que el emisor no tiene mas datos para enviar.

Ventana (16 bits): Indica cuantos bytes tiene la ventana de transmisión del protocolo de control de flujo utilizando el mecanismo de ventanas deslizantes. A diferencia de lo que ocurre en los protocolos del nivel de enlace, en los que la ventana era constante y contaba tramas, en el TCP la ventana es variable y cuenta bytes. Contiene el número de bytes de datos comenzando con el que se indica en el campo de confirmación y que el que envía está dispuesto a aceptar.

Suma de comprobación (16 bits): Este campo se utiliza para detectar errores mediante el complemento a uno de la suma en módulo 2^{16} - 1 de todas las palabras de 16 bits que hay en el segmento mas una seudo-cabecera. La seudo-cabecera incluye los siguientes campos de la cabecera IP: dirección Internet origen, dirección Internet destino, el protocolo y un campo longitud del segmento. Con la inclusión de esta seudo-cabecera, TCP se protege a si mismo de una transmisión errónea de IP. Esto es, si IP lleva un segmento a un computador erróneo, aunque el segmento esté libre de errores, el receptor detectará el error de transmisión.

Puntero urgente (16 bits): Cuando el indicador URG está activo, este campo indica cual es el último byte de datos que es urgente. De esta manera el receptor puede saber cuantos datos urgentes llegan. Este campo es utilizado por algunas aplicaciones como telnet, rlogin y FTP.

Opciones (variable): Si está presente permite añadir una única opción de entre las siguientes:

Timestamp para marcar en que momento se transmitió el segmento y de esta manera monitorizar los retardos que experimentan los segmentos desde el origen hasta el destino.
Aumentar el tamaño de la ventana.

Indicar el tamaño máximo del segmento que el origen puede enviar.
Como TCP ha sido diseñado para trabajar con IP, algunos parámetros de usuario se pasan a través de TCP a IP para su inclusión en la cabecera IP. Por ejemplo: prioridad (campo de 3 bits), retardo-normal/ retardo- bajo, rendimiento-normal/rendimiento-alto, seguridad-normal/seguridad-alta y protección (campo de 11 bits)

A continuación se presentan los mecanismos y parámetros para optimizar dentro del ambiente anteriormente propuesto.

2.2 Parámetros y mecanismos a optimizar del protocolo TCP

2.2.1 Ventana inicial de transmisión

El protocolo TCP inicia transmitiendo segmentos de tamaño igual al numero inicial de la ventana en la conexión en la red. El tamaño inicial de la ventana de uno de los segmentos es establecido por el estándar actual del control de congestión. El incremento del tamaño de la ventana inicial tiene la ventaja de ahorrar hasta tres RTTs del tiempo de conexión. Este incremento de la ventana inicial puede tener unas posibles desventajas para una conexión individual y es la del incremento en la probabilidad de una pérdida de congestión en el inicio de la conexión cuando el tamaño del *buffer* del enrutador es pequeño.

El aumento en el tamaño inicial de la ventana puede ser conveniente en un ambiente inalámbrico debido al gran tamaño de la RTT del enlace inalámbrico y la presencia de perdidas de error. Con la implementación de este mecanismo se espera que el rendimiento se incremente con el incremento de la ventana inicial, pero el mejoramiento solo afecta al inicio de las conexiones.

2.2.2 Ventana de recepción

La cantidad de datos que viajan por la red, es limitada en cualquier instante de tiempo de una conexión por el tamaño mínimo de la ventana de congestión y la ventana de recepción. El tamaño de la ventana de recepción es un parámetro de control estándar de TCP. El dar a conocer a la red de una ventana más pequeña por parte del receptor puede controlar el número de segmentos que el emisor puede transmitir. Si la ventana de recepción es limitada a un valor apropiado que refleje la capacidad disponible de la red, entonces las perdidas por congestión son prevenidas. El receptor tiene rara vez conocimiento de las propiedades de la red y de su estado actual. Sin embargo, cuando un terminal sabe que esta conectado al último salto del enlace inalámbrico, este podría limitar la ventana de recepción. El limitar la ventana de recepción también previene excesivo encolamiento (*queueing*) en la red (*overbuffering*), lo cual ocurre cuando el tamaño del *buffer* del router (o gateway) es mucho más grande que el que

requiere el enlace. Es interesante examinar cuando la ventana de recepción pequeña previene el efecto *start-up buffer overflow* (efecto de *overbuffering* al inicio de la conexión), si la recuperación de errores no se activa y el tamaño de la ventana de recepción es el apropiado para un tamaño dado del *buffer* del router.

2.2.3 Tamaño máximo del segmento (Maximum Segment Size – MSS)

El MSS afecta el rendimiento de TCP. La MTU (*Maximum Transfer Unit*) del trayecto de la red impone un límite máximo para el MSS; en ciertos casos la utilización de una MSS más pequeña es preferible. Por ejemplo, con una MSS de 1024 bytes, cada segmento ocuparía un enlace de 9600 bps. Esto es inaceptable para una aplicación interactiva, debido a la transferencia de paquetes de tamaño grande puede retrasar un pequeño paquete *telnet* por un tiempo mucho más grande que el perceptible por un humano.

Los enlaces que confíen en la recuperación de errores de TCP en un enlace terminal a terminal también requieren un pequeño MSS. Para un BER fijo, la probabilidad de la corrupción de segmentos incrementa con su tamaño. Por otro lado, el aumento del tamaño de la cabecera crece con un pequeño MSS, especialmente en la ausencia de compresión de las cabeceras TCP/IP. Un valor de MSS de 256 bytes para un enlace de 9600 bps es utilizado a menudo como estándar.

Es interesante examinar el efecto de un tamaño más grande de MSS en la congestión TCP y el control de errores. TCP aumenta la ventana de congestión en unidades de segmentos, independientemente del número de bytes con acuse de recibo. La utilización de un tamaño más grande de MSS permite a la conexión completar más rápido la fase de *Slow Start*. Por otro lado, las conexiones con un tamaño más grande de MSS pueden sufrir más de RTOs (Retransmit TimeOuts). Un tamaño de segmento más grande provoca un más grande tamaño del RTT y además un número de paquetes transmitidos sobre la red aumenta más lentamente que para un tamaño más pequeño de MSS. Hay una más pequeña probabilidad de tener suficientes DUPACKs para activar una retransmisión rápida. Además, un más grande RTO, incrementa el tiempo de recuperación.

2.2.4 Acuses de Recibo Selectivos (*Selective Acknowledgments- SACK*)

Los acuses de recibo selectivos (**Selective Acknowledgments - SACKs**) son ACKs acumulativos; un ACK confirma la recepción de todos los datos hasta un paquete de bytes dado, pero no provee información alguna de los bytes siguientes al cual se ha acusado de recibo; por ejemplo si una transmisión envía 100 paquetes y llegaron al receptor los paquetes del 1 al 25 y del 50 al 75, entonces al emisor le llegará confirmación (acuse de recibo) de llegada hasta el paquete 25, haciendo que el emisor envié de nuevo los paquetes a partir del 26 desperdiciando recursos, con SACK el emisor tiene conocimiento de todos los paquetes que han llegado. De cómo el emisor utiliza la información que se le envía es completamente dependiente de la implementación. Otra implementación que utiliza este algoritmo es la llamada "*Reno+SACK*". SACK no cambia la semántica de los acuses de recibo acumulativos. Solo después de la llegada de un ACK acumulado, los datos son realmente confirmados y pueden ser descartados desde el *buffer* del emisor. Al receptor se le permite descartar datos con acuse de recibo utilizando SACK pero no datos con acuse de recibo normales.

El algoritmo FACK utiliza la información adicional dada por la opción SACK para mantener una medida explicita del número total de bytes de los datos transmitidos a través de la red. Al contrario, TCP *Reno* y TCP *Reno+SACK* intentan estimar el número de segmentos en la red, asumiendo que cada ACK duplicado recibido representa un segmento el cual ha dejado la red. En otras palabras, FACK asume que los segmentos en los "huecos" de la lista SACK, están perdidos y por lo tanto han dejado la red. Esto permite que FACK sea mas agresivo que *Reno+SACK* en la recuperación de datos perdidos. En particular, la Retransmisión Rápida (*Fast Retransmit*) puede ser activada después de un solo DUPACK (*DUPlicated ACK*) en las implementaciones FACK si la información SACK en el DUPACK indica que muchos segmentos se perdieron. Al contrario, TCP *Reno+SACK* esperara que lleguen tres DUPACKS para activar la Retransmisión Rápida.

2.2.5 Manejo Activo De Colas (*Active Queue Management*) En El Control De Congestión

El control de congestión para redes IP ha sido un problema recurrente durante muchos años. La esencia del esquema del control de congestión es que un emisor TCP ajusta su tasa de envió de acuerdo a la tasa (probabilidad) de que los paquetes sean rechazados (*dropped*) dentro de la red (lo cual se considera una medida de la congestión en la red). Este algoritmo esta

relativamente muy estudiado y muchos modelos han sido propuestos y verificados con grados de incremento en su eficiencia a través de simulaciones y medidas de Internet [MM, JS, M, 97], [JP, VF, JK, 98].

En implementaciones tradicionales del manejo de colas (*queue management*) en enrutadores, los paquetes son rechazados (*dropped*) cuando el *buffer* se llena (en tal caso el mecanismo se llama *Drop Tail*). Recientemente, otros mecanismos de manejo de colas han sido propuestos (el mas popular ha sido *Random Early Discard (RED), [SF, VJ, 97])*.

RED tiene el potencial de resolver algunos de los problemas descubiertos en Drop-Tail como es la sincronización de los flujos TCP y la correlación de los eventos de rechazo (múltiples paquetes rechazados en secuencia) dentro de un flujo TCP. En RED, los paquetes son aleatoriamente rechazados antes de que el *buffer* esté completamente lleno, y la probabilidad de rechazo incrementa con el promedio del tamaño de la cola.

RED es un poderoso mecanismo para el control de tráfico. Este puede proveer mejor utilización de los recursos de la red que Drop Tail si es adecuadamente utilizado, pero puede inducir a la inestabilidad de la red y a una mayor interrupción del tráfico si no es configurado apropiadamente. Desafortunadamente, la mayoría de los estudios proponen configuraciones RED (parámetros de configuración de RED óptimos) basados en simulaciones y métodos heurísticos, y no en estudios sistemáticos. El problema común es que cada prepuesta de configuración es solo buena para las condiciones de trafico particulares para las cuales se hizo el estudio, pero puede tener efectos negativos si es utilizado en otras condiciones, y dado que este trabajo de investigación está orientado a redes celulares inalámbricas las cuales tienen características variables debido a la interfaz aérea y a las condiciones de trafico de la red, se plantea un método general para la configuración de módulos de control de congestión RED, basados en un modelo de RED visto como un sistema de control con realimentación con flujos TCP. Se utiliza este modelo y los requerimientos de estabilidad y eficiencia para deducir un conjunto de parámetros de configuración de RED para un rango dado de características de tráfico y anchos de banda.

2.2.6 Mecanismo de Optimización Basado en la Compresión de las Cabeceras.

Existen diferentes perfiles de compresión de acuerdo a la recomendación que se desee implementar. En la recomendación RFC1144 se describe un método para la compresión de las cabeceras TCP/IP para aumentar el desempeño sobre comunicaciones seriales de baja velocidad. Aunque este modelo de optimización del protocolo TCP no fue diseñado para ser utilizado en un perfil inalámbrico, lo utilizamos como base para analizar la compresión de cabecera de los protocolos TCP e IP en un perfil inalámbrico, ya que al disminuir la cabecera del paquete, se ahorra ancho de banda del enlace, lo cual mejora la eficiencia del enlace.

De la misma forma, la recomendación RFC2507 define un esquema de compresión de las cabeceras de los protocolos IP, TCP y UDP para enlaces punto a punto. Estos métodos pueden ser aplicados a las cabeceras IPv6, IPv4, TCP y UDP y encapsulados en cabeceras IPv6 e IPv4.

Las cabeceras de los protocolos TCP y UDP pueden ser comprimidos hasta por debajo de 7 a 4 octetos [RFC2507], incluyendo los dos octetos de *checksum* de UDP y TCP. Esto en gran medida remueve el impacto negativo de las grandes cabeceras IP y permite el uso eficiente del ancho de banda en enlaces de baja y media velocidad.

Existen en la actualidad diferentes investigaciones sobre el mejoramiento de algunos protocolos utilizados en enlaces inalámbricos, como es el caso de la optimización de IP, la cual ha sido bastante estudiada y la cual ha dado como resultado una serie de recomendaciones las cuales son aun objeto de investigación. Una de estas investigaciones ha sido la implementación de mayores niveles de seguridad no solo para las cabeceras de las tramas sino también como de los datos *(payload)*; la recomendación RFC2401 *Security Architecture for the Internet Protocol*, es la recomendación para la implementación de los mecanismos de IPsec tanto para IPv4 como para IPv6, haciendo especial hincapié en la utilización de estos mecanismos en la nueva versión de IP.

A continuación se mostraran esquemas de compresión del protocolo TCP como algunos mecanismos de mejoramiento del rendimiento de este protocolo.

Los esquemas para la compresión tanto de las cabeceras TCP como IP tienen diferentes características y varían su funcionamiento dependiendo del tipo de red sobre la cual se transmite los datos, en la mayoría de estos esquemas se plantea un enlace entre dos terminales a través de un medio con pocas perdidas. Para el análisis de un esquema de optimización de los protocolos de transmisión de un perfil inalámbrico en un medio con perdidas sin un patrón regular, como es el caso de la mayoría de enlaces celulares, se debe tener en cuenta no solo mecanismos de compresión de la cabecera sino otros que aseguren que el comportamiento y características del protocolo no se degraden, es por eso que se incluyen en esta investigación, mecanismos como el envió de ACKs selectivos (Selective Acknowledge SACKs) y de otras opciones como la compresión de banderas las cuales se transmiten en el *handshaking* (la primera comunicación entre base estación y el dispositivo móvil).

2.2.6.1 Esquemas de compresión.

2.2.6.1.1 RFC1144, Compresión de las cabeceras TCP/IP para enlaces seriales de baja velocidad

Esta recomendación intenta reducir a lo más mínimo las cabeceras tanto de TCP como de IP, haciendo que el receptor reconstruya la cabecera de forma que solo se transmitan los campos que varían de forma no secuencial y que podrían ser generados teniendo en cuenta otros campos u otros protocolos.

La utilización de esta referencia es para conexiones en las que las perdidas de datos no son criticas, es decir que su utilización en redes inalámbricas no es recomendada debido a que estas tienen perdidas aleatorias de datos y poca estabilidad del ancho de banda, sin embargo debemos tenerla muy en cuenta debido a que nos sirve de base para las compresiones que utilizaremos en enlaces inalámbricos.

En la figura 2.7 se muestran los campos del protocolo TCP e IP, al unirlos obtenemos la cabecera TCP/IP. Cuyos campos están dispuestos de la siguiente manera:

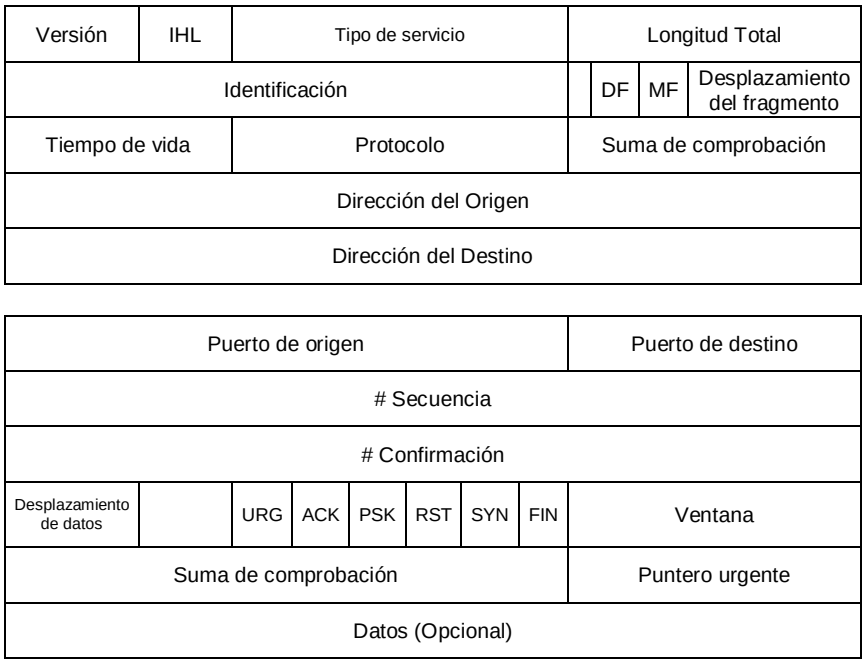


Figura 2. 7. Cabecera TCP/IP

La figura 2.8 muestra resaltados los campos que se suprimen al implementar la recomendación en su totalidad.

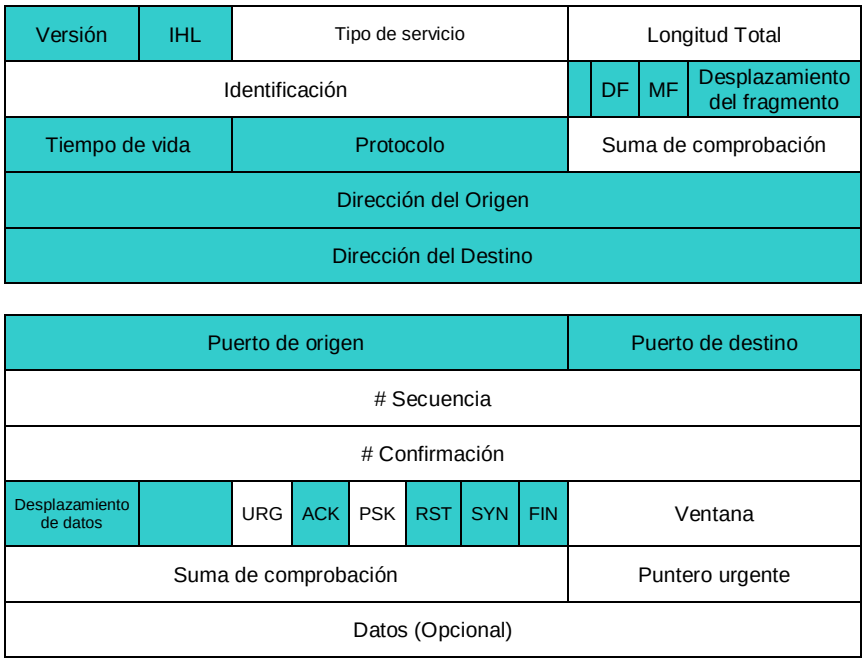


Figura 2. 8. Cabecera TCP/IP con los campos a ser suprimidos.

La recomendación no se limita a la optimización del protocolo TCP, sino que también optimiza la cabecera de IP.

El resultado es la siguiente cabecera TCP/IP:

<table>
<tr><td colspan="2">Tipo de servicio</td><td>Longitud Total</td></tr>
<tr><td colspan="3">Identificación</td></tr>
<tr><td colspan="3">Suma de comprobación</td></tr>
<tr><td colspan="3"># Secuencia</td></tr>
<tr><td colspan="3"># Confirmación</td></tr>
<tr><td>URG</td><td>PSK</td><td>Ventana</td></tr>
<tr><td colspan="2">Suma de comprobación</td><td>Puntero urgente</td></tr>
<tr><td colspan="3">Datos (Opcional)</td></tr>
</table>

Figura 2. 9 Cabecera TCP/IP Comprimida.

La forma en que se recomienda implementar el RFC1144 es la siguiente:

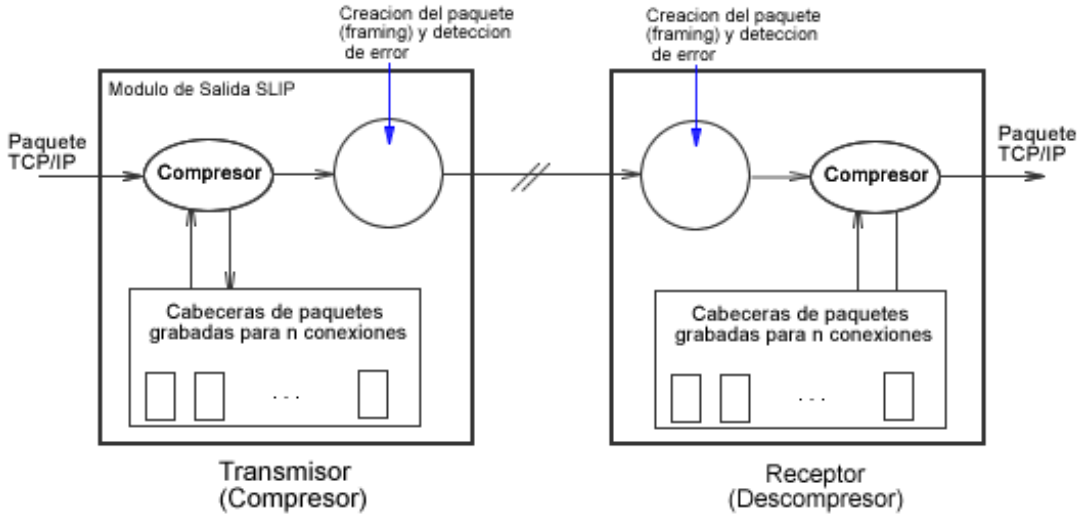


Figura 2. 10. Bloques Funcionales de la compresión RFC1144

El diagrama anterior muestra los bloques funcionales necesarios para realizar la optimización del protocolo. Los paquetes provenientes de una aplicación llegan al compresor el cual se

encarga de seleccionar los paquetes de acuerdo a si son paquetes TCP, paquetes IP o si tienen otro tipo de encabezado.

En realidad el compresor crea tres tipos diferentes de paquetes, TYPE_IP, UNCOMPRESSED_TCP o COMPRESSED_TCP.

Un paquete TYPE_IP, es una copia sin modificar de la entrada del paquete y su procesamiento no cambia el estado del compresor de ninguna forma.

Un paquete UNCOMPRESSED_TCP es un paquete idéntico a la entrada excepto que el campo del protocolo IP (byte 9) es cambiado de su valor '6' (protocolo TCP) a un numero de conexión. Además, el campo del estado asociado con el número de conexión es actualizado con una copia de la cabecera del paquete de entrada IP y TCP y el número de conexión es grabado en la última conexión enviada sobre esta línea serial.

Un paquete COMPRESSED_TCP contiene los datos (*payload*), si existe alguno de los paquetes originales pero las cabeceras IP y TCP son completamente reemplazadas por unas nuevas cabeceras comprimidas. El campo de estado de la conexión y la ultima conexión enviada son actualizadas por la entrada del paquete exactamente como para un paquete UNCOMPRESSED_TCP.

El procedimiento de decisión del compresor es la siguiente:

- Si el paquete no es un protocolo TCP, lo envía como TYPE_IP.
- Si el paquete es un IP fragmentado (por ejemplo, si cualquiera de los campos fragmentados de compensación no es cero o la mayoría de fragmentos es fijado en uno), lo envía como un paquete TYPE_IP.
- Si cualquiera de los bits de control de TCP, SYN, FIN o RST son fijados en uno o si el bit ACK es puesto en cero, se considera que el paquete no se puede comprimir y se envía un paquete TYPE_IP.

Si los paquetes provienen de una conexión TCP nueva, todo el perfil se almacena en un buffer el cual también se envía al receptor, éste lo almacena en un buffer y empieza a recibir solo los

campos que varían, reconstruyendo la cabecera de la trama con la información almacenada (perfil) y con los datos de los campos que han variado.

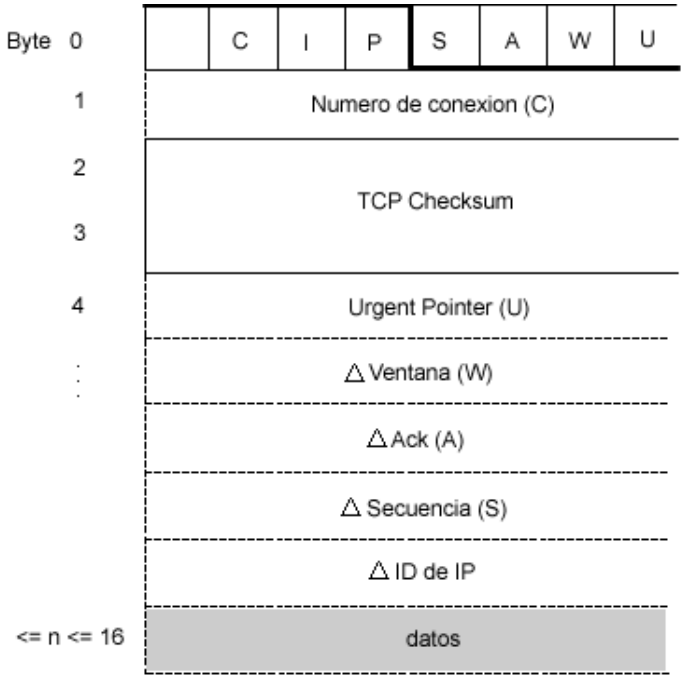


Figura 2. 11. Formato de un paquete TCP/IP comprimido

La figura 2.11 muestra el formato de un paquete TCP/IP el cual ha sido comprimido. Hay una mascara de cambio de bits que identifica cual de los campos esperados para cambiar ha cambiado por paquete, un numero de conexión se utiliza para localizar la copia del último paquete para esta conexión TCP, almacenada en el receptor, el campo TCP Checksum no se modifica, por lo tanto la integridad en el receptor se podrá analizar, entonces para cada bit fijado en la mascara, la cantidad del campo asociado cambia.

Los campos opcionales, controlados por la mascara, se muestran en la figura 2.11 en líneas discontinuas. En todos los casos, el bit es 1 si el campo asociado esta presente y 0 si el campo no se encuentra.

Puesto que los “delta” (∆) en el número de secuencia (S), etc., son usualmente pequeños; todos estos son codificados en un esquema de longitud variable, en la práctica, se manipula la mayoría de tráfico con ocho bits:

Un cambio de 1 hasta 255 es representado en un byte. Un campo con cero es improbable (un cambio de cero nunca es enviado) así que un byte con ceros señala una extensión: los dos siguientes bytes son el MSB y el LSB, respectivamente, cada uno con valor de 16 bits. Un número más grande que 16 bits fuerza al emisor a enviar un paquete sin comprimir. Por ejemplo, el decimal 15 es codificado en hexadecimal 0f, 255 como ff, 65534 como 00 ff fe, y cero como 00 00 00. Este esquema codifica y decodifica de una manera eficiente.

Los números enviados por el ***Número De Secuencia*** y ***ACK*** de TCP son la diferencia entre el valor actual y el valor en el anterior paquete (un paquete sin compresión es enviado si la diferencia es negativa o mayor que 64K). El número enviado por la ventana es también la diferencia entre el actual y el anterior valor. Sin embargo, cualquier cambio positivo o negativo es permitido desde que la ***Ventana*** sea un campo de 16 bits. El paquete de ***Puntero Urgente*** es enviado si ***URG*** es fijado como 1 (un paquete sin compresión es enviado si el puntero urgente cambia pero ***URG*** no es fijado a 1). Para el campo ***ID***, el número enviado es la diferencia entre el actual y el anterior valor. Sin embargo, a diferencia del resto de los campos comprimidos, el cambio es asumido ***I*** es 1 y no 0.

Hay dos casos especiales importantes:

1. El ***Número De Secuencia*** y ***ACK*** cambian en la cantidad de datos en el último paquete; la ***Ventana*** no cambia ni ***URG***.

2. El ***Número De Secuencia*** cambia en la cantidad de los datos en el último paquete; la ***Ventana*** no cambia ni ***URG***.

El primer caso es para el tráfico generado por la operación de *eco* entre terminales.

El segundo caso es para ningún tráfico generado por la operación de *eco* o una transferencia unidireccional de datos. Algunas combinaciones de los bits ***S, A, W*** y ***U*** de la mascara son utilizados para señalar estos casos especiales. ***U*** (datos urgentes) es muy raro de presentarse así que dos diferentes combinaciones son, ***S W U*** (utilizadas para el caso 1) y ***S A W U*** (utilizadas para el caso 2). Para evitar ambigüedades, un paquete sin compresión es enviado si los cambios actuales en un paquete son ***S * W U***.

Puesto que una conexión “activa” cambia rara vez (por ejemplo un usuario permanece en una ventana de *telnet* antes de cambiarse a una diferente ventana), el bit ***C*** deja el número de conexión para que este sea abreviado. Si ***C*** es 0, la conexión se asume que es la misma para el último paquete comprimido (*compressed*) o uno sin comprimir (*uncompressed*). Si ***C*** es 1, el número de conexión esta en el byte inmediatamente siguiente del cambio de la ***Mascara***.

2.2.6.1.2 Proceso de Compresión

El compresor es llamado con el paquete IP que será procesado y la estructura del estado de compresión para la línea serial saliente. Este retorna un paquete listo para empacar (*framing*) los datos y enlazar al “tipo*” (type)* de paquete que pertenece. Como se mencionó anteriormente, el compresor convierte cada paquete de entrada en un paquete TYPE_IP, UNCOMPRESSED_TCP o COMPRESSED_TCP.

Un paquete TYPE_IP es una copia sin modificar del paquete de entrada y al procesarlo no cambia el estado del compresor de ninguna forma.

Un paquete UNCOMPRESSED_TCP es idéntico al paquete de entrada excepto que el campo del protocolo IP (byte 9) es cambiado de su valor ‘6’ (Protocolo TCP) a un Número de Conexión. Además, el campo del estado asociado con el Número De Conexión es actualizado con una copia de la cabecera del paquete de entrada IP y TCP y el número de conexión es almacenado como la ultima conexión enviada sobre la línea serial.

Un paquete COMPRESSED_TCP contiene los datos (*payload*), si existe alguno, del paquete original, pero las cabeceras IP y TCP son completamente reemplazadas por unas nuevas cabeceras comprimidas. El *spot* (campo) del estado de la conexión y la ultima conexión enviada son actualizadas por la entrada del paquete exactamente como para un paquete UNCOMPRESSED_TCP.

El procedimiento de decisión del compresor es el siguiente:

- Si el paquete no es un protocolo TCP, lo envía como TYPE_IP.

- Si el paquete es un fragmento IP (por ejemplo cualquiera de los fragmentos del campo ***OFFSET*** no es cero o la mayoría de los fragmentos de bit es 1) lo envía como TYPE_IP.
- Si cualquiera de los bits de control de TCP ***SYN, FIN*** o ***RST*** son fijados en 1 o si el bit ***ACK*** es 0, se considera que el paquete no se puede comprimir y se envía como un paquete TYPE_IP.

Si un paquete lo hace a través de los anteriores chequeos, este los envía como UNCOMPRESSED_TCP o COMPRESSED_TCP:

- Si ningún estado de conexión puede ser encontrado que corresponda a la fuente del paquete y direcciones IP de destino y puertos TCP, algún estado es recuperado (el cual debería probablemente ser el menos utilizado recientemente) y un paquete UNCOMPRESSED_TCP es enviado.
- Si un estado de conexión es encontrado, la cabecera del paquete que lo contiene es cotejado con el paquete actual para asegurar que no fueron cambios inesperados. El protocolo IP, fragmentos de compensación, mas fragmentos, campos SYN, FIN, y RST fueron revisados anteriormente y la dirección de la fuente y el destino y los puertos fueron revisados como parte de la localización del estado. Por lo tanto los campos restantes por revisar son la versión del protocolo, la longitud del protocolo, el tipo de servicio, el tiempo de vida, datos de compensación, opciones IP (si existe alguna) y las opciones de TCP (si existe alguna). Si algunos de estos campos difieren entre dos cabeceras, un paquete UNCOMPRESSED_TCP es enviado.

Si todos los campos invariables cotejan, un intento es hecho para comprimir el paquete actual:

- Si la bandera ***URG*** esta en 1, los datos del campo ***Urgent*** es codificado (note que este puede ser cero) y el bit ***U*** es fijado en 1 en la Mascara. Desafortunadamente, si ***URG*** esta en 0, los datos del campo ***Urgent*** deben ser cotejados con el anterior paquete, si éste cambia, un paquete UNCOMPRESSED_TCP es enviado. (***Urgent*** no debería cambiar cuando ***URG*** esta en cero pero esto no es obligatorio).
- La diferencia entre el campo de la ***Ventana (Window)*** del paquete actual comparada con la del paquete anterior es computado y si no es cero, es codificado y el bit ***W*** es fijado en la Máscara.

- La diferencia entre los campos ***ACK*** es computado. Si el resultado es menor que cero o mas grande que 2^16 – 1, un paquete UNCOMPRESSED_TCP es enviado. De otra forma si el resultado no es cero, este lo codifica y el bit ***A*** es fijado en la Máscara.
- La diferencia entre los campos del ***Número de Secuencia*** es computado. Si el resultado es menor que cero o mas grande que 2^16 – 1, un paquete UNCOMPRESSED_TCP es enviado. De otra forma, si el resultado no es cero, este lo codifica y el bit ***S*** es fijado en la Máscara.

Una vez los cambios en los bits ***U, W, A*** y ***S*** han sido determinados, las codificaciones de *casos especiales* son comprobadas:
Si ***U, S y W*** son fijadas en 1, los cambios concuerdan con una de las siguientes codificaciones de *casos especiales*. Se envía un paquete UNCOMPRESSED_TCP.

- Si solo el bit ***S*** es fijado en 1, se comprueba si el cambio es igual a la cantidad de datos del usuario en el ultimo paquete, por ejemplo, substrayendo las longitudes de cabecera de TCP e IP desde el ultimo campo de longitud total y compara el resultado con el cambio de ***S***. Si estos son iguales, se fija en la mascara ***SAWU*** (el *caso especial* de "transferencia de datos unidireccional") y descarta el cambio del ***Numero de Secuencia*** codificado (el descompresor puede reconstruirlo desde que este conozca la longitud del paquete total y la longitud de la cabecera).
- Si solo se fijan en 1 los bits ***S y A***, se verifica si ambos cambiaron en la misma cantidad y esa cantidad es la cantidad de datos de usuario en el ultimo paquete. De ser así, se fija en 1 el cambio en la Máscara como ***SWU*** (el *caso especial* para el trafico de "eco interactivo -*echoed interactive*- ") y descarta los cambios codificados.
- Si nada cambia, se verifica si este paquete tiene o no datos de usuario (en el caso de no tenerlos es probable un ***ACK*** duplicado o una ***Ventana*** de prueba) o si el paquete anterior contiene datos de usuario (lo cual significa que este paquete es una retransmisión sobre una conexión con "*pipelining*"). En cualquier de estos casos, se envía un paquete UNCOMPRESSED_TCP.

Finalmente, la cabecera TCP/IP del paquete saliente es reemplazada con una cabecera comprimida:

- El cambio en el ***ID*** del paquete es computado y, si no es 1, la diferencia es codificada (note que este puede ser cero o negativo) y el bit ***I*** es fijado a 1 en la Máscara.
- Si el bit ***PUSH*** es fijado en 1 en el datagrama original, el bit ***P*** es fijado a 1 en la Máscara.
- Las cabeceras TCP e IP de los paquetes son copiados al campo de estado de conexión.
- Las cabeceras TCP e IP del paquete son descartadas y una nueva cabecera es adicionada conteniendo lo siguiente:
    - Lo acumulado, los cambios codificados
    - El TCP *Checksum* (si la nueva cabecera esta siendo construida "en lugar", el *checksum* puede haber sido sobrescrito y tendrá que ser tomado desde la copia de la cabecera en el *estado de conexión* o almacenada temporalmente antes que la cabecera original sea descartada).
    - El numero de conexión (si es diferente al último enviado en la línea serial). Esto también significa que la ultima conexión de la línea enviada sebe ser fijada con el numero de conexión y el bit ***C*** fijado en la Máscara.
    - El cambio de la Máscara.

En este punto, el paquete comprimido de TCP es pasado al *framer* para su transmisión.

2.2.6.1.3 Proceso de Descompresión

Debido a que el modelo de la comunicación es *simplex*, el procesamiento en el descompresor es mas simple que en el compresor--- todas las decisiones ha sido hechas y el descompresor simplemente hace lo que el compresor le ha dicho que haga.

El descompresor es activado con algún paquete entrante, la longitud y el tipo de los paquetes y la estructura del estado de compresión de la línea serial entrante. Un (posiblemente reconstruido) paquete IP será retornado.

El descompresor puede recibir cuatro tipos de paquetes: los tres generados por el compresor y un pseudo paquete TYPE_ERROR generado cuando el "*framer*" en el receptor detecta un error. El primer paso es un cambio "*switch*" en el tipo del paquete:

- Si el paquete es TYPE_ERROR o un tipo no reconocido, una bandera "*toss*" es fijada en el estado para forzar a los paquetes COMPRESSED_TCP a ser descartados hasta uno con el bit ***C*** fijado en 1 o un paquete que llegue con UNCOMPRESSED_TCP. Nada (un paquete nulo) es retornado.
- Si el tipo es TYPE_IP, una copia sin modificar de este es retornada y el estado no es modificado.
- Si el paquete es UNCOMPRESSED_TCP, el índice de estado desde el campo del protocolo IP es revisado. Si este es ilegal, la bandera "*toss"* es fijada en 1 y nada es retornado. De otra forma, la bandera *"toss"* se pone en 0, el índice es copiado al campo del estado de la ultima conexión recibida, una copia del paquete de entrada es hecho, el número del protocolo TCP es restaurado al campo del protocolo IP, la cabecera del paquete es copiada al campo del estado indicado, luego la copia del paquete es retornada.

Si el paquete no fue manipulado anteriormente, este es COMPRESSED_TCP y una nueva cabecera TCP/IP tiene que ser sintetizada de la información en el paquete, mas el último encabezado del paquete en el "*slot"* del estado. Primero, el número de conexión explicito o implícito es utilizado para localizar el "*slot"* de estado:

- Si el bit ***C*** es fijado en 1 en la máscara, el índice de estado es verificado. Si este es ilegal, la bandera *"toss"* es fijada en 1 y nada es retornado. De otra forma, la última conexión recibida es fijada en 1 para el índice de estado del paquete y la bandera *"toss"* se pone en 0.
- Si el bit ***C*** es puesto en 0 y la bandera *"toss"* es fijada en 1, el paquete es ignorado y nada es retornado.

En este punto, la última conexión recibida es el índice del apropiado *"slot"* de estado y el primer byte(s) del paquete comprimido (la máscara y posiblemente, el índice de conexión) han sido consumidos. Puesto que la cabecera TCP/IP en el campo de estado (*state*) debe finalizar reflejando el nuevo paquete que ha llegado, es más simple de aplicar los cambios desde el paquete a esa cabecera luego construir el paquete de salida desde la cabecera concatenada con los datos del paquete de entrada. (En la siguiente descripción, 'cabecera grabada' -*saved header*- es utilizado como una abreviatura para 'la cabecera TCP/IP grabada en el campo de estado'.).

- Los dos siguientes bytes en el paquete de entrada son el ***TCP Checksum.*** Ellos son copiados a la 'cabecera grabada'.
- Si el bit ***P*** es fijado en la Máscara, el bit TCP PUSH es fijado en la cabecera grabada. De otra forma el bit PUSH se pone en cero.
- Si los 4 bits de mas bajo orden (S, A, W y U) de la mascara de cambio son todas fijadas en 1 (el caso especial 'datos unidireccionales'), la cantidad de datos de usuario en el ultimo paquete es calculado por la sustracción de la longitud de la cabecera TCP e IP desde la longitud total del paquete IP en la cabecera grabada. La cantidad es luego agregada al número de secuencia en la cabecera grabada.
- Si S, W y U son fijados en uno y el bit A esta en cero (el caso especial de 'trafico terminal'), la cantidad de datos de usuario en el último paquete es calculado y agregado a ambos, el número de secuencia TCP y el campo ***ACK*** en la cabecera grabada.
- De otra forma, los bits de cambio de la Mascara son interpretados individualmente en el orden que el compresor los fija.
- Si el bit ***I*** es fijado en uno en la Mascara de cambio, el siguiente o los siguientes bytes del paquete entrante son decodificados y agregados al campo ***ID*** de IP del paquete grabado. De otra forma, uno es agregado al ***ID*** de IP.

En este punto, toda la información de la cabecera del paquete entrante ha sido agotada y solo quedan los datos. La longitud de los datos restantes es agregada a la longitud de la cabecera grabada IP y TCP.

2.2.6.1.4 Detección de errores

El ***Checksum*** de TCP a menudo no detecta dos únicos errores de bit separados por 16 bits. La correcta forma para tratar con este problema es proveer de una detección de errores en el nivel de *"framing"*. Puesto que el *"framing"* (al menos en teoría) puede ser ajustado a las características de una conexión en particular, la detección puede ser mínima o puede ser máxima dependiendo del enlace. Ya que la detección de errores en el paquete es hecha en el nivel de *"framing"*, el descompresor simplemente asume que este tendrá una indicación que el paquete actual fue recibido con errores. (El descompresor siempre ignora (descarta) un paquete con errores. Sin embargo, la indicación es necesaria para prevenir que el error sea propagado).

La política de “descarte de paquetes erróneos” da lugar a la segunda interacción de errores y la compresión. Considere la siguiente conversación:

Original	Enviado	Recibido	Reconstruido
1:A	1:A	1:A	1:A
2:BC	Δ1, BC	Δ1, BC	2:BC
4:DE	Δ2, DE	-	-
6:F	Δ2, F	Δ2, F	4:F
7:GH	Δ1, GH	Δ1, GH	5:GH

Figura 2. 12 Patrón de transmisión - recepción

(En la figura 2.12 cada entrada tiene la forma “número de secuencia de inicio: dato enviado” o “número de secuencia de cambio: dato enviado”.) Lo primero que se envía es un paquete sin comprimir *“uncompressed”*, seguido de cuatro paquetes comprimidos *“compressed”*. El tercer paquete contiene un error y es descartado. Para reconstruir el cuarto paquete, el receptor aplica el número de cambio de secuencia del entrante paquete comprimido al número de secuencia del último paquete recibido correctamente, paquete 2, genera un incorrecto número de secuencia para el paquete 4, después del error, todos los números de secuencia de paquetes serán erróneos, cambiados por la cantidad de datos en el paquete perdido.

Sin alguna clase de verificación, el anterior error resultaría en una invisible perdida de dos bytes en el medio de la transferencia (a partir de que el descompresor regenera los números de secuencia, los paquetes que contienen F y GH llegan al receptor de TCP con exactamente los números de secuencia que ellos tendrían que tener si el paquete DE nunca ha existido). A pesar de que algunas conversaciones TCP pueden sobrevivir con la falta de datos no es una práctica que sea promovida. Afortunadamente el ***Checksum*** de TCP, es una simple suma de los contenidos de los paquetes incluyendo los números de secuencia, esto detecta 100% de estos errores. Por ejemplo el ***Checksum*** computado del receptor para los últimos dos paquetes, del ejemplo anterior, siempre difieren del ***Checksum*** del paquete en 2.

Desafortunadamente, hay una forma de que la protección del ***Checksum*** de TCP puede fallar si los cambios en un paquete entrante comprimido son aplicados a la errónea conversación:

Considere dos conversaciones activas C1 y C2 y un paquete de C1 seguido por dos paquetes de C2. puesto que el numero de conexión no cambia, este es omitido a partir del segundo paquete de C2. Pero, si el primer paquete de C2 es recibido con un error en el CRC, el segundo paquete de C2 equivocadamente será considerado el siguiente paquete en C1. Puesto que el ***Checksum*** de C2 es un número aleatorio con respecto al número de secuencia de C1, hay al menos una probabilidad de 2^-16 de que este paquete sea aceptado por el receptor de TCP de C1. Para prevenir esto, después de una indicación de error del *"framer"* el receptor descarta los paquetes hasta que el reciba cualquier paquete COMPRESSED_TCP con el bit ***C*** fijado en 1 o un paquete UNCOMPRESSED_TCP. Por ejemplo los paquetes son descartados hasta que el receptor tenga un número de conexión explicita.

Para resumir esta sección, hay dos diferentes tipos de errores: por corrupción del paquete y por pérdida de sincronía en la conversación. El primer tipo es detectado en el descompresor desde el nivel de enlace del error de CRC, el segundo tipo de error se corrige en el receptor TCP de un invalido ***Ckecksum*** de TCP. La combinación de estos dos independientes mecanismos asegura que paquetes erróneos sean descartados.

2.2.6.1.5 Compresión de la cabecera de TCP independiente de los canales con ruido.

Stephen J. Perkins y Matt W. Mutka del Department of Computer Science de Michigan State University, crearon una variación de la recomendación RFC1144, y esta orientado a la compresión de la cabecera TCP/IP pero para enlaces con altos niveles de ruido en el medio.

Para abordar estos problemas se plantea la misma estrategia que se plantea en la referencia RFC1144 pero adicionando alguna información en la cabecera para evitar la perdida de sincronización entre el compresor y descompresor.

La modificación propuesta llamada *New VJ (Van Jacobson TCP header compression RFC1144)* se hace sin actualizar la copia almacenada de la cabecera pero continuando con los otros aspectos del RFC1144.

Existe una dependencia entre paquetes en el algoritmo de compresión del RFC1144 el cual podría evitar la descompresión de los paquetes y la continuidad de la transferencia de datos en

la conexión. La dependencia entre paquetes se origina debido a que los datos de la cabecera de los paquetes provenientes de la misma conexión se envían en forma de diferenciales ***Δ*** y la copia de la cabecera guardada en el receptor es la encargada de llevar la continuidad de los datos a través de los diferentes paquetes transmitidos; cuando uno de los paquetes no llega al compresor (debido al ruido presente en el medio, como es el caso de enlaces inalámbricos) se pierde la continuidad de los paquetes pero aun así el descompresor puede deducir cual podría ser el contenido del paquete, por el análisis de los anteriores diferenciales recibidos; este tipo de algoritmo de recuperación de errores no es confiable ya que puede darse el caso de perdida de paquetes pertenecientes a diferentes conexiones y el descompresor podría generar contenido de cabecera erróneo.

Para evitar esa interdependencia entre paquetes el nuevo algoritmo planteado aumenta un campo en los paquetes enviados, este campo lleva la información del ***Identificador de Conexión (CID).***

La perdida de sincronía significa que la copia de la cabecera del paquete en el descompresor es diferente a la que tiene el compresor. El efecto de esto es que el análisis del checksum fallará y los paquetes TCP serán descartados. Estos paquetes son descartados porque el descompresor es incapaz de determinar de forma confiable la conexión TCP a la cual es perteneciente cada paquete. Esta confusión es independiente de la perdida de sincronía reconocida por TCP. Esto es causado por una opción de la compresión dada por la recomendación RFC1144 que permite al compresor eliminar el identificador de conexión (Connection Idenitifier CID). Si un paquete llega y el CID no esta presente, el sistema utilizará la misma conexión del paquete anterior. Desafortunadamente, un paquete perdido puede dar a entender que una conexión cambia o puede que se haya perdido el paquete. La compresión utilizando la recomendación RFC1144 aplicaría una actualización de los paquetes sobre la conexión perdida. En algunas circunstancias, hay la posibilidad de que este tipo de error pase sin detectar aun para el control del TCP en el receptor. El algoritmo dado por la recomendación RFC1144 puede ser notificado de que un paquete se ha perdido por la capa mas baja. Si esta notificación es enviada, luego el descompresor descartara cada paquete hasta que reciba un paquete con un explicito CID. La solución obvia a este problema es enviar siempre el CID. En tales casos, un byte se agregará siempre a la cabecera TCP con el beneficio de que el descompresor no se confundirá con la identificación de cada paquete y su respectiva conexión. Desafortunadamente, esta solución no soluciona el problema de una perdida del paquete de actualización.

2.2.6.1.6 Remoción de la dependencia entre paquetes

Puesto que la recomendación RFC1144 utiliza un esquema de paquetes basado en una codificación diferencial, la pérdida de un paquete causa la perdida de sincronización de la conexión TCP. La modificación al algoritmo de la recomendación, permitiría mejorar la tolerancia a los errores. A diferencia del envío de diferenciales (Δs) en paquetes de paquetes base (fijos), se puede enviar diferenciales con base en paquetes que cambian rara vez. Aun más, para combatir el problema del número de secuencia descrito anteriormente se puede enviar los identificadores de conexión con cada paquete.

La principal fortaleza de este algoritmo modificado es que la mayoría de perdidas no causaran una perdida de sincronía y que las operaciones principales del algoritmo dado por la recomendación RFC1144 se mantienen sin cambio. Entre las desventajas se incluye un incremento en el consumo de memoria e incremento en la medida de la cabecera comprimida.

En el algoritmo dado por la recomendación RFC1144, los paquetes son codificados y decodificados contra una copia almacenada de la cabecera del paquete anterior. Los diferenciales son calculados y reemplazados en la cabecera en el paquete de salida. Mas importante aun es que la copia almacenada es actualizada para ser un duplicado de la cabecera del paquete que se ha enviado. En el descompresor, la copia almacenada es actualizada, después de la descompresión, se hace una copia de la cabecera recibida. Utilizando este mecanismo es fácil detectar las retransmisiones ya que ellas contendrán un número de secuencia de TCP igual o más pequeño que el valor de la cabecera almacenada. El algoritmo propuesto aquí es, el de no actualizar la copia almacenada de la cabecera como se propone en la recomendación RFC1144, pero si utilizando los otros aspectos de la recomendación. Puesto que todas las codificaciones diferenciales son hechas utilizando una base fija en ambos fines, la pérdida de paquetes no causaran perdidas de sincronía en los terminales.

Este esquema no es tan simple como remover el código de actualización de la cabecera. Se debe tener en cuenta el de mantener la suficiente información de tal forma que las políticas de retransmisión y de casos especiales dados por la recomendación RFC1144 funcionen apropiadamente. Esto conlleva a la expansión de la cantidad de información que se mantiene sobre cada conexión activa TCP. Junto con la cabecera TCP/IP que se mantiene, se debe

también mantener información sobre, la ventana TCP, acuses de recibo (ACKs), números de secuencias, y tamaños de los últimos paquetes pertenecientes a cada conexión. Esta información es siempre actualizada con cada paquete que llega mientras que la cabecera almacenada TCP/IP permanece sin cambio.

2.2.6.1.2 Mejoramiento del algoritmo de compresión de TCP/IP en enlaces inalámbricos

Anna Calveras-Augé, Miquel Arnau-Osorio, Josep Paradells-Aspas del Departament de Matemàtica Aplicada i Telemàtica, Universitat Politècnica de Cataluña, generaron un modelo de compresión basado también en la recomendación RFC1144, pero con algunas variaciones para adecuarse a ambientes con perdidas como es el caso de enlaces inalámbricos.

Los segmentos comprimidos con el algoritmo de compresión creado por Stephen J. Perkins y Matt W. Mutua, son algunos bytes más grandes debido a su filosofía de una base fija. El resultado de acumular encabezamientos reduce el rendimiento de las transmisiones sin ruido *(clean transmissions)*.

El incremento de un campo con el Identificador de Conexión, no mejora en gran medida la perdida de sincronización debida a perdidas de paquetes, por lo tanto en esta arquitectura se propone:

- La inclusión de un adicional, umbral mínimo.
- Un cambio en la *base* a diferencia del envió de un segmento sin comprimir (cabecera completa) cuando el umbral de encabezamientos es cumplido.

Este algoritmo propone mantener la sólida protección del algoritmo creado por Stephen J. Perkins y Matt W. Mutua contra segmentos perdidos o corruptos, y mejorar aun más el rendimiento sobre un canal dual.

El mecanismo planteado funciona de la siguiente forma. Cada nuevo segmento el cual llega al compresor es comprimido sobre una cabecera fija almacenada. Las diferencias entre los campos del segmento que llegan y la cabecera almacenada son determinadas. Esta base es obviamente la misma en el descompresor. Cuando las diferencias logran el nuevo umbral, se

elimina la cabecera fija almacenada. El segmento entrante actualizará la base sobre la cual mas incrementos serán computados. Esto reducirá las cabeceras, a partir de los siguientes incrementos será mas bajo y será codificado con menos bytes.

La utilización de una base variable y no de una base fija como es el caso de todos las otras recomendaciones genera un tamaño dinámico de los segmentos enviados entre compresor y descompresor, pero aun así la perdida de uno de los segmentos que actualizan la base del descompresor harían que los segmentos entrantes a éste no se descompriman de forma adecuada y sean rechazados.

Aunque este algoritmo es mucho mejor en cuanto al rendimiento ya que no envía una cabecera completa sino que uno de los segmentos lleva la información de cambio de la base, tiene la debilidad que si se pierde ese segmento se daña todo el proceso, teniendo el mismo problema que el algoritmo de la recomendación RFC1144.

2.2.6.1.3 Esquema robusto para la compresión de la cabecera TCP/IP para redes inalámbricas

Hongbin Liao, Qian Zhang, Wenwu Zhu, y Ya-Qin Zhang del Microsoft Research, China desarrollaron el siguiente algoritmo para la compresión del protocolo TCP para WAP.

En general el esquema de compresión de la cabecera, según lo planteado en el RFC1144, mantiene los contextos de un flujo en el compresor y los lados del descompresor, que contienen la información relevante para comprimir y descomprimir el paquete de la cabecera correctamente. Normalmente, la descompresión se mantendrá sincronizada con la compresión, así puede descomprimir la cabecera comprimida del paquete correctamente.

Pero bajo algunas condiciones, el contexto del descompresor puede ser inconsistente o fuera de sincronía con el compresor, por ejemplo las pérdidas del paquete debido al error del acoplamiento. Esta inconsistencia causará que los paquetes sucesivos no se puedan descomprimir correctamente y se caerá el enlace eventualmente. Este efecto, llamado error de propagación, durará hasta que los contextos sean sincronizados de alguna forma. Esto degradará el funcionamiento del TCP, especialmente en los enlaces que tengan un alto BER. Por ejemplo los canales inalámbricos.

La propuesta original del esquema de compresión de la cabecera TCP es el algoritmo de compresión de la cabecera Van Jacobson, VJHC dado en la recomendación RFC1144. Después de la transmisión de la primera cabecera sin comprimir TCP/IP, solamente se transmite la diferencia codificada de la cabecera precedente. En la mayoría de los casos, VJHC puede comprimir de 40 octetos completos de la cabecera TCP/IPv4 a solamente 4 octetos y mejora el funcionamiento del TCP significativamente en los enlaces con ancho de banda limitado. Sin embargo, debido a la codificación diferenciada, una vez que un delta se pierda en el enlace entre el compresor y el descompresor, una serie de paquetes reconstruidos en el contexto del descompresor serán descartados (extremo del receptor). Durante este período, no se enviará ningún reconocimiento de nuevo al receptor TCP, así la naturaleza del *self-clocking*[1] TCP se rompe. Como resultado, el emisor TCP es forzado a un *timeout*[2]. Cuando es usado sobre los enlaces inalámbricos, VJHC causa frecuentes *timeouts* y esto degrada el funcionamiento del TCP significativamente.

Degermark por ejemplo propone un esquema de compresión de la cabecera TCP/IP en enlaces con una tasa de perdidas de paquete bajo (*low-loss*) llamado IPHC (IP Header Compressor), basado en el esquema VJHC (RFC1144). Además de pocas modificaciones del esquema VJHC, IPHC agrega dos mecanismos simples, dos algoritmos y el mecanismo explícito de petición de la cabecera. El algoritmo es *TWICE,* un mecanismo local de reparación del contexto. Este asume que solo el número de secuencia o el número de ACK cambian durante la conexión y que los deltas entre paquetes consecutivos permanecen constantes en la mayoría de los casos. Sin embargo, tales suposiciones no son siempre verdades, especialmente cuando las opciones *timestamp* y *SACK* son adicionadas en la cabecera TCP. Esto es fijo para que el *TWICE* recupere el contexto simplemente aplicando el delta previo al contexto. Cuando el descompresor no puede reparar el contexto, puede enviar opcionalmente un paquete de CONTEXT_STATE de nuevo al compresor para indicar que uno o más contextos son inválidos. Sobre la recepción de tales peticiones, el compresor enviará cabeceras TCP/IP completas para re-sincronizar los contextos. Claramente, IPHC necesita por lo menos un enlace *round-trip* (ida-vuelta) para recuperar la inconsistencia.

[1] Reloj que tanto receptor como emisor de TCP tienen para la sincronización de los paquetes.
[2] Es un tiempo fijo al cabo del cual se emite una bandera para la retransmisión de un paquete dado.

CAPITULO III

3. PARAMETROS Y METRICAS PARA LA EVALUACION EXPERIMENTAL.

La utilización de un modelo del perfil TCP para dispositivos inalámbricos, permite que las implementaciones de éste sean más ágiles en diversas arquitecturas para aplicaciones inalámbricas, teniendo en cuenta el factor de globalización de las telecomunicaciones.

Plataformas para aplicaciones inalámbricas como WAP e iMode, han acordado investigar en este campo para implementarlo dentro de su conjunto de protocolos.

En agosto de 2001, NTTDoCoMo una compañía de telecomunicaciones de Japón, acordó con las empresas pertenecientes al WAP Forum, desarrollar un protocolo de transporte específico para el control de la transmisión de datos, el cual se llama perfil inalámbrico de TCP, el cual tiene características diferentes a las que la recomendación RFC0793 establece.

Las características del protocolo dadas en la recomendación RFC0793 tienden a que la transmisión falle o se degrade debido a un problema del algoritmo que se implementa en la recomendación, el problema fundamental de TCP como lo plantea la recomendación RFC0793, es que al ser utilizado en un medio inalámbrico con perdidas, este asume que éstas son debidas a la congestión de la red disminuyendo la ventana de transmisión perjudicando la transferencia de archivos. Debido a los comportamientos no deseados del protocolo se plantean recomendaciones de diferentes tipos destinados a mejorar las condiciones de algunos tipos de tráfico; esta investigación pretende ser una recomendación para la implementación del protocolo TCP para el enlace entre un dispositivo móvil y una gateway WAP.

A continuación se muestra una pequeña descripción de la recomendación RFC0793, de los mecanismos participantes en la comunicación entre dos terminales, así como de las características fundamentales que hacen del protocolo TCP uno de los mas utilizados en la actualidad.

3.1 Protocolo TCP (RFC0793)

TCP es un protocolo que proporciona transmisión fiable de paquetes de datos sobre redes. El nombre TCP/IP Proviene de dos protocolos importantes de la familia, el *Transmission Control Protocol* (TCP) y el *Internet Protocol* (IP).

El conjunto TCP/IP es la base del Internet que sirve para enlazar computadoras que utilizan diferentes sistemas operativos, incluyendo PC y servidores sobre redes de área local y área extensa. TCP/IP fue desarrollado y demostrado por primera vez en 1972 por el departamento de defensa de los Estados Unidos, ejecutándolo en el ARPANET una red de área extensa del departamento de defensa.

El Protocolo de Control de Transmisión, define un servicio clave proporcionado por Internet, es decir, una entrega confiable de flujo de datos. TCP proporciona una conexión dúplex completa (full dúplex) entre dos máquinas, permitiéndoles intercambiar de forma eficiente grandes volúmenes de datos.

Debido a que usa un protocolo de ventana deslizante, TCP puede hacer uso eficiente de una red, y es lo suficientemente flexible para operar sobre una gran variedad de sistemas de entrega ya que hace pocas suposiciones acerca del sistema fundamental de entrega. Gracias a que proporciona control de flujo, TCP permite que se comuniquen sistemas de la más amplia variedad de velocidades.

La unidad de transferencia básica usada por TCP es un segmento. Los segmentos se usan para pasar datos o información de control (por ejemplo, permitir al software TCP en dos máquinas establecer conexiones o romperlas). El formato del segmento permite a una máquina transportar acuses de recibo para los datos que fluyen en una dirección incluyéndolos en los encabezamientos del segmento de datos que fluye en la dirección opuesta.

TCP implementa el control de flujo haciendo que el receptor anuncie la cantidad de datos que desea recibir. Soporta también mensajes fuera de banda utilizando un medio para datos urgentes y fuerza la entrega usando un mecanismo push

El actual estándar TCP especifica un retroceso exponencial para los temporizadores de retransmisión y algoritmos de prevención de colisión como inicio lento, decrecimiento multiplicativo, y aumento aditivo. Además, TCP usa heurística para evitar la transferencia de paquetes pequeños.

3.1.1 RELACION CON OTROS PROTOCOLOS

El siguiente diagrama ilustra el lugar del protocolo TCP en la jerarquía de protocolos:

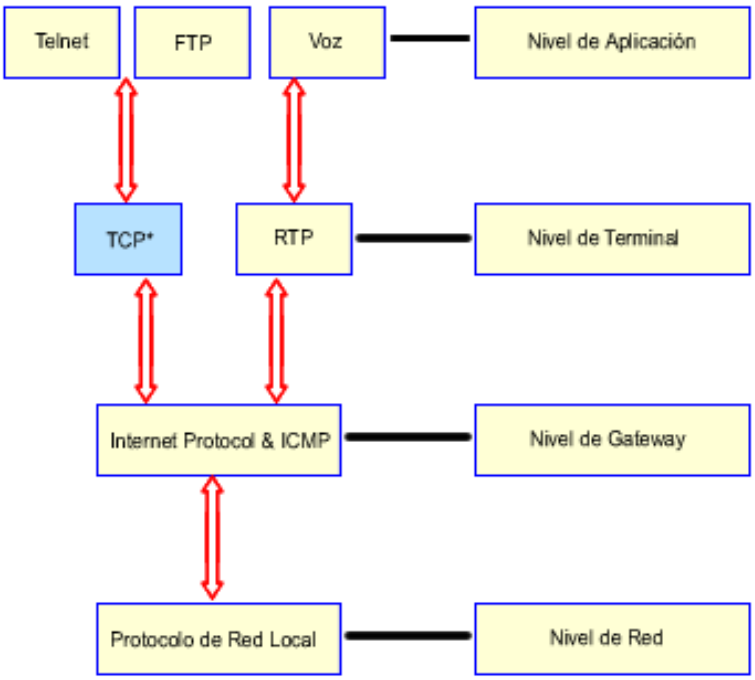


Figura 3. 1. Relaciones entre Protocolos

3.2 Parámetros y mecanismos de operación.

A continuación se mostrará de forma concisa el establecimiento de una conexión TCP, con el objetivo de tener una mejor referencia de los procesos que la maquina de estados de TCP genera.

3.2.1 Establecimiento de una conexión TCP

Para establecer una conexión, TCP utiliza un procedimiento llamado *handshake* en el cual los terminales se transmiten características tanto de la red en donde se encuentran, como las del estado de estos.

El primer segmento de un handshake puede identificarse porque tiene el bit SYN fijado en el campo de código. El segundo mensaje tiene tanto el bit SYN como el conjunto de bits ACK, indicando que da acuse de recibo del primer segmento SYN en tanto que el handshake continúa. El mensaje final de handshake es solo un acuse de recibo y se usa meramente para informar al destino que ambos lados están de acuerdo en que se ha establecido una conexión.

Usualmente, el software TCP en una máquina espera pasivamente por el handshake, y el software TCP en otra máquina lo inicia. Sin embargo, se diseña cuidadosamente el handshake para que trabaje aún si ambas máquinas intentan iniciar simultáneamente una conexión. Así, se puede establecer una conexión desde cualquier extremo o desde ambos extremos simultáneamente. Una vez que se ha establecido la conexión, los datos pueden fluir en ambas direcciones igualmente bien. No hay ni maestro ni esclavo.

El *handshake* de tres vías es tanto necesario como suficiente para la correcta sincronización entre los dos terminales de la conexión. Para comprender el por qué, recordar que TCP se construye sobre un servicio de entrega de paquetes no confiable, de forma que los mensajes pueden perderse, retardarse, duplicarse o entregarse a destiempo. Así, el protocolo debe usar un mecanismo temporizador y retransmitir las solicitudes perdidas. Se origina un problema si las solicitudes original y retransmitida llegan mientras la conexión está siendo establecida, usada y terminada. Un handshake de tres vías (más la regla que TCP ignora peticiones adicionales de conexión después que se ha establecido la conexión) resuelve estos problemas.

3.2.2 Números de secuencia inicial

El *handshake* de tres vías desempeña dos importantes funciones. Garantiza que ambos lados estén listos para transferir datos (y que ellos sepan que ambos están listos), y

permite que ambos lados se pongan de acuerdo sobre los números de secuencia inicial. Los números de secuencia se envían y son objeto de cuse de recibo durante el *handshake*. Cada máquina debe escoger un número de secuencia inicial al azar que usará para identificar bytes en el flujo que está enviando. Los números de secuencia pueden no siempre comenzar en el mismo valor. En particular, TCP no puede escoger la secuencia cada vez que crea una conexión. Es importante que ambos lados estén de acuerdo con aquellos usados en los segmentos de datos.

Para ver cómo las máquinas pueden ponerse de acuerdo sobre los números de secuencia para los dos flujos después de solo tres mensajes, se debe tener en cuenta que cada segmento contiene tanto un campo para número de secuencia como un campo de acuse de recibo. La máquina que inicia un handshake, llamada A, pasa su número de secuencia inicial, x, en el campo de secuencia del primer segmento SYN en el handshake de tres vías. La segunda máquina, B, recibe el SYN, graba el número de secuencia, y replica enviando su número de secuencia inicial en el campo de secuencia tanto como un acuse de recibo que especifica que B espera el octeto x+l. en el mensaje final del handshake, A "acusa recibo" recibiendo de B todos los octetos a través de y. en todos los casos, los acuses de recibo siguen la convención de usar el número del *siguiente* octeto esperado.

Se ha descrito cómo usualmente TCP realiza el handshake de tres vías intercambiando segmentos que contienen una cantidad mínima de información. Debido al diseño del protocolo, es posible enviar datos junto con los números de secuencia inicial en los segmentos de handshake. En tales casos, el software TCP debe retener los datos hasta que se complete el handshake. Una vez que se ha establecido una conexión, el software TCP puede liberar los datos que retenía y entregarlos rápidamente a un programa de aplicación en espera.

3.2.3 Tiempos de espera máximos y retransmisión

Una de las ideas más importantes y complejas en TCP está incrustada en la manera en como se maneja el tiempo de espera máximo y la retransmisión. Como otros protocolos confiables, TCP espera que el destino envíe acuses de recibo cuando quiera que reciba de forma exitosa nuevos octetos desde el flujo de datos. Cada vez que envía un segmento, TCP inicializa un temporizador y espera un acuse de recibo. Si el temporizador expira

antes que los datos en el segmento hayan sido objeto de acuse de recibo, TCP asume que el segmento se perdió o se corrompió y lo retransmite.

Para comprender porqué el algoritmo de retransmisión TCP difiere del algoritmo empleado en muchos protocolos de red, se necesita recordar que TCP se diseñó para emplearse en un ambiente de Internet (redes cableadas). En una Internet, un segmento que viaja entre un par de máquinas puede moverse por una red única, con bajo retardo (por ejemplo, una LAN de alta velocidad), o puede viajar a través de múltiples redes intermedias pasando por múltiples enrutadores. Así, es imposible saber *a priori* que tan rápido los acuses de recibo retornarán a la fuente. Además, el retardo en cada enrutador depende del tráfico, de manera que el tiempo total requerido por un segmento para viajar hacia el destino y el de un acuse de recibo para retornar a la fuente varía de forma dramática de un instante al siguiente. La figura 3.10, que muestra las mediciones de los tiempos de ida y vuelta a través de Internet global para 100 paquetes consecutivos, ilustra el problema. El software TCP debe acomodarse tanto a las grandes diferencias en el tiempo requerido para alcanzar los diversos destinos como a los cambios en tiempo requeridos para alcanzar un destino dado a medida que varía la carga de tráfico.

TCP se acomoda a los retardos variables de Internet utilizando un *algoritmo adaptativo de retransmisión.* En esencia, TCP vigila el rendimiento de cada conexión y deduce valores razonables para los tiempos de espera máximos. Dado que el rendimiento de una conexión cambia, TCP revisa su valor de tiempo de espera máximo (esto es, se adapta al cambio).

Para recolectar los datos necesarios para un algoritmo adaptativo, TCP registra el tiempo al cual se envía cada segmento, y el tiempo al cual llega un acuse de recibo para los datos en ese segmento. A partir de los dos tiempos, TCP calcula el tiempo transcurrido conocido como una *muestra de tiempo de ida y vuelta o muestra de ida y vuelta.* Cuando quiera que obtenga una nueva muestra de ida y vuelta, TCP ajusta su noción de tiempo de ida y vuelta promedio para la conexión. Usualmente, el software TCP almacena el tiempo de ida y vuelta estimado, RTT; como un promedio ponderado y usa las muestras de ida y vuelta nuevas para cambiar el promedio lentamente. Por ejemplo, cuando se calcula un nuevo promedio ponderado, se usa una técnica primitiva de promediado con un factor de ponderado constante α, donde $0<=\alpha<l$, para ponderar el antiguo promedio contra la muestra de ida y vuelta más reciente:

$$RTT = (\alpha \cdot RTT\_antiguo) + ((1-\alpha) \cdot RTT\_nueva)$$

Al escoger un valor de α cercano a l se hace el promedio ponderado inmune a los cambios que duran un corto tiempo (por ejemplo, un único segmento que encuentra un retardo largo). Al escoger un valor de α cercano a 0 se hace que el promedio ponderado responda a cambios en retardo muy rápidamente.

Cuando envía un paquete, TCP calcula un valor de tiempo de espera máximo como una función del estimado corriente de ida y vuelta. Las primeras implementaciones de TCP usaban un factor constante de ponderación **β(β>l)**, y hacían en tiempo de espera máximo mayor que el estimado corriente de ida y vuelta:

$$Tiempo\_de\_espera\_máximo(timeout) = \beta \cdot RTT$$

Escoger un valor para β puede ser difícil. Por un lado, para detectar la pérdida de paquete de forma rápida, el tiempo de espera máximo debe estar cerca del tiempo corriente de ida y vuelta (esto es, β debe estar cerca de l). La detección rápida de pérdida de paquete mejora el rendimiento total porque TCP no esperará un tiempo largo innecesario antes de la retransmisión. Por otro lado, si β=l, TCP está muy ansioso-cualquier pequeño retardo causará una retransmisión innecesaria, que desperdicia ancho de banda de la red. La especificación original recomienda fijar β=2; trabajos más recientes descritos después han producido mejores técnicas para ajustar el tiempo de espera máximo.

3.2.4 Medición precisa de las muestras de ida y vuelta (round trip time - rtt)

En teoría, la medición de una muestra de ida y vuelta es trivial, se trata de la sustracción del tiempo al cual se envía el segmento del tiempo al cual llega el acuse de recibo. Sin embargo, surgen complicaciones debido a que TCP usa una configuración de acuse de recibo acumulativa en la cual un acuse de recibo se refiere a los datos recibidos, y no a la instancia de un datagrama específico que porta los datos. Considérese una retransmisión, TCP forma un segmento, lo coloca en un datagrama y lo envía, el temporizador expira, y TCP envía el segmento de nuevo en un segundo datagrama. Dado que ambos datagramas llevan exactamente los mismos datos, el remitente no tiene forma de saber si un acuse de

recibo corresponde al datagrama original o al retransmitido. Este fenómeno se ha denominado *ambigüedad del acuse de recibo*, y se dice que los acuses de recibo TCP son *ambiguos.* Asociar el acuse de recibo con la transmisión original puede hacer que el tiempo estimado de ida y vuelta crezca sin límites en casos donde una red presente perdida de paquetes. Si un acuse de recibo llega después de una o más retransmisiones, TCP medirá la muestra de ida y vuelta desde la transmisión original y calculará un nuevo RTT usando la muestra excesivamente grande. Así, RTT crecerá ligeramente. La próxima vez que TCP envía un segmento, el RTT más largo producirá tiempos de espera máximos ligeramente más largos, de modo que si un acuse de recibo llega después de una o más retransmisiones, la próxima muestra de tiempo de ida y vuelta será aún más larga y así sucesivamente.

Asociar el acuse de recibo con la retransmisión más reciente puede también fracasar. Considérese lo que sucede cuando el retardo extremo a extremo aumenta repentinamente. Cuando TCP envía un segmento, usa el antiguo estimado de ida y vuelta para calcular un tiempo de espera máximo, que es ahora demasiado pequeño. El segmento llega y el acuse de recibo empieza su retorno, pero el aumento en el retardo significa que el temporizador expira antes que el acuse de recibo llegue, y TCP retransmite el segmento. Poco después de que TCP retransmite, el primer acuse de recibo llega y es asociado con la retransmisión. La muestra de ida y vuelta será mucho más pequeña y producirá un ligero decrecimiento en el tiempo de ida y vuelta estimado, RTT. Desafortunadamente, al disminuir el tiempo de ida y vuelta estimado se garantiza que TCP fijará el tiempo de espera máximo demasiado pequeño para el próximo segmento. En últimas, el tiempo de ida y vuelta estimado puede estabilizarse en un valor T, tal que el tiempo de ida y vuelta correcto es ligeramente mas largo que algún múltiplo de T.

Las implementaciones de TCP que asocian los acuses de recibo con la retransmisión más reciente se han observado en estado estable con RTT ligeramente menor que la mitad del valor correcto (esto es, TCP envía cada segmento exactamente dos veces aunque no ocurre pérdida).

Las medidas del RTT de conexiones TCP pueden ser efectuadas en la Estación Base (de una célula inalámbrica) utilizando los paquetes SYN. Sin embargo, los paquetes SYN pueden experimentar diferentes retrasos que los que tienen los paquetes normales de

datos. Este puede ser debido al retraso que se efectúa al inicio de la conexión en las capas mas bajas de TCP así como a la manipulación de los paquetes dentro de los enrutadores y maquinas dentro del camino virtual del enlace. Otro método para medir el RTT es hacer las mediciones durante la fase de *Slow Start*. La figura 2 muestra la línea de tiempo de una conexión TCP.

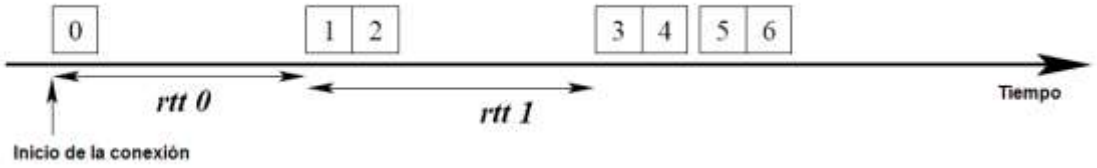


Figura 3. 2. Línea de tiempo de una conexión TCP y la medición de RTT

3.2.5 Algoritmo de karn y retroceso del temporizador

TCP no debe actualizar el estimado de ida y vuelta (RTT) para los segmentos retransmitidos. Esta idea, conocida como *Algoritmo de Karn,* evita totalmente el problema de acuses de recibo ambiguos ajustando solamente la ida y vuelta estimada para los acuses de recibo no ambiguos (acuses de recibo que llegan para los segmentos que se han transmitido solamente una vez).

Una implementación simple del algoritmo de Karn, una que solamente ignore los tiempos de los segmentos retransmitidos, puede llevar también a fallas. Cuando TCP envía un segmento después que se produce un pequeño aumento en el retardo TCP calcula un tiempo de espera máximo usando el estimado de ida y vuelta existente, el tiempo de espera máximo será demasiado pequeño para el nuevo retardo y forzará a la retransmisión. Si TCP ignora acuses de recibo de segmentos retransmitidos, nunca actualizará el estimado y el ciclo continuará.

Para acomodarse a tales fallas, el algoritmo de Karn necesita que el transmisor combine los tiempos de espera máximos de retransmisión con una estrategia de *retroceso de temporizador*. Sin embargo, si el temporizador expira y causa una retransmisión, TCP aumenta el tiempo de espera máximo. De hecho, cada vez que debe retransmitir un segmento, TCP aumenta el tiempo de espera máximo (para evitar que los tiempos de espera máximos se vuelvan excesivamente largos, la mayoría de las implementaciones

limitan los aumentos a un límite superior que es mayor que el retardo a lo largo de cualquier trayectoria en una red).

Las implementaciones utilizan diversas técnicas para calcular el retroceso. La mayoría eligen un factor multiplicativo γ y fijan el nuevo valor a:

$$Nuevo\_tiempo\_de\_espera\_máximo = \gamma \cdot tiempo\_de\_espera\_máximo$$

Típicamente, γ es 2. (Se argumenta que valores de γ menores que 2 llevan a inestabilidad). Otras implementaciones usan una tabla de factores multiplicativos permitiendo en cada paso retrocesos arbitrarios.

El algoritmo de Karn combina la técnica de retroceso con la estimación de ida y vuelta para resolver el problema de nunca incrementar los estimados de ida y vuelta. Generalmente hablando, cuando una red tiene pérdidas o retrasos, el algoritmo de Karn separa el cálculo del valor del tiempo de espera máximo del estimado de ida y vuelta actual. Utiliza el estimado de ida y vuelta para calcular un valor de tiempo de espera máximo inicial, pero entonces retrocede el tiempo de espera máximo en cada retransmisión hasta que puede transferir de forma exitosa un segmento. Cuando envía segmentos subsecuentes, retiene el tiempo de espera máximo que resulta del retroceso. Finalmente, cuando llega un acuse de recibo correspondiente a un segmento que no necesitó retransmisión, TCP recalcula el estimado de ida y vuelta y vuelve a fijar en forma acorde el tiempo de espera máximo. La experiencia muestra que el algoritmo de Karn trabaja bien aún en redes con altas pérdidas de paquetes, lo cual es conveniente para su empleo en redes inalámbricas en las cuales se presentan múltiples tipos de perdidas de paquetes.

3.2.6 Respondiendo a la alta variabilidad en el retardo

La investigación de la estimación de ida y vuelta (RTT) han mostrado que los cálculos descritos antes no se adaptan a un amplio campo de variaciones en el retardo. La teoría de colas de espera sugiere que la variación en el tiempo de ida y vuelta, σ varía proporcionalmente a: $\left(\frac{1}{1-L}\right)$

Donde L es la carga de red actual, 0<=L<=l, Si una red funciona al 50% de su capacidad, se espera que el retardo de ida y vuelta varíe en un factor de $\pm 2\sigma$, ó 4. Cuando la carga alcanza el 80%, se espera una variación de 10. La norma original TCP especificaba la técnica para estimar el tiempo de ida y vuelta que se describió antes. Usando esa técnica y limitando β al valor sugerido de 2 se implica que la estimación de ida y vuelta se puede adaptar a cargas de a lo más 30%.

La especificación de 1989 de TCP requiere implementaciones para estimar tanto el tiempo promedio de ida y vuelta como la variabilidad, y usar la variabilidad estimada en lugar de la constanteβ. En consecuencia, las nuevas implementaciones de TCP se pueden adaptar a un campo más amplio de variación en el retardo y producir un rendimiento total sustancialmente más alto. Las aproximaciones necesitan pocos cálculos y se pueden derivar implementaciones eficientes a partir de las siguientes ecuaciones simples:

$$DIFF = muestra - viejoRTT$$
$$RTT\_suavizado = viejoRTT + \delta \cdot DIFF$$
$$DEV = viejaDEV + \rho(ABS(DIFF) - viejaDEV)$$
$$Tiempo\_de\_espera\_máximo = RTT\_suavizado + \eta \cdot DEV$$

Donde DEV es la desviación media estimada, δ es la fracción entre 0 y l que controla que tan rápido la nueva muestra afecta el promedio ponderado, ρ es una fracción entre 0 y l que controla que tan rápido la nueva muestra afecta la desviación media, y η es un factor que controla que tanto afecta la desviación el tiempo de espera máximo de ida y vuelta. Para hacer el cálculo eficiente, TCP elige δ y ρ cada uno como un inverso de una potencia de dos, escala el cálculo por 2^n para un *n* apropiado, y usa aritmética entera. La investigación sugiere que los valores de $\delta = l/2^3$, $\rho = l/2^2$, y n=3, trabajarán bien. El valor original de η en 4.3BSD UNIX era de 2; se cambió a 4 en 4.4BSD UNIX.

3.2.7 Respuesta a la congestión

Es posible ver que las implementaciones TCP pueden diseñarse considerando la interacción entre los dos puntos extremos de una conexión y los retardos de comunicación

entre esos puntos. En la práctica, sin embargo, TCP debe reaccionar también a la *congestión* de todo el enlace (Internet). La congestión es una condición de retardo severo causada por una sobrecarga de datagramas en uno o más puntos de conmutación (por ejemplo, enrutadores). Cuando se presenta congestión, se incrementan los retardos y el enrutador comienza a colocar en cola de espera los datagramas hasta que puede enrutarlos. Se debe recordar que cada enrutador tiene una capacidad de almacenamiento finita y que los datagramas compiten por ese espacio de almacenamiento (esto es, en un datagrama basado en Internet, no existe preasignación de recursos a conexiones TCP individuales). En el peor de los casos, el número total de paquetes que llegan al enrutador congestionado crece hasta que se alcanza su capacidad y empieza a rechazar paquetes.

Los puntos extremos (terminales) normalmente no conocen los detalles de donde ha ocurrido la congestión o el por qué. Para ellos, la congestión significa simplemente mayor retardo. Desafortunadamente, la mayoría de protocolos de transporte usan tiempo de espera máximo y retransmisión, de manera que responden a un mayor retardo retransmitiendo paquetes. La retransmisión agrava la congestión en lugar de aliviarla lo cual se intensifica en redes inalámbricas donde los recursos de ancho de banda son mucho menores que en redes cableadas. Si no se verifica, el tráfico incrementado producirá mayor retardo, llevando a mayor tráfico, y así sucesivamente, hasta que la red se vuelve inútil. La condición se conoce como *colapso por congestión.*

Para evitar el colapso por congestión, TCP debe reducir las velocidades de transmisión cuando sucede congestión. Los enrutadores vigilan la longitud de las colas de espera y usan una técnica como extinción de fuente ICMP para informar a los host que ha ocurrido congestión, pero protocolos de transporte como TCP pueden ayudar a evitar la congestión reduciendo las tasas de transmisión de forma automática cuando quiera que se presenten retardos.

Por supuesto, se deben construir de forma cuidadosa algoritmos para evitar la congestión ya que aun bajo condiciones normales de operación una red inalámbrica exhibirá una amplia variación en los retardos de ida y vuelta debido no solo a verdadera congestión de paquetes sino a confusión de los terminales debido al tratamiento de pérdida de paquetes como congestión.

Para evitar la congestión, el estándar TCP recomienda usar dos técnicas: *arranque lento (Slow Start)* y *decrecimiento multiplicativo.* Ellas están relacionadas y se puede fácilmente hacer su implementación. Se dice que para cada conexión, TCP debe recordar el tamaño de la ventana del receptor (i.e., el tamaño de buffer anunciado en el acuse de recibo). Para controlar la congestión TCP mantiene un segundo límite, denominado *límite de ventana de congestión* o *ventana de congestión.* En cualquier instante, TCP actúa como si el tamaño de ventana es:

Ventana_permitida= min (anuncio del receptor, ventana de congestión)

En estado estacionario en una conexión sin congestión, la ventana de congestión es del mismo tamaño que la ventana del receptor. Al reducirse la ventana de congestión se reduce el tráfico que TCP inyectará en la conexión

Debido a que TCP reduce la ventana de congestión a la mitad, decrece exponencialmente la ventana, si continúa la pérdida. En otras palabras, si hay posibilidad de congestión, TCP reduce el volumen de tráfico exponencialmente y la velocidad de retransmisión exponencialmente. Si la pérdida continúa, TCP limita eventualmente la transmisión a un solo datagrama y continúa a valores de tiempo de espera máximo del doble antes de la retransmisión. La idea es proporcionar una reducción de tráfico rápida y significativa para suministrar a los enrutadores el tiempo suficiente para despejarse de los datagramas ya en sus colas de espera.

TCP no revierte el decrecimiento multiplicativo y tampoco duplica la ventana de congestión cuando comienza el tráfico a fluir de nuevo. Al hacerlo así se produciría un sistema inestable que oscila ampliamente entre no tráfico y congestión. En lugar de esto, TCP utiliza una técnica denominada *Slow Start* para aumentar aditivamente la tasa de transmisión arrancando la ventana de congestión con un tamaño de un solo segmento y aumenta la ventana de congestión en un segmento cada vez que llega un acuse de recibo. *Slow Start* evita la saturación de la red con tráfico adicional, inmediatamente después de que la etapa de congestión finaliza o cuando comienza de forma repentina una nueva conexión.

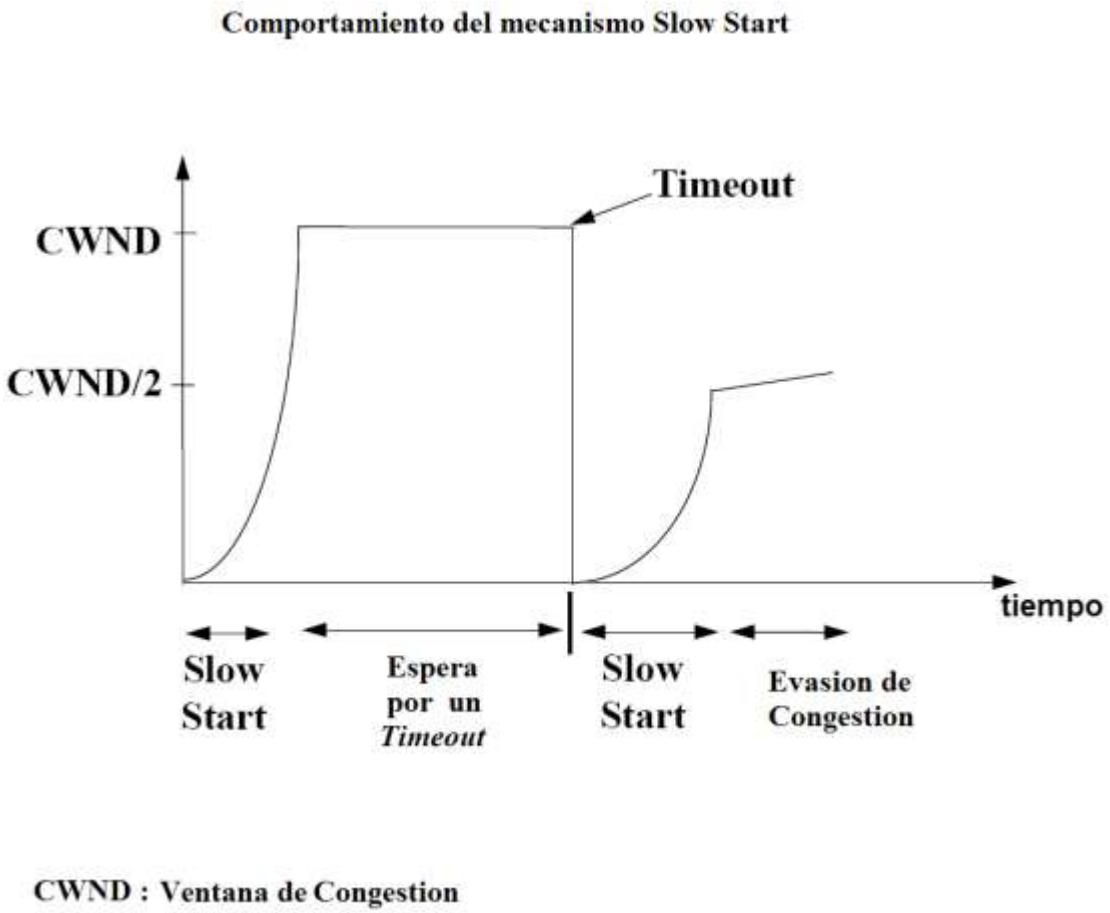


Figura 3. 3. Comportamiento del mecanismo Slow Start

El término arranque lento puede ser una designación no aplicable porque bajo condiciones ideales, el arranque no es muy lento. TCP inicializa la ventana de congestión en l, envía un segmento inicial, y espera. Cuando llega el acuse de recibo, aumenta la ventana de congestión a 2, envía dos segmentos, y espera. Cuando dos acuses de recibo llegan, cada uno incrementa la ventana de congestión en l, de modo que TCP envía 4 segmentos. Los acuses de recibo para aquellos incrementarán la ventana de congestión hasta 8. Dentro de cuatro tiempos de ida y vuelta, TCP puede enviar 16 segmentos, a menudo suficientes para que el receptor alcance el límite de la ventana. Aun para ventanas extremadamente grandes, solo toma $\log_2 N$ idas y vueltas antes que TCP pueda enviar N segmentos.

Para evitar que aumente demasiado rápido el tamaño de la ventana y se genere congestión adicional, TCP coloca una restricción adicional. Una vez que la ventana de congestión alcanza la mitad de su tamaño original antes de la congestión, TCP ingresa en

una fase de *Evasión de Congestión* y disminuye la velocidad del incremento. Durante la evasión de congestión, se aumenta el tamaño de la ventana de congestión en l solo si todos los segmentos en la ventana han sido sujeto de acuse de recibo.

Tomados en conjunto, el incremento por arranque lento, el decrecimiento multiplicativo, la evasión de congestión, la medición de la variación y el retroceso exponencial del temporizador mejoran el desempeño de TCP de forma dramática sin añadir ninguna tarea computacional importante al software del protocolo. Las versiones de TCP que usan estas técnicas han mejorado el desempeño de las versiones previas en factores que van de 2 a 10.

3.2.8 Conflicto de ventana tonta y paquetes pequeños

Los investigadores que ayudaron a desarrollar TCP observaron un serio problema de rendimiento que podía producirse cuando las operaciones que emitían y recibían operan a diferentes velocidades, para comprender el problema, se debe recordar que TCP almacena en buffer los datos que llegan, y considere lo que sucede si una aplicación que recibe escoge leer los datos que llegan un octeto a la vez. Cuando se establece primero una conexión, el TCP que recibe asigna un buffer de Kbytes, y usa el campo WINDOW en segmentos de acuse de recibo para dar a conocer al remitente el tamaño disponible de buffer. Si la aplicación que envía genera datos rápidamente, el TCP que envía transmitirá segmentos con datos para toda la ventana. Eventualmente, el remitente recibirá un acuse de recibo que especifica que toda la ventana se ha llenado, y que no queda espacio en el buffer del receptor.

Cuando la aplicación que recibe lee un octeto de datos desde un buffer lleno, se vuelve disponible un octeto de espacio. Se dice que cuando el espacio se vuelve disponible en su buffer, TCP en la máquina que recibe genera un acuse de recibo que usa el campo WINDOW para informar al remitente. En el ejemplo, el receptor dará a conocer una ventana de 1 octeto. Cuando se da cuenta que el espacio está disponible, el TCP que envía responde transmitiendo un segmento que contiene un octeto de datos.

Aunque dar a conocer una ventana de un solo octeto funciona correctamente para mantener el buffer del receptor lleno, produce una serie de pequeños segmentos de

datos. El TCP que envía debe componer un segmento que contenga un octeto de datos, colocar el segmento en un datagrama IP, y transmitir el resultado. Cuando la aplicación que recibe lee otro octeto, TCP genera otro acuse de recibo, que causa que el remitente transmita otro segmento que contiene un octeto de datos. La interacción resultante puede alcanzar un estado estable en el cual TCP envía un segmento separado por cada octeto de datos.

Transferir segmentos pequeños consume innecesariamente el ancho de banda de la red e introduce cálculos innecesarios que desperdician el ancho de banda. La transmisión de segmentos pequeños consume innecesariamente el ancho de banda de la red porque cada datagrama lleva solo un octeto de datos; la relación de encabezamiento a datos es grande. El Software TCP que envía debe asignar el espacio de buffer, formar el encabezamiento de un segmento, y calcular la suma de verificación para el segmento. De forma semejante, el software IP en la máquina que envía debe encapsular el segmento en un datagrama, calcular la suma de verificación del encabezamiento, enrutar el datagrama y transferirlo a la interfaz de red apropiada. En la máquina que recibe, IP debe verificar la suma de verificación del encabezamiento IP y pasar el segmento a TCP. TCP debe verificar la suma de verificación del segmento, examinar el número de secuencia, extraer los datos y colocarlos en un buffer.

Aunque se ha descrito cómo se producen segmentos pequeños cuando un receptor da a conocer una ventana pequeña disponible, el remitente puede también ocasionar que cada segmento contenga una pequeña cantidad de datos. Por ejemplo, imaginar una implementación TCP que genera datos de forma agresiva cuando quiera que estén disponibles, y considérese lo que sucede si una aplicación que envía genera datos un octeto a la vez. Después que la aplicación genera un octeto de datos, TCP crea y transmite un segmento. TCP puede enviar también un segmento pequeño si una aplicación genera datos en bloques de tamaño fijo de B octetos, y el TCP que envía los extrae desde el buffer en bloques de segmentos de tamaño máximo, M, donde $M \neq B$, porque el último bloque en un buffer puede ser pequeño.

3.2.8.1 Evitando el conflicto de ventana tonta

Las especificaciones TCP incluyen ahora heurística que evita el síndrome de ventana tonta. Una heurística usada en la máquina que envía, evita que se transmita una pequeña cantidad de datos en cada segmento. Otra heurística usada en la máquina receptora evita que se envíen pequeños incrementos en los anuncios de las ventanas que pueden disparar paquetes pequeños. Aunque las heurísticas trabajan bien juntas, hacer que tanto el emisor como el receptor eviten las *ventanas tontas* ayuda a asegurar un buen desempeño en el caso que uno de los extremos de una conexión falle en la implementación correcta para evitar la *ventana tonta*.

En la práctica, el software TCP debe contener tanto el código de prevención de *ventana tonta* del emisor como del receptor. Para comprender la razón, cabe recordar que una conexión TCP es dúplex completa, los datos pueden fluir en cualquiera de las dos direcciones. Así, una implementación de TCP incluye código para enviar datos tanto como para recibirlos.

3.2.8.2 Prevención de ventana tonta en el lado del receptor

La heurística que usa un receptor para evitar la ventana tonta es directa y fácil de comprender. En general, un receptor mantiene un registro interno de la ventana actualmente disponible, pero retarda el dar a conocer al remitente un aumento en el tamaño de la ventana hasta que la ventana pueda avanzar en una cantidad significativa. La definición de “significativa” depende del tamaño del buffer del receptor y del tamaño máximo del segmento. TCP lo define como mínimo de la mitad del tamaño del buffer del receptor o el número de octetos de datos en un segmento de tamaño máximo.

La ventana tonta en el lado del receptor evita anuncios de ventana pequeña en el caso donde una aplicación receptora extrae octetos de datos lentamente. Por ejemplo, cuando el buffer del receptor está totalmente lleno, envía un acuse de recibo que contiene un aviso de ventana cero. A medida que la aplicación receptora extrae octetos del buffer, el TCP receptor calcula el nuevo espacio disponible en el buffer. En lugar de enviar de inmediato un aviso de ventana, sin embargo, el receptor espera hasta que el espacio disponible alcanza la mitad del tamaño total del buffer o un segmento de tamaño máximo. Así, el

remitente siempre recibe grandes incrementos en la ventana actual, permitiéndole transferir grandes segmentos.

3.2.9 Acuses de recibo retardados

Dos enfoques se asumen para realizar la implementación de la prevención de ventana tonta en el lado del receptor. En el primer enfoque, TCP hace un acuse de recibo para cada segmento que llega pero no avisa de un aumento en su ventana hasta que la ventan alcanza el límite especificado por la heurística de prevención de ventana tonta. En el segundo enfoque, TCP retarda el envío de un acuse de recibo cuando la prevención de ventana tonta especifica que la ventana no es suficientemente grande para que pueda ser anunciada. Los estándares recomiendan retardar los acuses de recibo.

Los acuses de recibo retardados tienen tanto ventajas como desventajas. La principal ventaja se produce porque los acuses de recibo retardados pueden disminuir el tráfico y por lo tanto aumentar el rendimiento total. Por ejemplo, si llegan datos adicionales durante el período de retardo, un solo acuse de recibo acusará recibo para todos los datos recibidos. Si la aplicación que recibe genera una respuesta inmediatamente después que llegan los datos (por ejemplo, eco de un carácter en una sesión de acceso remoto), un corto retardo puede permitir que los acuses de recibo sean transportados sobre un segmento de datos. Además, TCP no puede mover su ventana hasta que la aplicación receptora extraiga los datos del buffer. En casos donde la aplicación receptora lee los datos tan pronto como llegan, un pequeño retardo permite que TCP envíe un solo segmento de datos que acuse recibo de los datos y avise de una ventana actualizada. Sin acuses de recibo retardados, TCP dará acuse de recibo del arribo de los datos inmediatamente, y después enviará un acuse de recibo adicional para actualizar el tamaño de la ventana.

Las desventajas de los acuse de recibo retardados deben ser claras. Lo más importante, si un receptor retarda sus acuses de recibo demasiado tiempo, el TCP que envía retransmitirá el segmento. Una retransmisión innecesaria disminuye el rendimiento total porque desperdicia ancho de banda de red. Además, la retransmisión necesita cálculos que desperdician ancho de banda en las máquinas receptora y transmisora. Más aún, TCP usa los acuses de recibo que llegan para estimar los tiempos de ida y vuelta: los acuses de

recibo retardados pueden confundir la estimación y hacer los tiempos de retransmisión demasiado prolongados.

Para evitar los problemas potenciales, la norma TCP coloca un límite en el tiempo en que retarda TCP un acuse de recibo. Las implementaciones no deben retardar un acuse de recibo por más de 500 milisegundos. Además, para garantizar que TCP recibe un número suficiente de estimados de ida y vuelta, la norma recomienda que un receptor debe acusar recibo al menos cada dos segmentos de datos.

Si una aplicación genera datos de a un octeto a la vez, TCP enviará el primer octeto inmediatamente. Sin embargo, hasta que llegue el ACK, TCP acumulará los octetos adicionales en su buffer. Así, si la aplicación es razonablemente veloz comparada con la red (i.e., una transferencia de archivo), los segmentos sucesivos contendrán muchos octetos. Si la aplicación es lenta comparada con la red (por ejemplo, un usuario que teclea en un teclado), se enviarán segmentos pequeños sin un retardo largo.

Conocido como el *algoritmo Tagle* en honor de su inventor, la técnica es especialmente elegante ya que requiere pocos cálculos que desperdician ancho de banda. Un host no necesita mantener temporizadores separados para cada conexión, ni tampoco necesita examinar un reloj cuando una aplicación genera datos. Más importante, aunque la técnica se adapta a combinaciones arbitrarias de retardo de red, tamaño máximo del segmento, y la velocidad de la aplicación, no se disminuye el rendimiento total en casos convencionales.

Para comprender por qué el rendimiento total permanece alto para la comunicación convencional, se debe observar que las aplicaciones optimizadas para un alto rendimiento total no generan datos de un octeto a la vez (el hacerlo ocasionaría en un innecesario desperdicio de ancho de banda del sistema operativo) en su lugar, tales aplicaciones escriben grandes bloques de datos a cada llamada. Así, el buffer TCP que se procesa empieza con datos suficientes de al menos un segmento máximo de tamaño. Además, ya que la aplicación produce datos de forma más rápida de la que TCP puede transferir los datos, el buffer de envío permanece casi lleno, y TCP no retarda la transmisión, en consecuencia, TCP continúa enviando segmentos a cualquiera velocidad que Internet pueda tolerar, mientras la aplicación continúa llenando el buffer.

3.3 Métricas para los experimentos simulados en NS-2

Los experimentos que se van a llevar a cabo dentro de esta investigación con el objeto de analizar los diferentes mecanismos de optimización del protocolo TCP se realizaran utilizando el simulador NS-2 con el cual se generaran graficas del comportamiento de los diferentes tipos de paquetes y con estas se sacaran algunas estadísticas (tablas de tiempos, número de paquetes, etc.) con las cuales se medirá el rendimiento de las modificaciones que se le haga a cada uno de los mecanismos que se eligió para optimizar.

Las simulaciones son generadas dentro de escenarios de transmisión los cuales son un conjunto de parámetros y condiciones que se establecen de forma estática (dentro del código TCL del NS-2) para que respondan de acuerdo al mecanismo a analizar.

Para cada uno de los mecanismos que se eligió modificar existe una serie de parámetros que se analizaran con las graficas que se capturan de NS-2, a continuación se mostrará cada uno de los mecanismos a optimizar con una tabla en la cual se establecen las métricas y parámetros.

3.3.1 Tamaño De La Ventana Inicial De Transmisión

El escenario que se establece para esta simulación es la de dos nodos (terminales) transmitiendo diferentes tipos de trafico (FTP, trafico WEB) a través del protocolo TCP.

Las modificaciones del tamaño de la ventana de transmisión se hacen iniciando con el tamaño de ventana igual a 1 segmento, el cual es utilizado en muchas de las implementaciones de TCP. Se plantea la posibilidad de analizar las modificaciones que fueron propuestas por el grupo de investigación de una de las empresas de telecomunicaciones más grandes de Japón, NTTDoCoMo, el cual propuso dejar una ventana de transmisión de 64K para su tecnología de transmisión WCDMA.

Parámetro a medir	Metodología
Tiempo para llegar al estado estable (segundos)	Se analiza en las graficas de paquetes transmitidos vs. Tiempo, en qué momento inicia la transmisión estable de los paquetes.

Tabla 3. 1

3.3.2 Umbral del número de dupacks (DUPLICATE ACKS) en la respuesta a la congestión de TCP.

Para esta simulación se establece un escenario igual que para las pruebas de la ventana de inicio de transmisión, pero las modificaciones que se hacen es aumentando a partir de 1 el número de DUPACKs con los cuales se activa el mecanismo de reenvío de paquetes.

Parámetro a medir	Metodología
Tiempo de activación del mecanismo de retransmisión (segundos)	Se analiza las graficas de paquetes transmitidos vs. Tiempo, y se toma el tiempo en el que se activa la retransmisión estable de los paquetes.

Tabla 3. 2

3.3.3 Mecanismos Gestores De Encolamiento Activo Active Queue Management (AQM)

Con esta simulación se pretende analizar el comportamiento del tamaño del *buffer* de recepción y la medida del promedio del tamaño de la cola del *buffer*, esto con el fin de prever el posible comportamiento de rechazo de paquetes en el receptor, por lo tanto se medirá el numero de paquetes rechazados por el receptor antes y después de la utilización del mecanismo de gestión de encolamiento RED.

3.3.4 Habilitación De La Opción De Acuses De Recibo Selectivos (Selective ACK – SACK)

En este escenario se presenta el comportamiento de SACK cuando se utiliza un tráfico FTP y un tráfico con distribución exponencial de paquetes de tamaño fijo; se prueba el comportamiento del protocolo cuando se presenta una transmisión de datos con máximo tamaño de la ventana (opción maxburst_ en el simulador)
Para medir los resultados de la habilitación de esta opción se analiza el número de paquetes retransmitidos, con lo cual se comprueba si la modificación del protocolo es eficiente.

3.3.5 Modificación del tamaño del paquete TCP (Maximum Size Segment MSS) debido a la compresión de cabecera.

Para evaluar el mecanismo de compresión de cabecera dentro de las pruebas experimentales se utiliza el indicador “Cantidad de información útil transmitida”, cuyo atributo es la cantidad de información útil transmitida por cada MTU (Maximum Transmission Unit, máxima unidad de transmisión) en el envío de información a través de la interfaz aérea, la unidad de medida será “bytes / MTU” y la unidad operacional será (# Actual de bytes / MTU) / (# Promedio de bytes / MTU):

$$Unidad\_Operacional = \frac{\#actual\_bytes/MTU}{\#promedio\_bytes/MTU}$$

Con este indicador podremos darnos cuenta con exactitud en que medida se puede aumentar la cantidad de información útil por MTU en la transferencia de datos.

Del mismo modo podemos analizar las mejoras en la eficiencia de acuerdo a la evaluación de cada uno de los mecanismos y parámetros de optimización adecuados al protocolo TCP inalámbrico.

CAPITULO IV

4. EVALUACION DE LOS MECANISMOS DE OPTIMIZACION

En este capitulo se analizan cada uno de los mecanismos propuestos para la modificación del protocolo TCP y se hacen pruebas a través del simulador [NS-2].

En el Capitulo II MECANISMOS ALTERNATIVOS DE OPTIMIZACION DEL PERFIL TCP, se muestra teóricamente los mecanismos que son más apropiados para una optimización del comportamiento del protocolo TCP para su utilización en comunicaciones inalámbricas y sobre todo para su utilización en el conjunto de protocolos WAP.
Los mecanismos que se plantean para su modificación son:

- Mecanismos
    - Algoritmo de determinación de la ventana inicial de transmisión
    - Mecanismos de gestión de cola activos.
    - Acuses de recibo selectivos (SACK).
    - Mecanismo de compresión de la información de cabecera (Modificación de MSS)
    - Modificación del umbral de DUPACKs para la activación de la retransmisión de paquetes perdidos.

A continuación se hace una breve descripción de cada una de las formas de modificación que se hace a cada mecanismo (algoritmo) para hacer su simulación en NS-2. La simulación de cada uno de los algoritmos se hace a través de graficas de tiempo contra la variable a estudiar.

Cada simulación muestra el comportamiento de los paquetes en una línea de tiempo. Para las simulaciones se crean “escenarios de simulación” los cuales son un conjunto de parámetros de la transmisión de datos entre los cuales se encuentran, los nodos de transmisión, las conexiones entre los nodos, el tipo de protocolo que se utiliza en la transmisión de los datos (TCP Reno, TCP Vegas, TCP New Reno,), las aplicaciones que corren dentro de cada nodo (simulaciones de aplicaciones), modelos de perdidas, probabilidades de presentarse errores en la comunicación entre los nodos (perdidas de datos), el tipo de perdidas (por paquetes o perdidas son mayores de datos – ventanas completas), mecanismos de encolamiento (*queue*

managment), retardos, etc. Estos escenarios no son iguales para todas las simulaciones y dependen del mecanismo y algoritmos a modificar.

Debido a que la investigación se centra en la implementación de TCP para la utilización dentro de los protocolos WAP, se utilizará un escenario típico utilizado en dispositivos móviles, en los cuales se utiliza mucho la descargas de archivos a través de FTP y de datos a través de trafico WEB (distribución exponencial).

La forma como se simulan tanto terminales, conexiones y aplicaciones de la vida real en NS-2 es la siguiente, cada terminal se simula como un nodo, al cual se anexa un agente de transporte que simula el protocolo de transporte que se utilizará, y a este se anexa un simulador de aplicación o un generador de trafico. A continuación se muestra la forma como NS-2 hace la conexión de los elementos en una transmisión de datos.

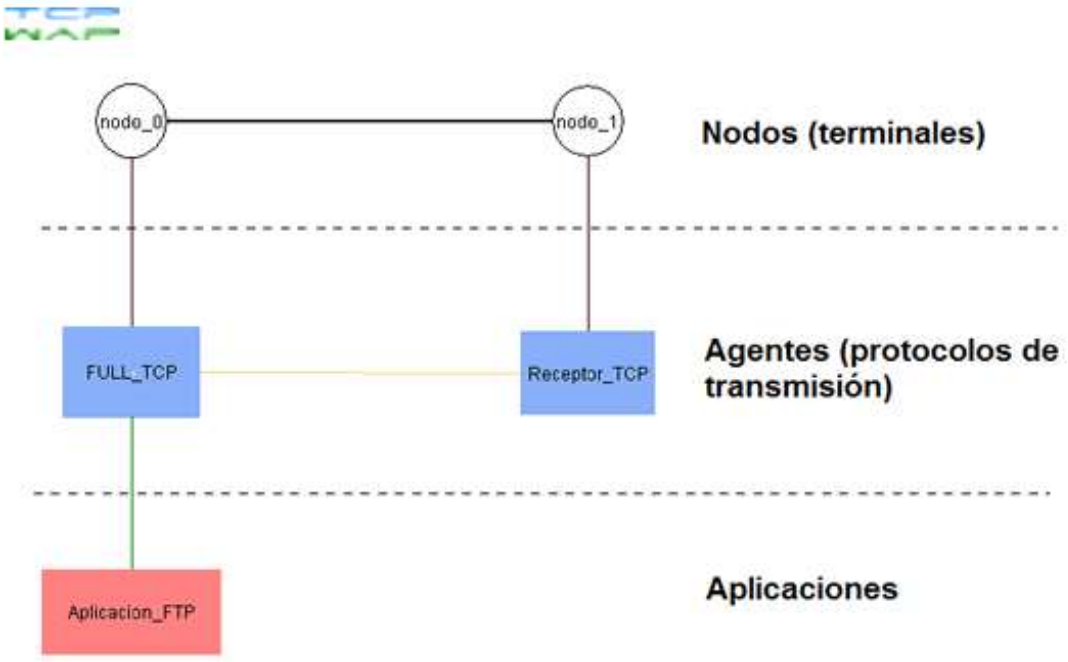


Figura 4. 1. Elementos de una conexión, Nodos, Agentes, Aplicaciones.

A continuación se muestran todos los escenarios para los experimentos de los mecanismos de optimización; cada escenario tiene experimentos diferentes con parámetros específicos para

cada prueba. En el Anexo 2 Experimentos adicionales de los Escenarios de Simulación, están todas las simulaciones que se realizaron para cada escenario. A continuación se muestra de forma resumida las pruebas para cada escenario.

4.1 Escenario 1: análisis del incremento del tamaño de la ventana inicial de transmisión de TCP.

Estas pruebas de la modificación de la ventana inicial son modeladas utilizando NS-2. Los códigos de estas simulaciones se encuentran en el Capitulo Anexo 1 Códigos de las Simulaciones.

Algunas de las pruebas que se hacen a continuación son las de la modificación del mecanismo de *Handshake* que se hace al inicio del establecimiento de la comunicación, en la cual se envían paquetes SYN.

4.1.1 Experimento 1: Conexión FTP/TCP FULL con ventana inicial de 1 segmento

Características Principales del Enlace	
Nodos	2
Protocolo	TCP FULL[3]
Aplicación	FTP
Ventana Inicial	1
Tiempo de Simulación	15 segundos

Tabla 4. 1. Características del Experimento 1

[3] NS-2 soporta varios tipos de agentes TCP para su simulación: TCP Tahoe, TCP Reno, TCP NewReno, y algunos agentes los cuales tienen modificaciones experimentales. TCP FULL es una de esas modificaciones la cual genera tráfico en las dos direcciones, paquetes enviados por el emisor y ACKs enviados por el receptor.

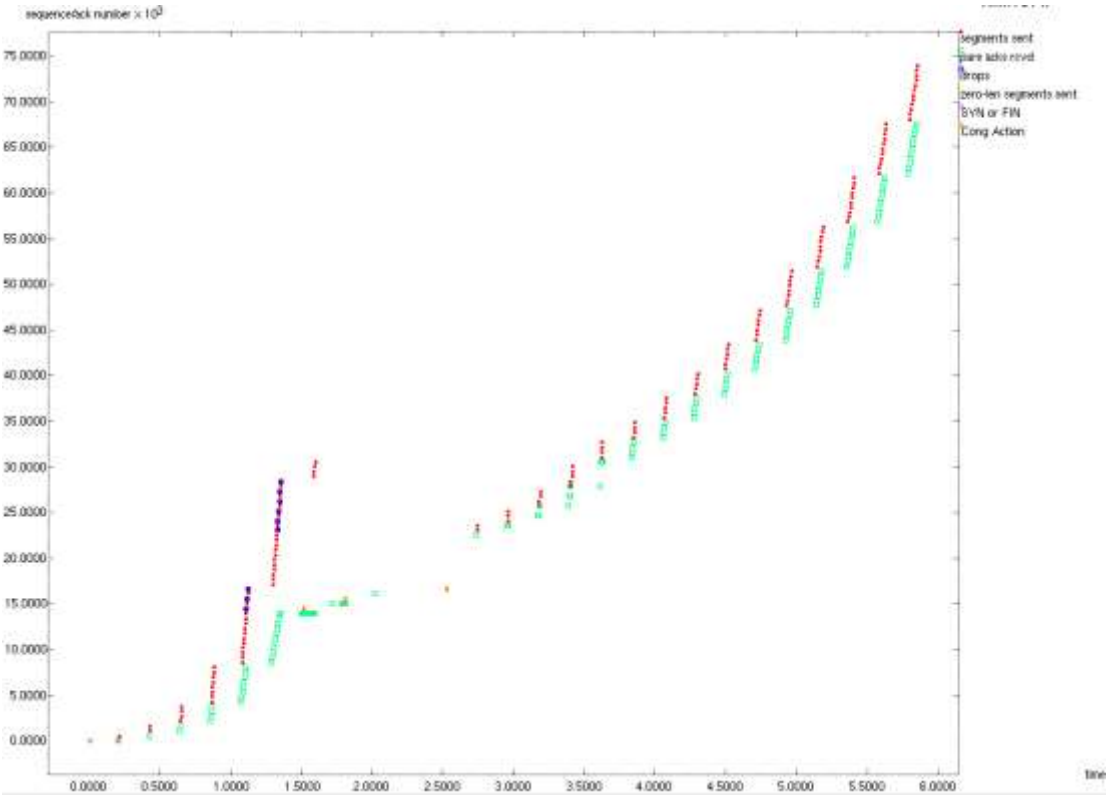


Figura 4. 2. Conexión FTP/TCP FULL con ventana inicial de 1 segmento.

En la figura 4.2, se muestra la simulación de una comunicación entre dos nodos a través de una aplicación FTP, utilizando el protocolo TCP/FULL. En esta simulación se presenta el caso (común en muchas implementaciones de TCP del Kernel de Linux antiguas) en el cual la ventana de inicio de la transmisión inicia con 1 segmento.

En la simulación se presenta la primera perdida de datos (puntos de color violeta) debido a congestión del *buffer* de salida a los 1.1150 segundos, debido al algoritmo de crecimiento exponencial de TCP; la mayor congestión se presenta en los 1.3586 segundos en los cuales se satura la transmisión dando como resultado una clara indicación (puntos de color naranja) de crecimiento indebido de la ventana de transmisión, lo cual hace activar el algoritmo para disminuir la ventana de transmisión, haciendo que se incremente linealmente (no exponencial) el tamaño de la ventana hasta llegar a un tamaño equivalente al tamaño de la ventana soportado por la red o igual al tamaño del *buffer* de recepción dependiendo cual de estos es menor.

4.1.2 Experimento 2: Conexión FTP/TCP FULL con ventana inicial de 4 segmentos

Características Principales del Enlace	
Nodos	2
Protocolo	TCP FULL
Aplicación	FTP
Ventana Inicial	4
Tiempo de Simulación	15 segundos

Tabla 4. 2. Características del Experimento 2

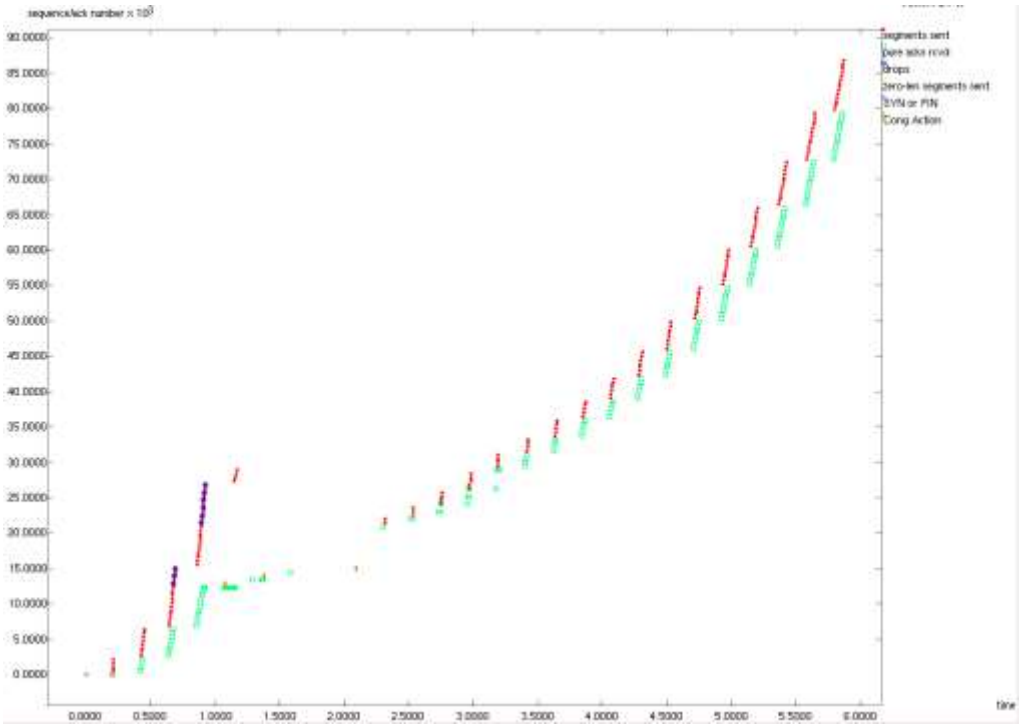


Figura 4. 3. Detalle de los primeros 6 segundos. Experimento 2

Como podemos observar, el aumento en el tamaño de la ventana inicial permite que se llegue a un estado estable de la transmisión mucho más rápido. A continuación se muestra una tabla en la cual se comparan las anteriores pruebas tomando como base el tiempo en el cual llegan a un estado estable de la comunicación. Los experimentos adicionales de este escenario se encuentran en el Anexo 2 de este trabajo.

Tamaño de la ventana inicial (segmentos)	**Tiempo para llegar al estado estable (segundos)**
1	2.7500
2	2.6300
4	2.3150
8	1.8500
64	1.4300

Tabla 4. 3. Comparación de los tamaños de la ventana inicial y su respuesta temporal para llegar al estado estable.

En la tabla 4.3 se muestra el resultado del experimento utilizando una ventana inicial de 64 segmentos en la cual se ve que el tiempo de estabilización de la transmisión es muy corto comparado con los demás. Esto podría a primera vista ser muy apropiado en un ambiente inalámbrico en el cual se necesita que los recursos no se desperdicien, pero como se muestra en la figura siguiente (Figura 4.4) existe un grave problema debido al tamaño del *buffer* ya que se presentan varias pérdidas debido al limitado tamaño de salida de los datos desde el transmisor.

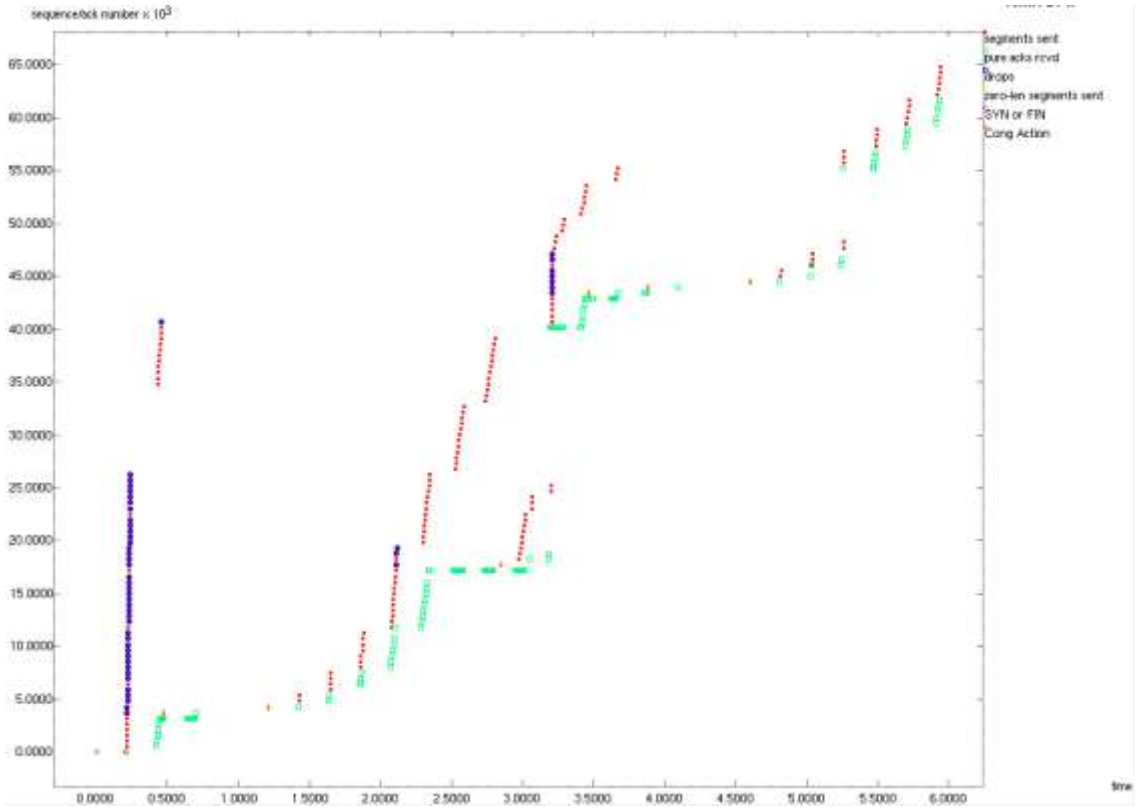


Figura 4. 4. Conexión FTP/TCP FULL con ventana inicial de 64 segmentos

La utilización de un tamaño de ventana inicial mayor a 1 segmento fue propuesta por [HI,TI 00] pertenecientes a NTT DoCoMo, Inc. de Japón, una de las empresas mas grandes en telecomunicaciones y líder en el desarrollo de tecnologías de transmisión para dispositivos móviles.

Aunque esta propuesta fue hecha en el año 2000 como un *draft de Internet* (draft-inamura-docomo-00.txt) *A TCP profile for W-CDMA: 3G wireless packet service*, se tiene como una propuesta experimental.

La propuesta de [HI,TI 00] esta dirigida a la tecnología de comunicación de tercera generación (3G) W-CDMA, la cual maneja anchos de banda superiores comparados con los que se maneja actualmente en nuestro país, pero a pesar de que la propuesta esta referenciada a un marco 3G, se pueden aplicar estos mismos principios para mejorar el comportamiento del protocolo en redes inalámbricas 2G y 2.5G.

[HI,TI 00] proponen que un incremento de la ventana inicial (*Init Window - IW*) es efectiva especialmente para la transmisión de datos pequeños, lo cual es comúnmente visto en aplicaciones de Internet para servicios de telefonía móvil. En ese estudio se ve también que para transferencias de tamaños grandes el efecto de este mecanismo es despreciable. De acuerdo a esas simulaciones, el incremento de la ventana inicial mejora el tiempo de la transferencia de 100 KB en 2 segundos, reduciendo el tiempo total de transmisión de 9 a 7 segundos.

4.2 Escenario 2: análisis de la modificación del umbral del número de DUPACKS (duplicate ACKs) en la respuesta a la congestión de TCP.

El protocolo TCP (Tahoe TCP) asume que un paquete se ha perdido (debido a congestión) cuando este observa un número determinado de DUPACKs (*duplicate ACKs*) o cuando un temporizador de retransmisión expira. En la mayoría de implementaciones TCP dentro del Kernel de Linux y en la mayoría de simuladores el número de DUPACKs, NUMDUPACKS (NUMDUPACKS está definido en NS-2 en tcp.h) y su valor por defecto es 3. Ya sea que el temporizador expira o el número de DUPACKS llegue a su umbral 3, TCP reacciona fijando la variable ssthresh_ (umbral para activar el algoritmo Slow Start) en la mitad del tamaño actual de la ventana (que es el mínimo entre las variables cwnd_ y window). Luego TCP inicializa cwnd_ de nuevo en el valor de la ventana inicial windowInit_.

Debido a que la investigación busca el mejoramiento de los algoritmos para el funcionamiento óptimo del protocolo, se analizará el aumento y la disminución del umbral al cual se fija la variable DUPACKS con el objetivo de probar si un aumento mejora el comportamiento en ambientes con gran número de perdidas.
El escenario es similar al escenario 1 (anteriormente visto), y contiene dos agentes TCP por los cuales se transmite diferentes tipos de trafico (FTP, trafico con distribución exponencial y trafico con distribución de Pareto) y el medio de transmisión tiene pérdidas similares a las que tienen las redes inalámbricas. El código TCL de la simulación completa se encuentra en el Anexo 1 - Códigos de las Simulaciones.

4.2.1 Experimento 3: Conexión Tráfico con FTP/TCP Tahoe utilizando un valor umbral de 2 para DUPACKS.

Características Principales del Enlace	
Nodos	2
Protocolo	TCP Tahoe
Aplicación	Trafico FTP
Ventana Inicial	2
Ventana de transmisión	50
Umbral DUPACKS	2
Tiempo de Simulación	15 segundos

Tabla 4. 4. Características del Experimento 3

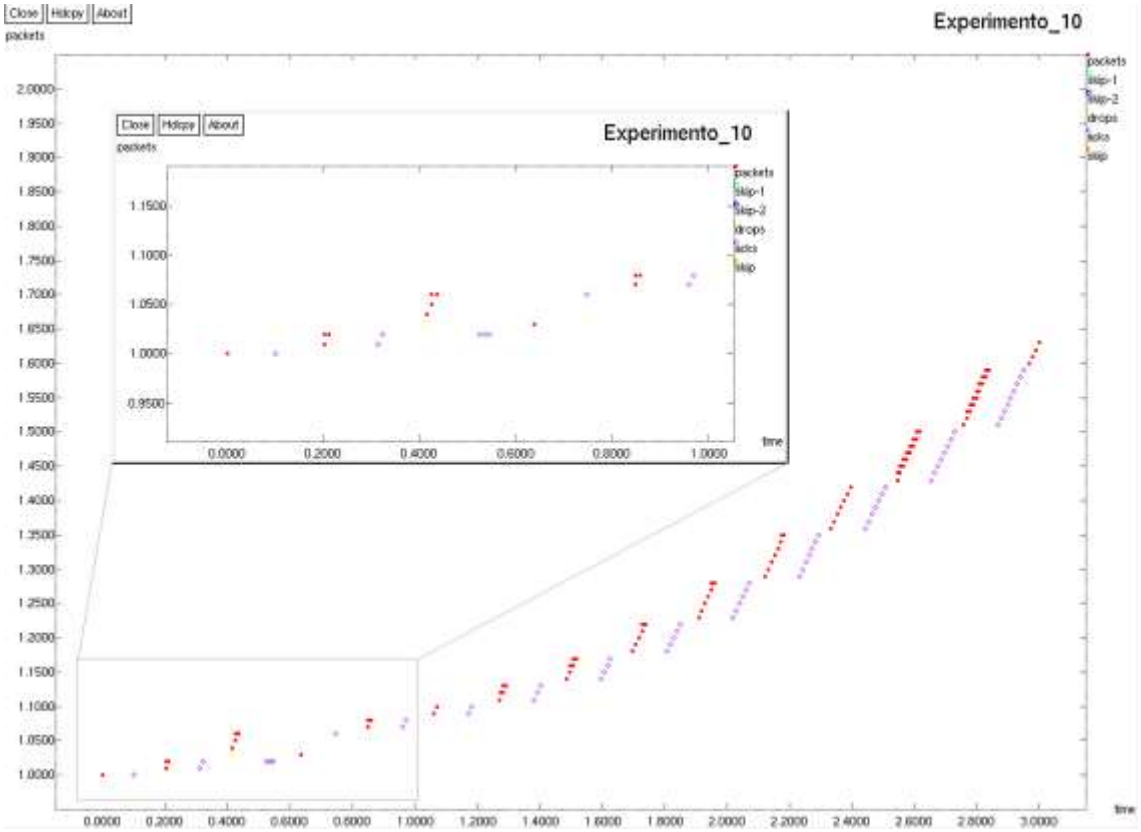


Figura 4. 5. Conexión Tráfico FTP/TCP Tahoe utilizando un valor umbral de 3 para DUPACKS.

A los 637.2000 milisegundos se activa el algoritmo Slow Start como respuesta a la llegada de 2 DUPACKS. Esta respuesta es bastante rápida, ya que el paquete perdido fue detectado a los 413.8000 milisegundos.

En la mayoría de las implementaciones el umbral esta fijado a 3 DUPACKS, a continuación se muestra la simulación con el umbral fijado en 3.

4.2.2 Experimento 4: Conexión Tráfico con FTP/TCP Tahoe utilizando un valor umbral de 3 para DUPACKS.

Características Principales del Enlace	
Nodos	2
Protocolo	TCP Tahoe
Aplicación	Trafico FTP
Ventana Inicial	2
Ventana de transmisión	50
Umbral DUPACKS	3
Tiempo de Simulación	15 segundos

Tabla 4. 5. Características del Experimento 4

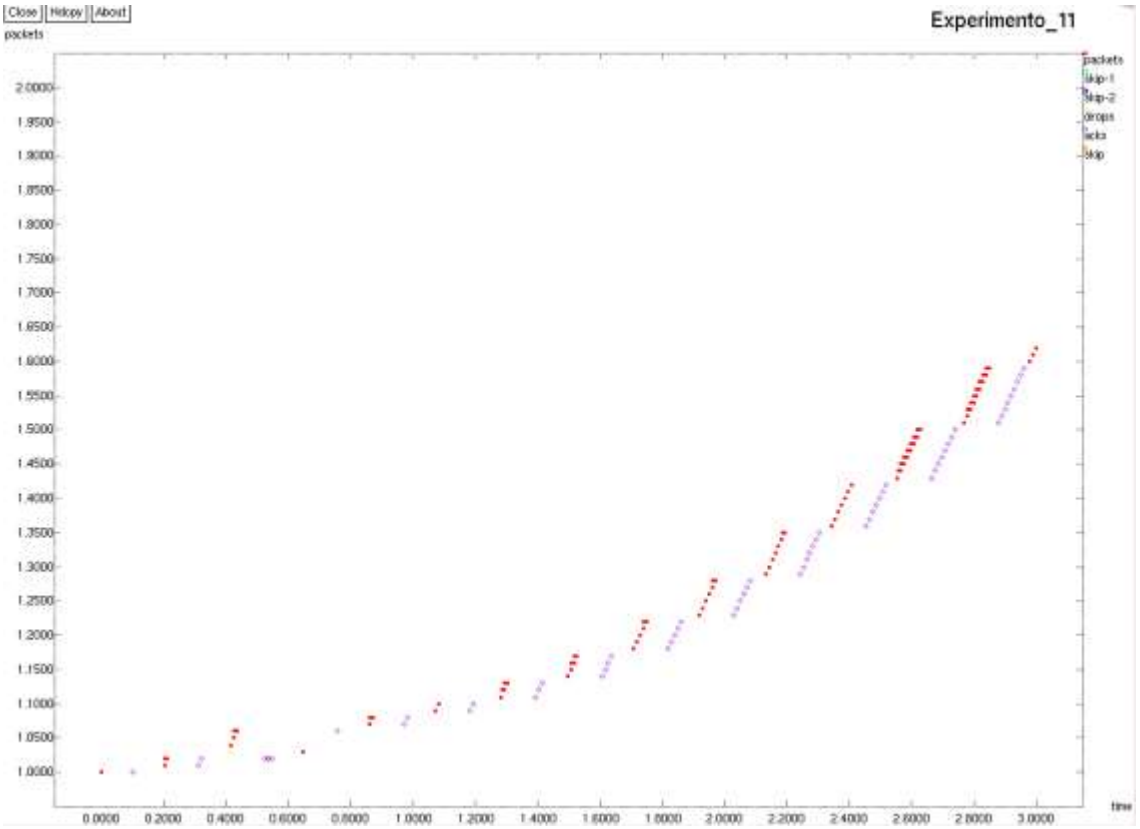


Figura 4. 6. Conexión Tráfico FTP/TCP Tahoe utilizando un valor umbral de 3 para DUPACKS.

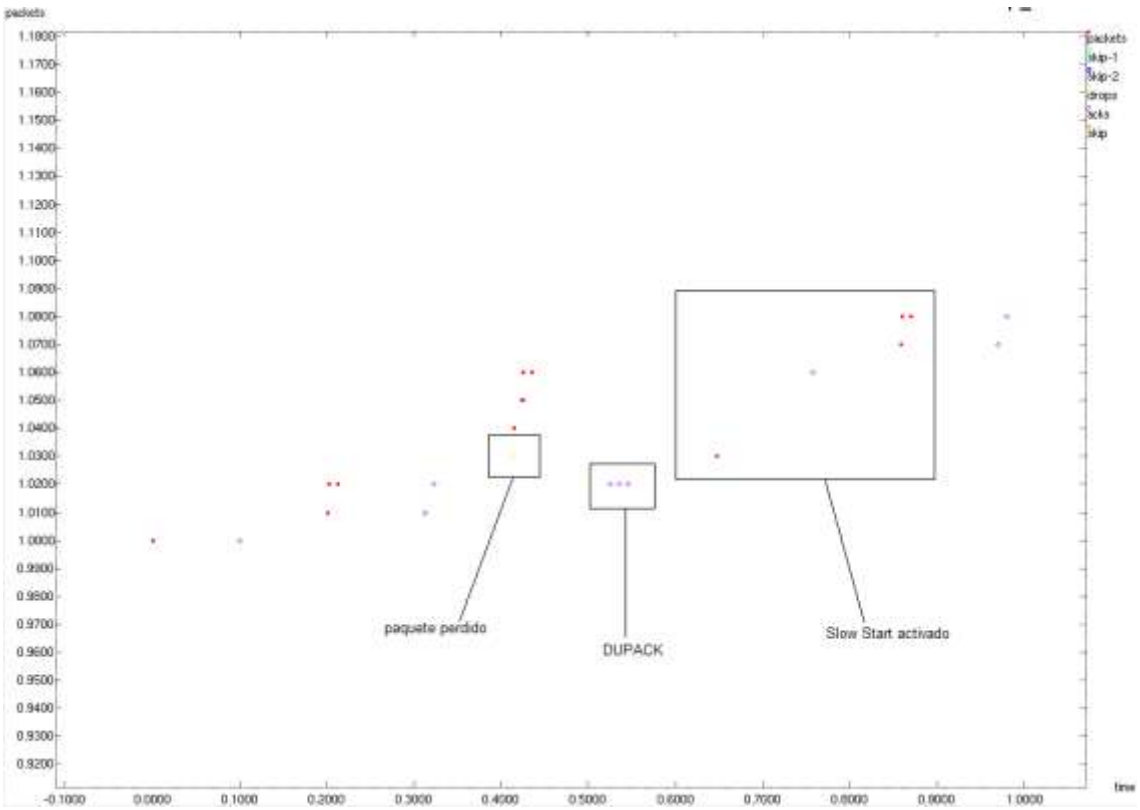


Figura 4. 7. Detalle del primer segundo. Conexión Tráfico FTP/TCP Tahoe utilizando un valor umbral de 3 para DUPACKS.

En la figura 4.7 se puede observar que TCP Tahoe responde a los 647.5200 milisegundos activando el algoritmo Slow Start, disminuyendo la ventana de transmisión y estabilizando la transmisión de paquetes a los 869.8000 milisegundos.

TCP reacciona a la supuesta pérdida de paquetes debida a congestión activando el algoritmo Slow Start, pero en ambientes inalámbricos las pérdidas debidas al medio ambiente son más probables, lo cual afecta el rendimiento de la transmisión.

En la tabla 4.6 se puede ver los valores de activación del algoritmo Slow Start, dado el umbral correspondiente.

Umbral DUPACK	Tiempo de activación del Slow Start ($\times 10^{-3}$ segundos)	Tiempo de pérdida del paquete ($\times 10^{-3}$ segundos)
2	637.2000	413.8000
3	647.5200	
4	1424.2000	

Tabla 4. 6. Valores de activación del algoritmo Slow Start

En el caso del umbral fijado en 4 el inicio de este algoritmo se hace mucho después de que se perdió el paquete, podemos ver que es una buena alternativa el aumentar el valor del umbral ya que la latencia de paquetes en redes WWAN como es el caso de las redes celulares se hace mayor y aun mas considerando las perdidas de paquetes debidas a perdidas en la interfaz aérea del enlace.

4.3 Escenario 3: análisis de la utilización de mecanismos gestores de encolamiento activo Active Queue Management (AQM).

El receptor tiene rara vez conocimiento de las propiedades de la red y de su estado actual. Sin embargo, cuando un terminal sabe que esta conectado al ultimo salto del enlace inalámbrico, este podría limitar la ventana de recepción. El limitar la ventana de recepción también previene excesivo encolamiento (*queueing*) en la red (*overbuffering*), lo cual ocurre cuando el tamaño del *buffer* del router (o gateway) es mucho más grande que el que requiere el enlace.

El algoritmo de detección de errores temprana aleatoria (*random early detection - RED*) fue propuesto como un mecanismo de encolamiento activo Active Queue Management (AQM). RED propone estrategias para cuando los paquetes comienzan a perderse. RED puede ser comparado con el mecanismo de gestión de encolamiento *Tail Drop – TD* empleado por muchos enrutadores de Internet, donde las políticas de descarte de paquetes entrantes esta basado en el desbordamiento (*overflow*) del *buffer* del puerto de salida. Contrario a TD, los mecanismos AQM empiezan a rechazar paquetes en etapas tempranas de la recepción, de acuerdo a su capacidad para notificar a las fuentes de tráfico sobre el incipiente estado de congestión. RED ha sido diseñado para sustituir a TD y es actualmente implementado en algunos enrutadores comerciales. El problema del control de congestión en Internet esta más allá del incremento de la transformación de Internet en una red multiservicio de alta velocidad,

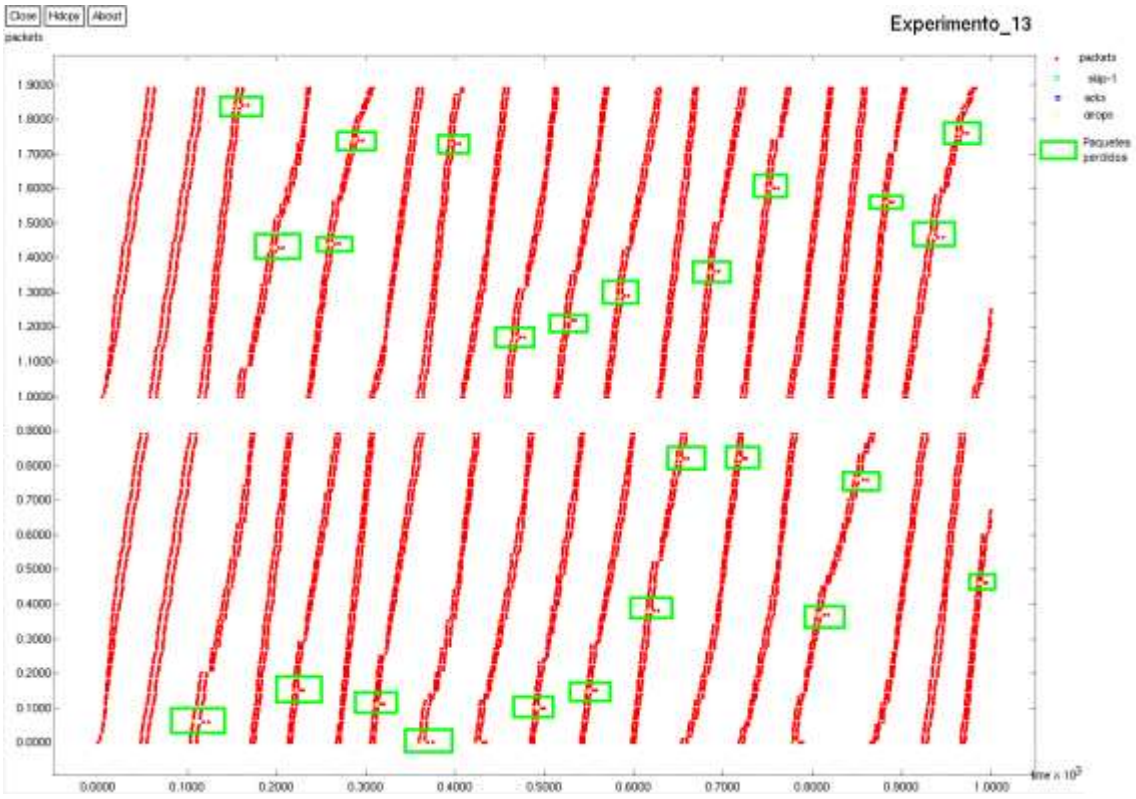
por ejemplo las dos arquitecturas planteadas para dar diferentes servicios dentro de la red: *Integrated Services (Int-Serv)* y *Diff-Serv*. Algunos mecanismos AQM (como RED) han sido propuestos dentro del entorno de la arquitectura *Diff-Serv* de Internet para rechazar paquetes no sin preferencia (priorizados). A parte de RED muchas variantes han sido propuestos tales como, n-RED, adaptive RED [FK, SS 99], RIO [CF98], BLUE[F99].

A continuación se mostrara el comportamiento del mecanismo RED utilizado en una red inalámbrica.

4.3.1 Experimento 5: transmisión de tráfico FTP entre dos nodos, utilizando el método de gestión de cola Drop Tail en el receptor (gateway WAP).

Características Principales del Enlace	
Nodos	2
Protocolo	TCP Reno
Ventana	15 segmentos
Aplicación	Trafico FTP
Tiempo de duración del trafico	4 segundos
Mecanismo de Gestión de Colas	Drop Tail
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb
Limite de paquetes en cola en receptor	128
*Retraso (*delay*) de los paquetes en la interfaz aérea*	200ms

Tabla 4. 7. Características del Experimento 5



Convenciones

	Paquetes
	ACKs
	Paquetes perdidos

Figura 4. 8. Vista del comportamiento de los paquetes en la transmisión de tráfico FTP entre dos nodos, utilizando el método de gestión de cola *Drop Tail* en el receptor (gateway WAP).

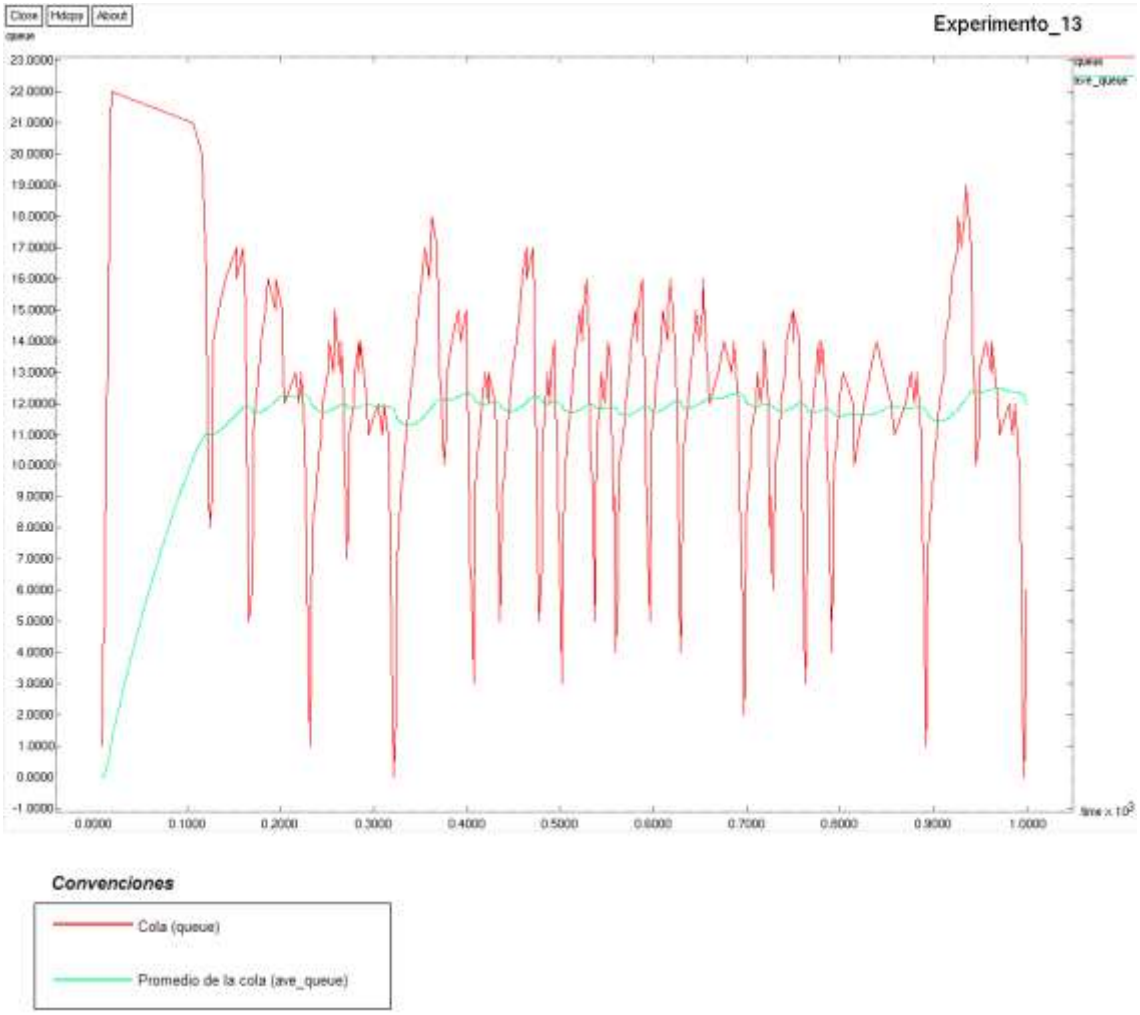


Figura 4. 9. Vista del comportamiento de la cola de paquetes en el receptor del tráfico FTP entre dos nodos, utilizando el método de gestión de cola *Drop Tail* en el receptor (gateway WAP).

La grafica 4.8 es la representación de los paquetes transmitidos en función del tiempo. Se puede ver que existen muchas perdidas, la mayoría por el modelo de perdidas que se estableció en la simulación, el cual se presentaba como un proceso de tiempo aleatorio (ver Anexo 1 para el código del Experimento 5). En la grafica 4.8 podemos darnos cuenta que la transmisión de datos con las condiciones establecidas no presenta problemas en cuanto tiene que ver con el retraso de paquetes debido a latencia (200 ms) aunque ese valor podría incrementarse dependiendo de muchos factores tanto internos como externos de la arquitectura de la red móvil.

En la grafica 4.9 se observa el comportamiento de la cola de paquetes en el Receptor, los valores que se adoptaron para la simulación fueron tomados de dispositivos reales, actualmente utilizados (*GGSN (Gateway GPRS support node) Cisco Systems*) y en transmisiones de datos comunes. La grafica 4.9 muestra que hasta el segundo 18.9815, la cola se llena hasta que el receptor puede manejar los datos dentro de esta, luego la carga de la cola disminuye de manera constante hasta el segundo 106.6740 en la cual disminuye sustancialmente, y el comportamiento del tratamiento de paquetes de la cola se vuelve difícil de predecir teniendo picos negativos y positivos de carga durante el tratamiento de los paquetes de la cola, este comportamiento se efectúa debido a que el algoritmo de encolamiento tiende a mantener en la cola de paquetes un numero dado por el numero promedio que se calcula dentro del receptor (línea verde en la grafica 4.9).

Si comparamos las dos gráficas podemos darnos cuenta que los paquetes perdidos dentro del receptor (figura 4.8) son debidos a picos (en la gráfica 4.9) de paquetes de la cola en el receptor, esto es debido a que el algoritmo de descarte de paquetes en el receptor *Drop Tail* funciona de forma tal que si se llena el *buffer* de recepción los paquetes nuevos que lleguen a la cola serán rechazados.

A continuación se muestra el comportamiento del mismo enlace pero aplicando un método de gestión de colas.

4.3.2 Experimento 6 transmisión de trafico FTP entre dos nodos, utilizando el método de gestión de cola RED (Random Early Detection) en el receptor (gateway WAP).

Características Principales del Enlace	
Nodos	2
Protocolo	TCP Reno
Ventana	15 segmentos
Aplicación	Trafico FTP
Tiempo de duración del trafico	4 segundos
Mecanismo de Gestión de Colas	RED (Random Early Detection)
Tiempo de Simulación	1000 segundos

Ancho de Banda	26Kb
Limite de paquetes en cola en receptor	128
*Retraso (*delay*) de los paquetes en la interfaz aérea*	200ms

Tabla 4. 8. Características del Experimento 6

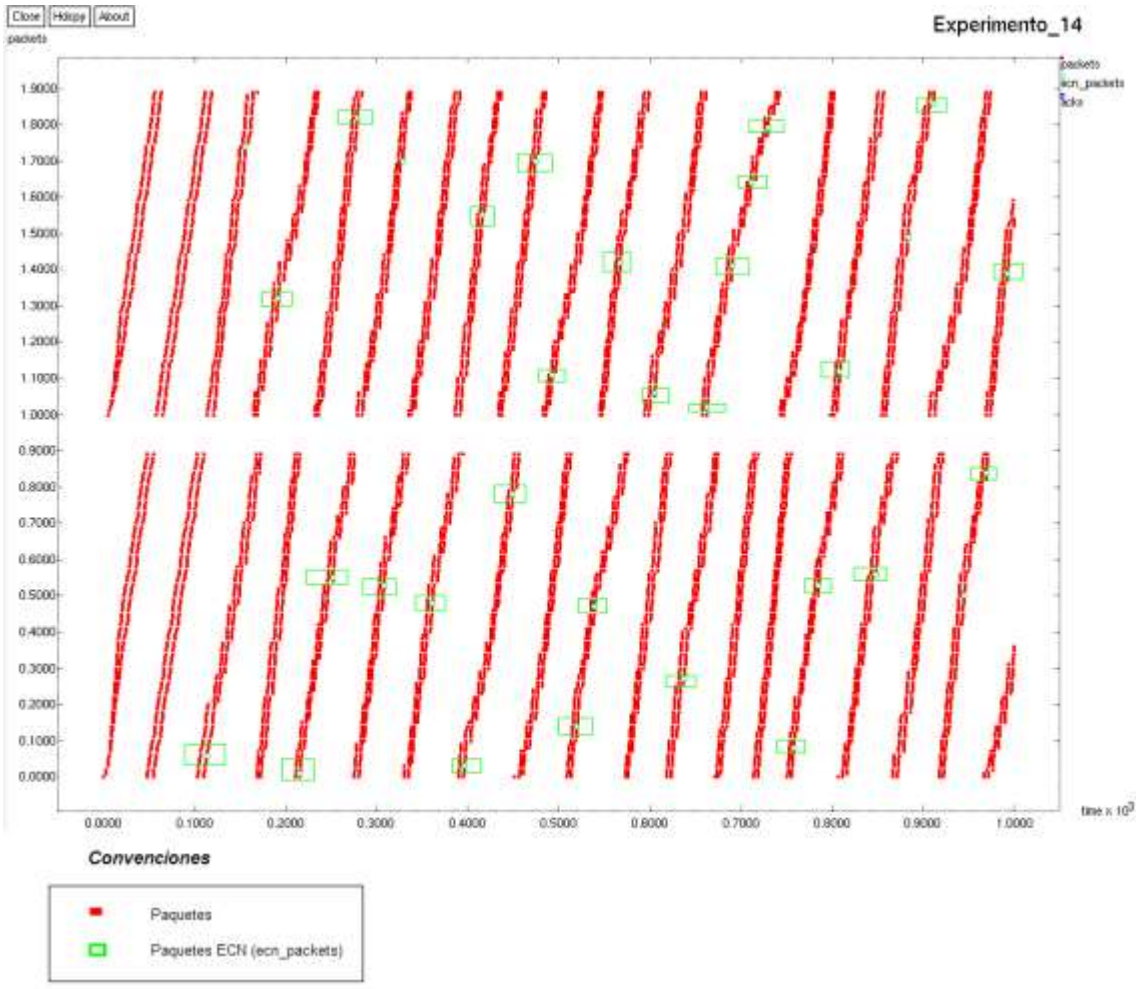


Figura 4. 10. Vista del comportamiento de los paquetes de la transmisión del tráfico FTP entre dos nodos, utilizando el método de gestión de cola *RED (Random Early Detection)* en el receptor (gateway WAP)

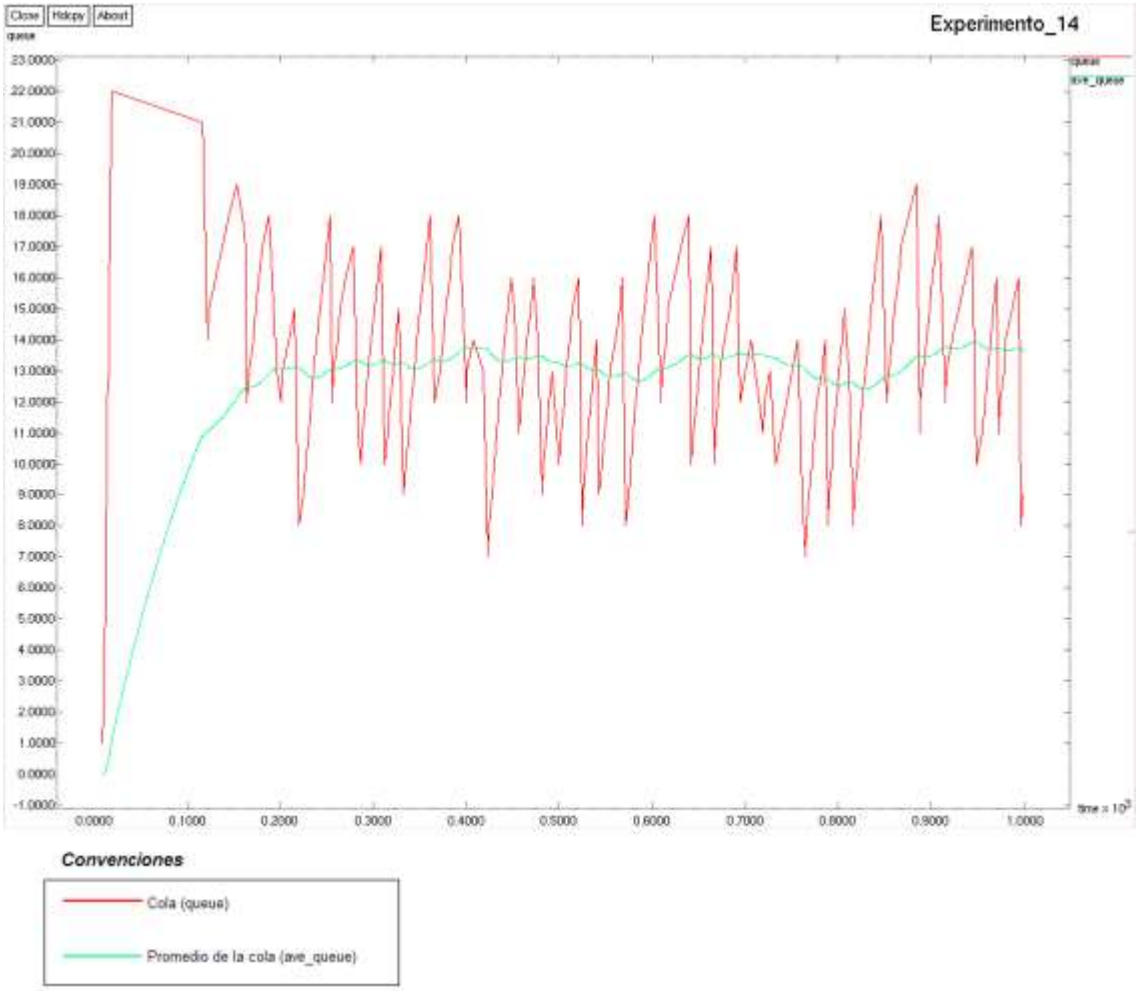


Figura 4. 11. Vista del comportamiento de la cola en la transmisión del trafico FTP entre dos nodos, utilizando el método de gestión de cola *RED (Random Early Detection)* en el receptor (gateway WAP)

En la figura 4.10 se puede ver que en el tráfico normal de paquetes se encuentran unos paquetes con información de congestión (ver paquetes verdes figura 4.10), estos paquetes llevan información desde el receptor hacia el emisor con notificación explicita de congestión (*ECN Explicit Congestion Notification*). ECN es un mecanismos de control de congestión muy utilizado y el cual puede ser implementado de forma sencilla en los dispositivos receptores (Gateways y Enrutadores). El envió de este tipo de paquetes es dado por el comportamiento de la cola en el receptor (figura 4.11) los paquetes son ingresados en la cola y dado que se han enviado notificaciones de congestión tempranas al emisor, éste disminuye la tasa de transmisión haciendo que la cola de recepción no tenga saltos abruptos de saturación como se aprecia en la figura 4.11. En realidad en comparación con el comportamiento de la cola sin un mecanismo de regulación de la cola (figura 4.9) la cola no tiene mayores diferencias a simple

vista, pero si detallamos los picos que se generan en la figura 4.11 estos son menos de los que se presentan en la figura 4.9, esto es porque el algoritmo del mecanismo ECN hace que la cola permanezca en un valor mas o menos constante (valor promedio, líneas color verde figuras 4.9 y 4.11).

4.3.3 Experimento 7 transmisión de trafico FTP entre dos nodos, utilizando el método de gestión de cola RED (Random Early Detection) en el receptor (gateway WAP) modificando los umbrales de encolamiento.

Características Principales del Enlace	
Nodos	2
Protocolo	TCP Reno
Ventana	15 segmentos
Aplicación	Trafico FTP
Tiempo de duración del trafico	4 segundos
Umbral mínimo para la medida promedio de la cola de paquetes	5
Umbral mínimo del tamaño de cola	10
Mecanismo de Gestión de Colas	RED (Random Early Detection)
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb
Limite de paquetes en cola en receptor	128
*Retraso (*delay*) de los paquetes en la interfaz aérea*	200ms

Tabla 4. 9. Características del Experimento 7

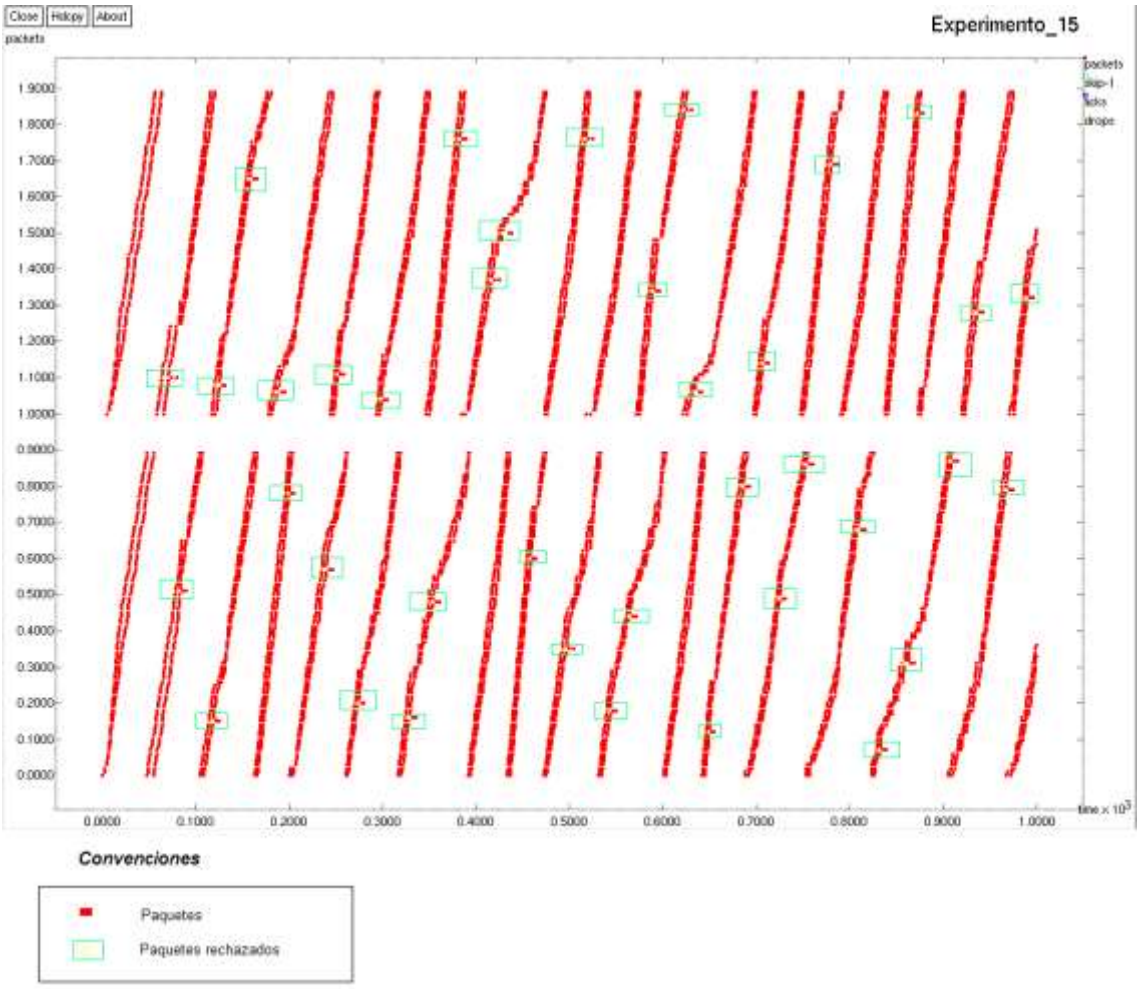


Figura 4. 12. Vista del comportamiento de los paquetes en la transmisión del trafico FTP entre dos nodos, utilizando el método de gestión de cola *RED (Random Early-Detection)* en el receptor (gateway WAP) modificando los umbrales de encolamiento.

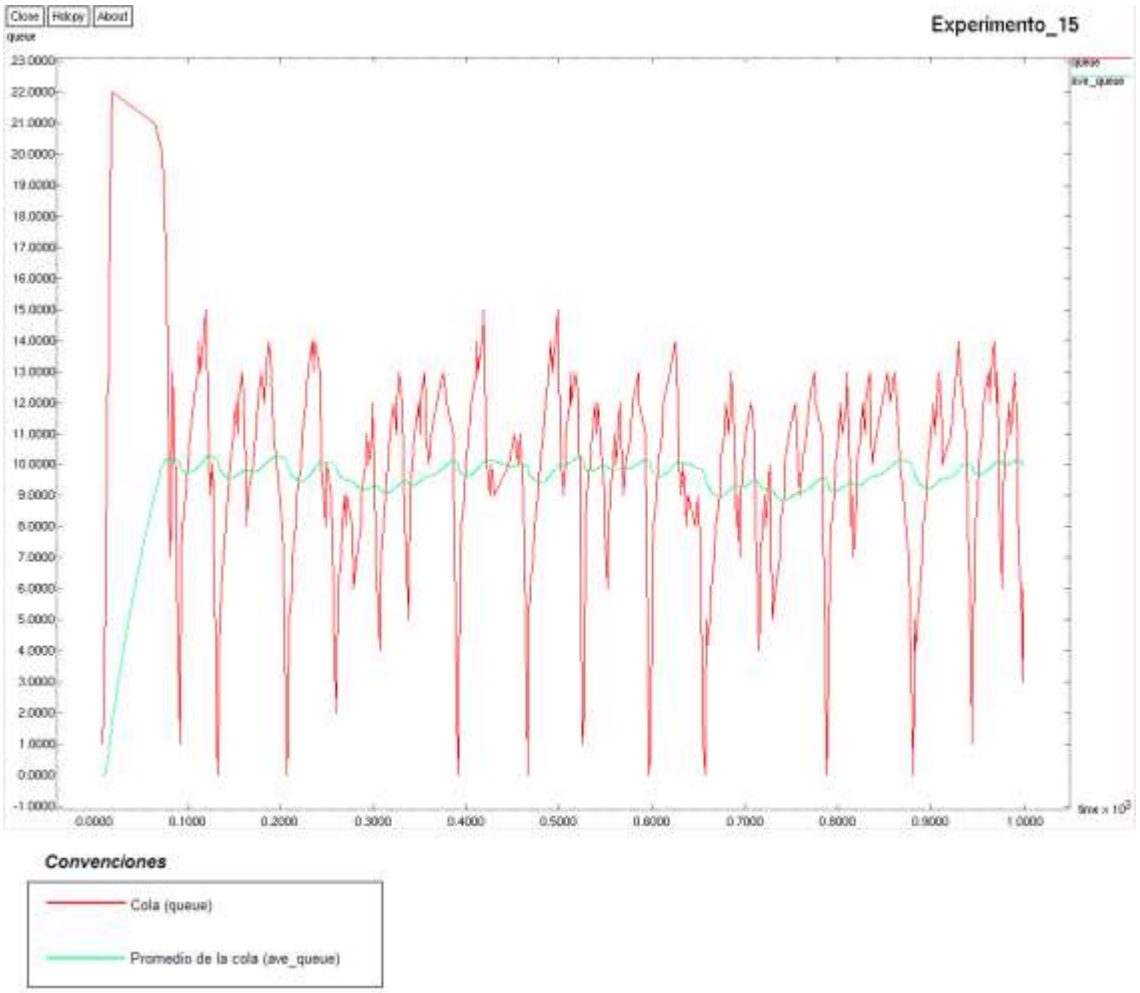


Figura 4. 13. Vista del comportamiento de la cola del buffer de recepción en la transmisión del trafico FTP entre dos nodos, utilizando el método de gestión de cola *RED (Random Early-Detection)* en el receptor (gateway WAP) modificando los umbrales de encolamiento.

El análisis del experimento 7 es similar al del 6, solo que en el primero no se modifican los parámetros por defecto que se tiene el simulador. Con las modificaciones adecuadas se tiene una respuesta mucho mejor del comportamiento de la cola el cual en una caso ideal como es el de este experimento es el de mantener el promedio de la cola (línea verde en la grafica 4.13) mucho mas abajo del umbral de descarte, en el cual los paquetes son descartados. Estas modificaciones son ideales, ya que en casos prácticos en los enrutadores y *gateways* no se puede modificar ciertos valores como es el caso del umbral mínimo del tamaño de cola y el umbral mínimo para la medida promedio de la cola de paquetes.

4.4 Escenario 4: análisis de la habilitación de la opción SACK en transferencias de datos entre dos nodos.

Con la opción SACK habilitada una transmisión de datos podría beneficiarse si el medio de transporte tiene perdidas como es el caso de un enlace inalámbrico. Este escenario analiza la utilización de la opción SACK en trafico FTP así como en trafico con distribución exponencial.

4.4.1 Experimento 8 Transmisión de tráfico FTP entre dos nodos, utilizando la opción SACK.

Características Principales del Enlace	
Nodos	2
Protocolo	TCP Reno
Ventana	14 segmentos
Opción SACK	activada
Aplicación	Trafico FTP
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb

Tabla 4. 10. Características del Experimento 8

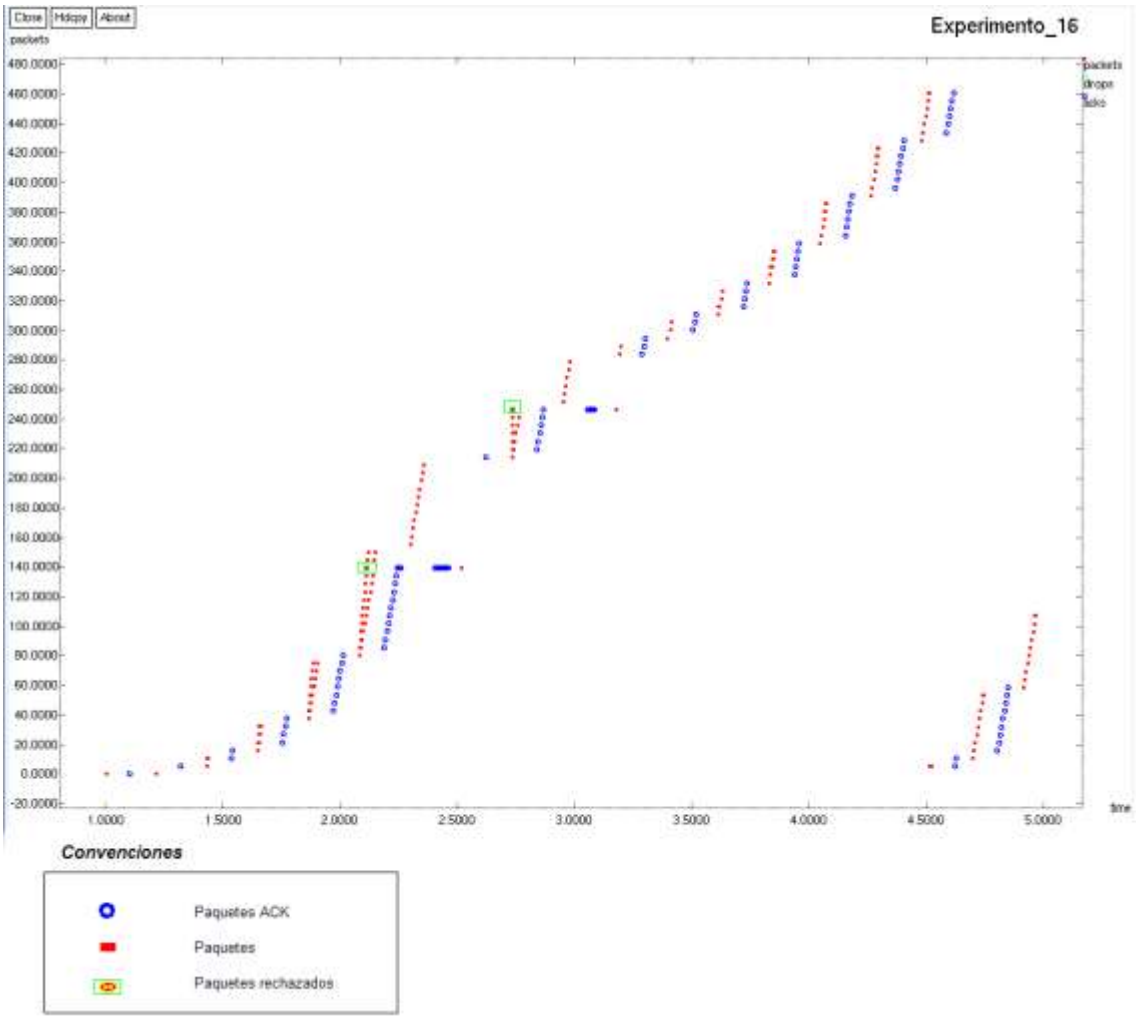


Figura 4. 14. Comportamiento de los paquetes en la transmisión de trafico FTP utilizando dos nodos TCP habilitada la opción SACK.

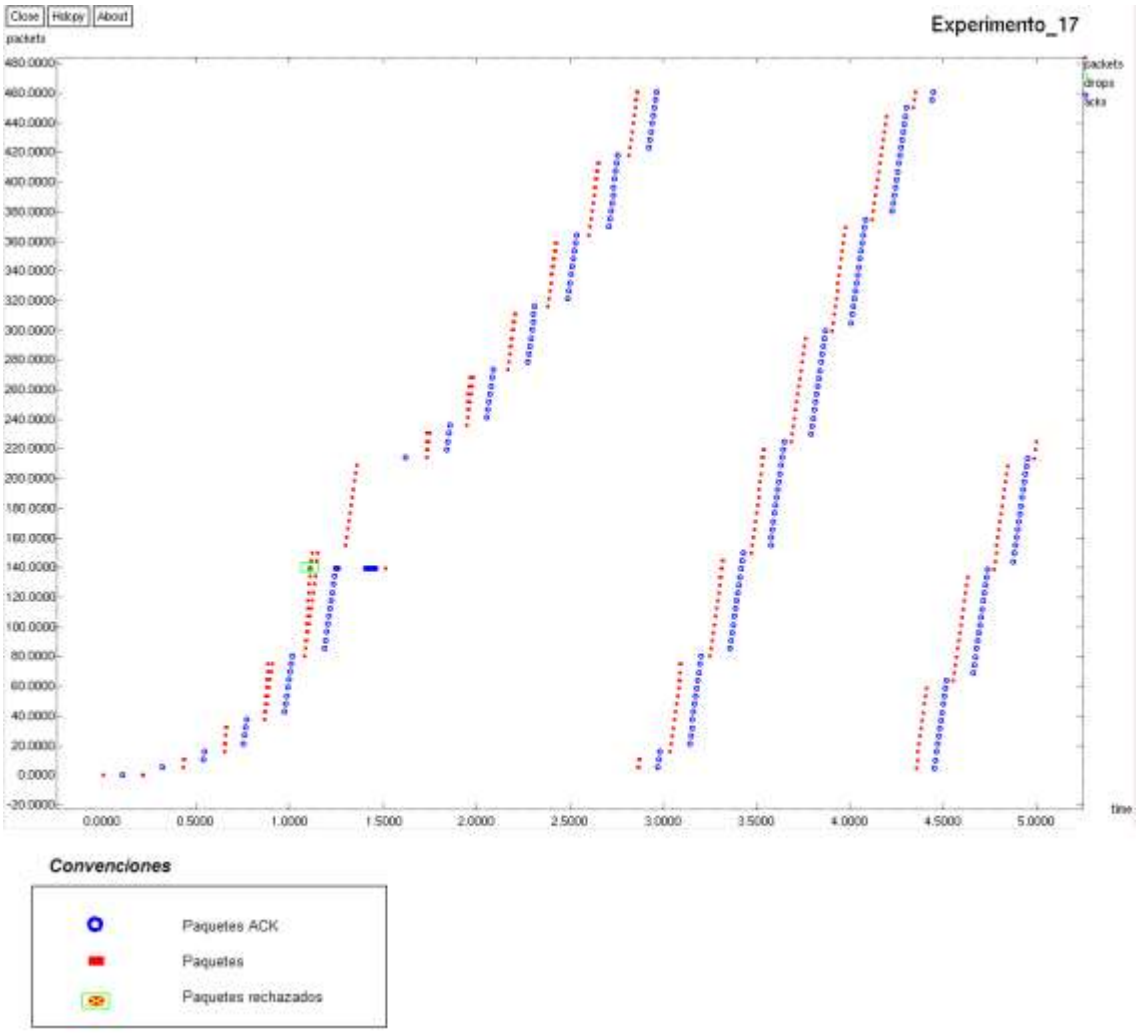


Figura 4. 15. Comportamiento de los paquetes en la transmisión de tráfico FTP utilizando dos nodos TCP habilitada la opción SACK y la opción ***maxburst_***.

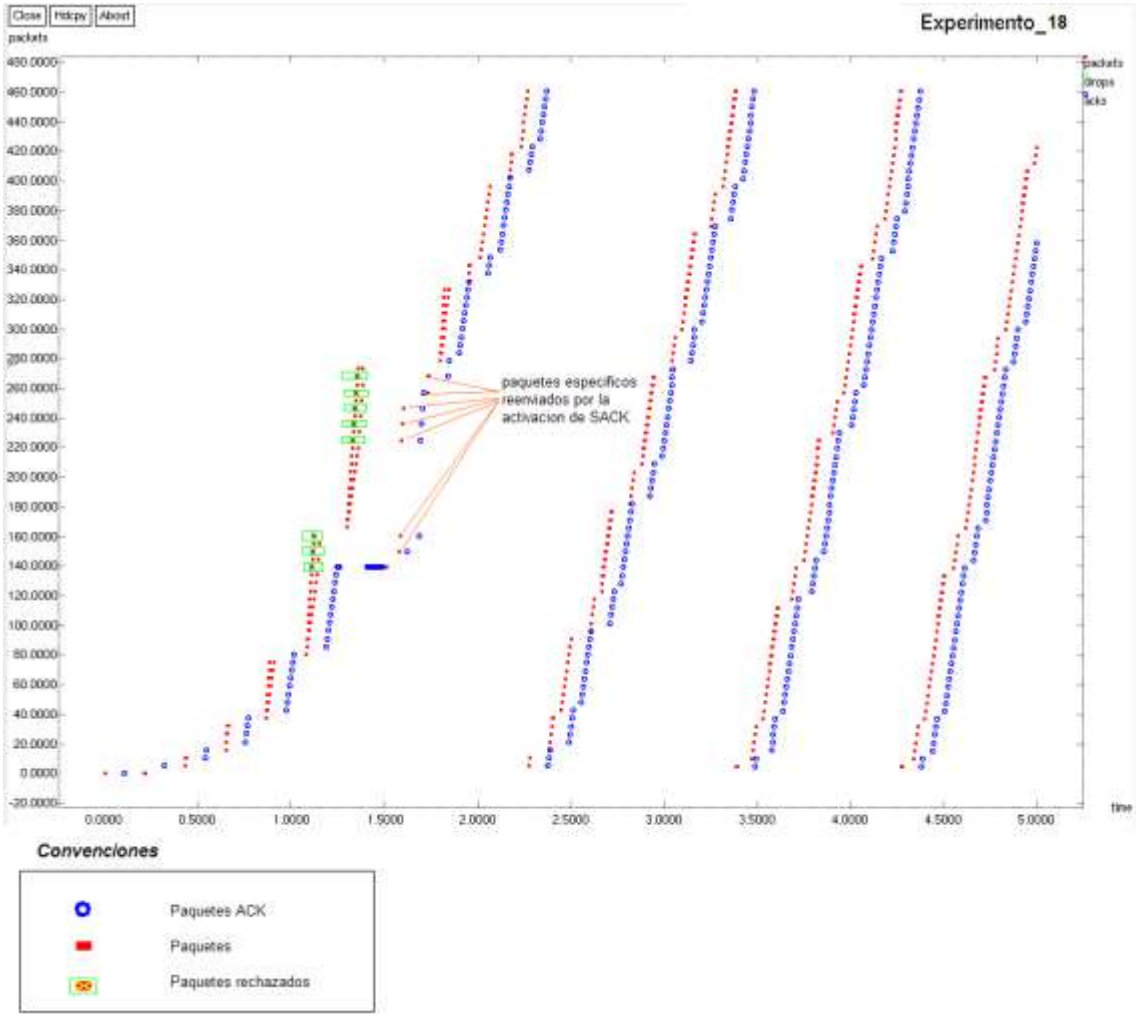


Figura 4. 16. Comportamiento de los paquetes en la transmisión de trafico FTP utilizando dos nodos TCP habilitada la opción SACK.

En las gráficas 4.14, 4.15 y 4.16 se muestra el comportamiento de los paquetes y los ACKs debido al flujo de transporte de datos FTP a través de un enlace compuesto por dos nodos utilizando TCP y la opción SACK habilitada. En las dos primeras gráficas se aprecia el comportamiento de los paquetes cuando se pierden 2 y 1 paquetes respectivamente, aunque no es muy evidente la importancia de SACK cuando se pierde mas de un paquete, podemos ver que en la grafica 4.14 y 4.15 la transmisión de datos continua reenviando el paquete que no fue acusado de recibo; en la grafica 4.16 se puede apreciar mucho mejor la importancia de este mecanismo, ya que en este existen mas pérdidas de paquetes. SACK permite retransmitir solo los paquetes que no han sido acusados de recibo por lo que se ahorra recursos de red al no reenviar paquetes que llegaron al receptor de forma correcta.

Debido al ahorro de recursos que presenta la habilitación de la opción SACK, se puede optar por la implementación.

4.5 Escenario 5: análisis de la utilización del mecanismo FACK en un enlace TCP entre dos nodos.

Con este escenario se analiza la utilización del algoritmo FACK y comparar su comportamiento con la opción *Reno+SACK* en la recuperación de datos perdidos.

Esta opción sumada al análisis del número de DUPACKs necesarios para activar la retransmisión hacen un buen punto de inicio para implementar un protocolo TCP mejorado y optimizado para redes inalámbricas.

.

4.5.1 Experimento 9 transmisión de trafico FTP entre dos nodos, utilizando la opción Reno + SACK y FACK cuando se presentan tres paquetes perdidos.

Características Principales del Enlace (figura 33)	
Nodos	2
Protocolo	TCP Reno
Ventana	16 segmentos
Opción SACK	activada
Aplicación	Trafico FTP
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb
Características Principales del Enlace (figura 34)	
Nodos	2
Protocolo	TCP Reno
Ventana	16 segmentos
Mecanismo FACK	activado

Aplicación	Trafico FTP
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb

Tabla 4. 11. Características de los Experimentos 9

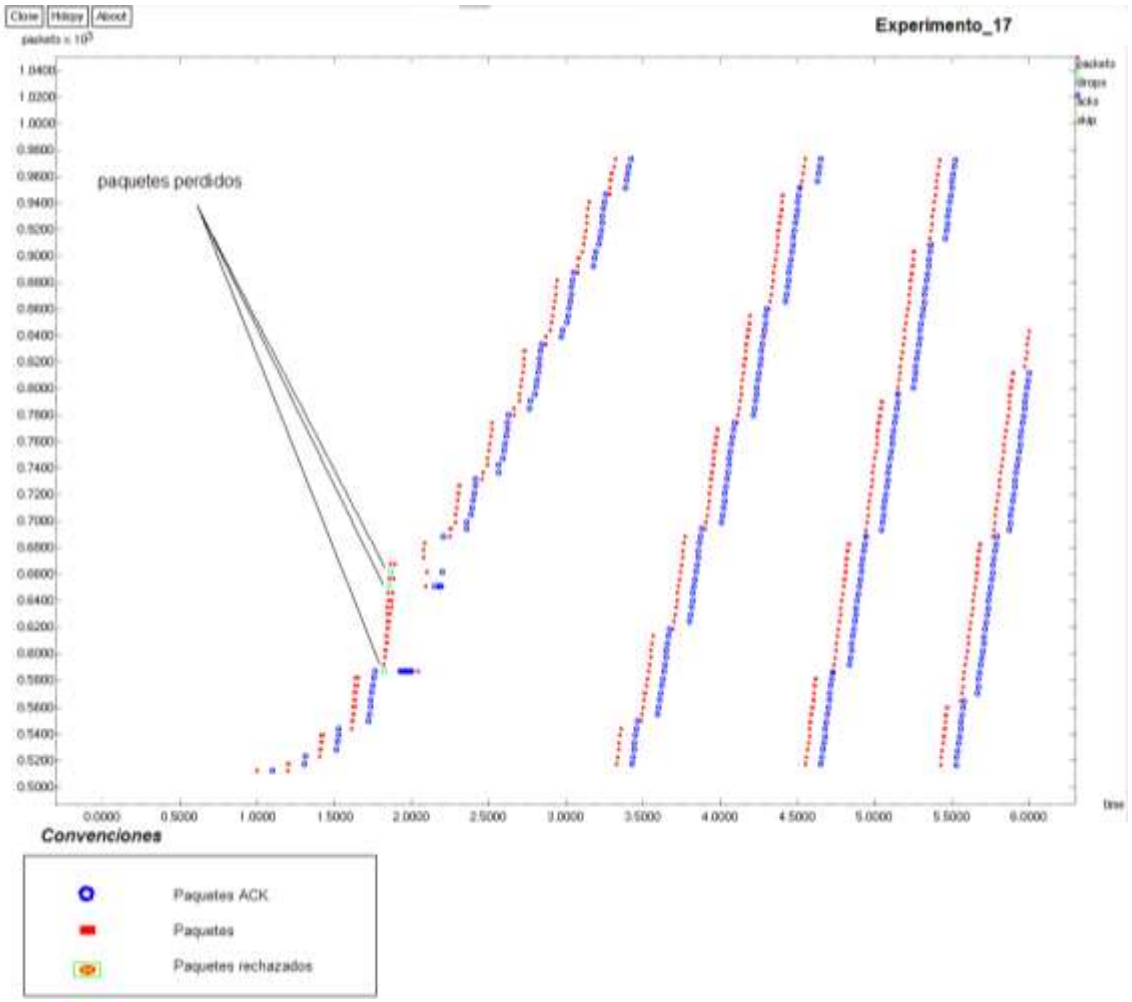


Figura 4. 17. Transmisión de tráfico FTP entre dos nodos, utilizando la opción Reno + SACK cuando se presentan tres paquetes perdidos.

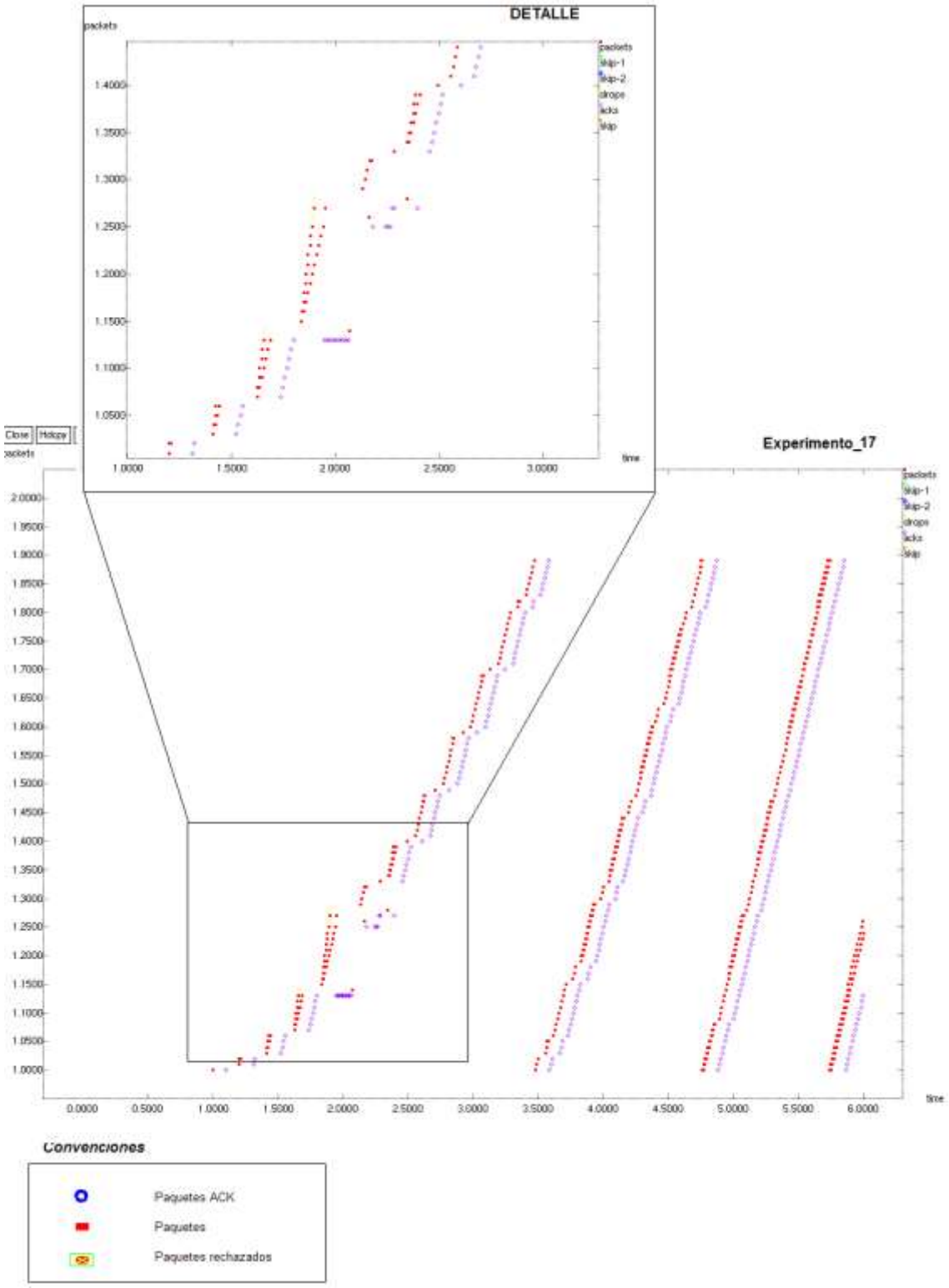


Figura 4. 18. Transmisión de tráfico FTP entre dos nodos, utilizando la opción FACK cuando se presentan tres paquetes perdidos.

En la figura 4.17, se puede apreciar el tráfico entre dos nodos TCP Reno en el cual se presentan pérdidas de paquetes. La opción SACK esta activada en la simulación de la gráfica 4.17, por lo que podemos ver que con la llegada de 3 DUPACKs se inicia la retransmisión de los paquetes faltantes (igual al escenario 4). En la figura 4.18se analiza el efecto del mecanismo FACK el cual fue explicado en el Capitulo III y su relación con la opción SACK. En este experimento nos podemos dar cuenta que el reinicio de paquetes se hace a los 2.1320 segundos (con la opción FACK) y que con la opción SACK (figura 4.17) el inicio de la transmisión estable de paquetes se realiza a los 2.0750 segundos; este retraso de unos milisegundos es debido al tratamiento que se hace en el transmisor, ya que el algoritmo FACK realiza mas operaciones que el SACK. Aunque a simple vista el mecanismo FACK no optimiza la transmisión de datos por el retraso existente entre la perdida del paquete y la reacción del mecanismo, se puede utilizar en una implementación TCP inalámbrica debido a que en la mayoría de implementaciones FACK es mas agresiva que TCP Reno o que TCP Reno+SACK, porque puede enviar la repetición de los bytes faltantes (perdidos) con un solo DUPACK.

La implementación de este mecanismo tiene que ser analizado cuidadosamente porque se podría caer en una retransmisión innecesaria de los datos ya que debido a la latencia de los paquetes en redes inalámbricas puede confundir al algoritmo una congestión y un retraso de paquetes debido a latencia o perdida.

4.6 Escenario 6: análisis de la modificación del tamaño del paquete TCP (Maximum Size Segment MSS) debido a la compresión de cabecera.

Una de las propuestas de optimización que fue planteada en el capitulo III es la de compresión de la cabecera TCP, la cual aplicada a enlaces inalámbricos podría ser benéfica en cuanto ahorro de paquetes ya que la idea es el disminuir el tamaño de la cabecera de un flujo determinado de TCP, haciendo que la relación bytes_transmitidos/información_útil sea mayor, ya que la misma información puede ser transmitida con un numero menor de bytes, evitando el desperdicio de ancho de banda.

A continuación se muestra la simulación y el análisis de las graficas del comportamiento de los paquetes modificando el tamaño de la MSS.

4.6.1 Experimento 10, modificación del tamaño del paquete TCP (Maximum Size Segment MSS) en un enlace de dos nodos TCP enviando trafico FTP a través de ellos.

Características Principales del Enlace (figura 4.19)	
Nodos	2
Protocolo	TCP Reno
Ventana	16 segmentos
MSS	576 (default RFC879)
Aplicación	Trafico FTP
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb

Tabla 4. 12. Características del Experimentos 10

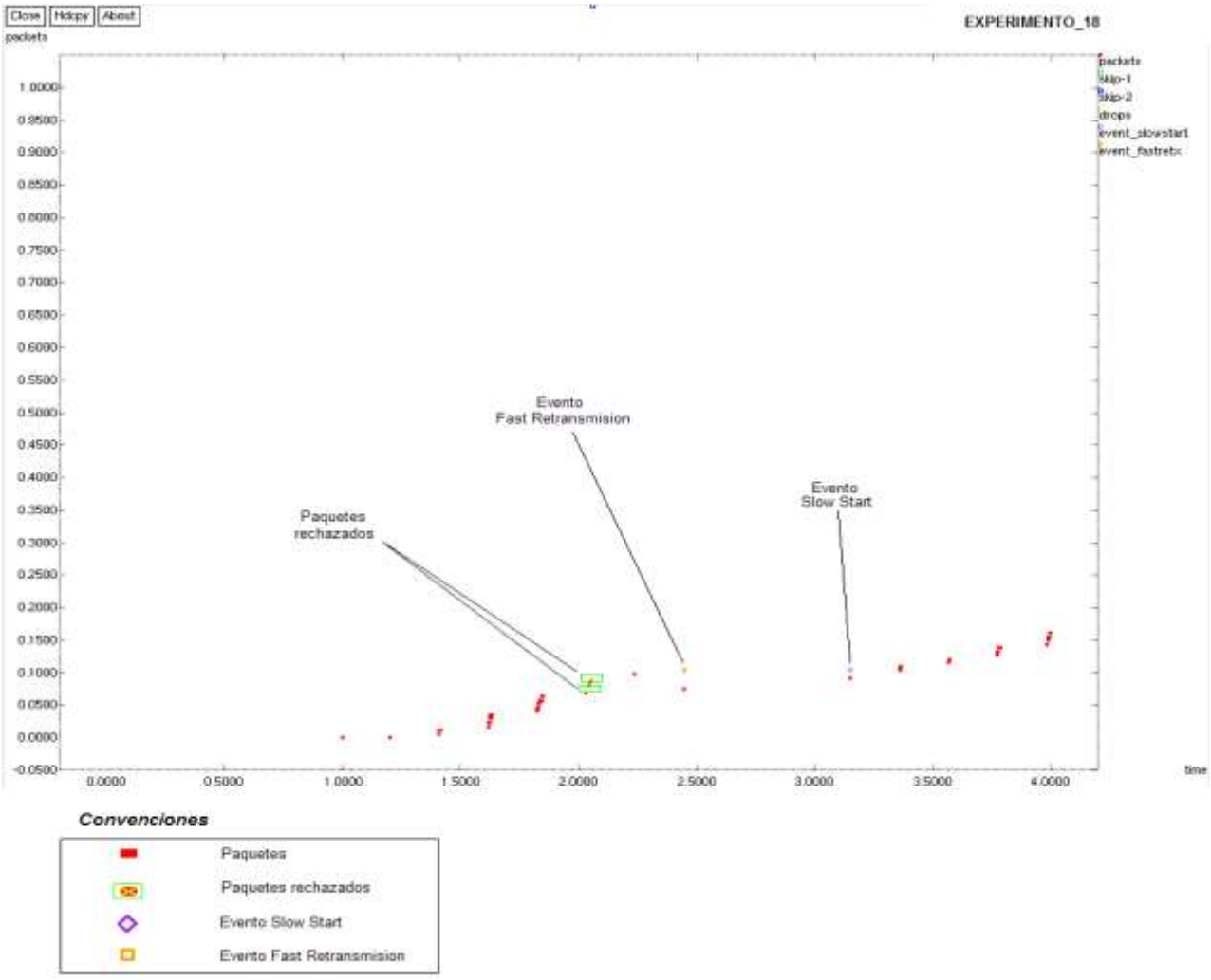


Figura 4. 19. Modificación del tamaño del paquete TCP (Maximum Size Segment MSS) en un enlace de dos nodos TCP enviando trafico FTP a través de ellos. MSS = 576 bytes.

Características Principales del Enlace (figura 4.20)	
Nodos	2
Protocolo	TCP Reno
Ventana	16 segmentos
MSS	540
Tamaño cabecera TCP	4 (RFC1144)
Aplicación	Trafico FTP
Tiempo de Simulación	1000 segundos
Ancho de Banda	26Kb

Tabla 4. 13. Características del Experimentos 10

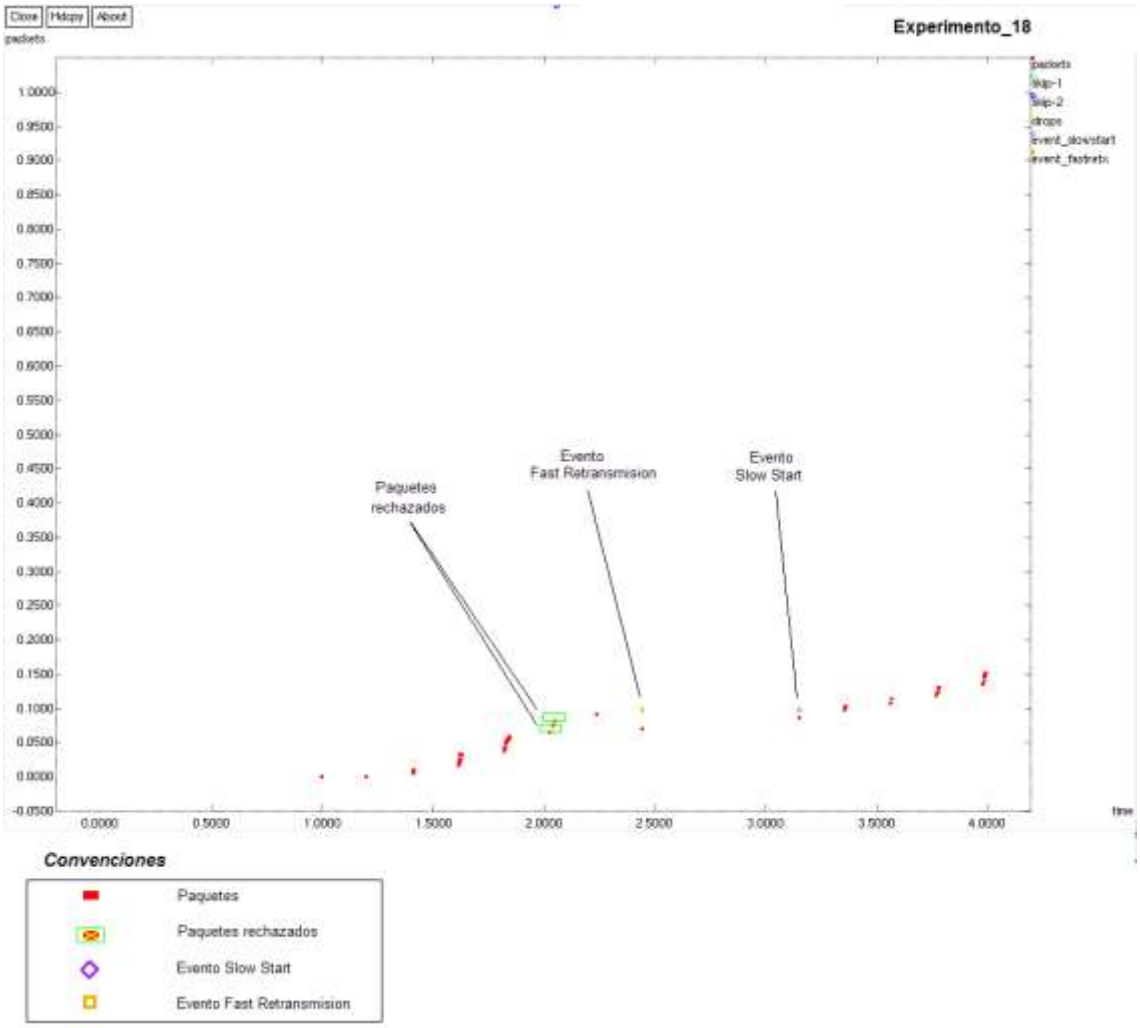


Figura 4. 20. Modificación del tamaño del paquete TCP (Maximum Size Segment MSS) en un enlace de dos nodos TCP enviando tráfico FTP a través de ellos. MSS= 540 bytes.

En las figuras 4.19 y 4.20 se muestra el tráfico TCP de dos nodos a través de TCP, utilizando una MSS de 576 y de 540, para un enlace sin compresión de la cabecera y la otra con compresión de esta, respectivamente.

En realidad no se encuentra una diferencia significativa entre el tamaño de cabecera de 576 bytes y 540 bytes. Aunque en términos de flujo total de las conexiones en una red de grandes dimensiones puede ser apreciable ya que si tenemos en cuenta que por cada flujo TCP se ahorran 36 bytes por paquete, entonces estamos haciendo una diferencia sustancial.

CAPITULO V

5. MODELO DE OPTIMIZACION

Después de un análisis de las opciones de optimización de los mecanismos y algoritmos, que tienen mejor desempeño para ser implementadas dentro de una red inalámbrica, se puede hacer una tabla de ventajas y desventajas de todos los mecanismos para que sea un modelo de referencia para la implementación en Sistemas Operativos.

Mecanismo de optimización	Ventajas	Desventajas
Incremento del tamaño de la ventana inicial de transmisión de TCP	• Mejora la respuesta en tiempo de la transmisión de los datos al iniciar la transferencia de datos en un enlace. • Mejora las condiciones de reinicio de una conexión después de una caída o restablecimiento de conexión. • En condiciones ideales de la red (red inalámbrica sin pérdidas) el tamaño máximo de la ventana de inicio de transmisión seria la misma que la máxima ventana de transmisión. • La implementación de esta optimización dentro de un sistema operativo que utilice el Kernel de Linux es rápida.	• Empeora el comportamiento de la red durante un episodio de congestión.
Análisis de la modificación del umbral del número de DUPACKS	• En condiciones de enlaces con un nivel alto de latencia y bajo nivel de congestión, el aumento del umbral de activación de retransmisión (activación de numero de DUPACKs de activación) es benéfico debido a que TCP no reenvía paquetes que están todavía en la red pero no han sido rechazados o	• La mala elección de un umbral de DUPACKs puede llevar a congestión de la red debido a retransmisiones innecesarias.

	dañados. • En condiciones de bajo nivel de congestión y gran número de pérdidas, es conveniente disminuir el número de DUPACKs para que active la retransmisión debido a que los paquetes perdidos deben ser retransmitidos rápidamente.	
Análisis de la utilización de mecanismos gestores de encolamiento activo *Active Queue Management* (AQM)	• La utilización de mecanismos gestores de colas es conveniente cuando hay gran nivel de trafico ya que este previene el descarte de paquetes en el receptor; esto se evidencia aun mas cuando los paquetes de cabecera son comprimidos ya que en la mayoría de modelos de compresión se utilizan los paquetes anteriores a los que se reciben para decodificar el paquete transmitido. • Los gestores de cola pueden mejorar el mecanismo de control de congestión del protocolo de transporte, ya que controlan el tráfico de los paquetes de forma anticipada. • La probabilidad de el envió de paquetes ECN hacia el emisor varia de acuerdo al tamaño de la cola, lo cual es conveniente debido a que si el promedio del tamaño de la cola aumenta, aumenta el envió de paquetes de notificación explicita de congestión.	• La desventaja más grande de este tipo de mecanismos es que la implementación de éstos es complicada debido a los algoritmos de encolamiento y descarte de paquetes que tienen. • Existe una gran diferencia entre el valor promedio de la cola y los valores instantáneos de cola de los paquetes, esto puede ser perjudicial porque si se elije un umbral alto para el promedio de cola, el receptor puede empezar a rechazar paquetes debido a sobrecarga del *buffer*.
Análisis de la habilitación de la opción SACK en transferencias de datos entre dos	• La utilización de la opción SACK dentro de TCP ahorra recursos de red debido a que no retransmite paquetes	• Esta opción es complicada de implementar si se utiliza compresión de la

nodos.	innecesarios. • La opción SACK es muy utilizada en redes cableadas y su implementación esta adoptada en muchos sistemas operativos.	cabecera TCP, esto debido a la codificación delta que se utiliza en la mayoría de compresiones.
Análisis de la utilización del mecanismo FACK en un enlace TCP entre dos nodos	• FACK es un mecanismo muy útil en redes con pérdidas debido al medio de transmisión, ya que retransmite rápidamente sin esperar a temporizadores ni a que se presenten una cantidad de DUPACKs alto. • La implementación de FACK dentro de un Sistema Operativo no es complicado; algunas implementaciones del Kernel de Linux vienen con esa opción.	• En redes con niveles altos de latencia, FACK no es útil debido a que puede hacer retransmisiones innecesarias.
Análisis de la modificación del tamaño del paquete TCP (Maximum Size Segment MSS) debido a la compresión de cabecera	• La disminución del tamaño de la cabecera mejora el gran medida la utilización del ancho de banda, sobretodo en grandes transferencias de paquetes y en medios con ancho de banda limitados como lo son los enlaces inalámbricos • La compresión de las cabeceras optimiza la utilización de los recursos de red, pero aumenta el procesamiento de la información, lo cual es mucho más rentable, que aumentar la capacidad del canal de transmisión, si tenemos en cuenta los últimos avances tecnológicos. • La compresión de las cabeceras disminuye la transmisión de información innecesaria.	• La codificación de la información de las cabeceras en deltas de información, incrementa el impacto de la perdida de paquetes, ya que en la mayoría de modelos de compresión se utilizan los paquetes anteriores a los que se reciben para decodificar el paquete transmitido, por lo que se pierde la sincronización de la decodificación.

Tabla 5. 1 tabla de las ventajas y desventajas de los mecanismos de optimización

Las características del nuevo modelo de las capas de transporte en la arquitectura WAP 2.0, van a basarse del TCP Reno, ya que en la mayoría de sistemas operativos existentes esta es la base del modulo TCP del núcleo del kernel. La nueva versión del protocolo que se plantea tiene agregados algunos de los mecanismos estudiados para optimizarlo a enlaces inalámbricos; a continuación se muestra la lista de mecanismos utilizados en el nuevo modelo.

- Incremento del tamaño de la ventana inicial de transmisión de TCP.
  Para la ventana de inicio de transmisión se recomienda que sea de 4 segmentos lo cual podría ser una buena opción para elegir, ya que disminuye el tiempo de estabilización y no tiene tantas perdidas por la sobrecarga del *buffer* de salida del emisor, lo cual ocurre cuando se utilizan ventanas de inicio de transmisión grandes. Debe tenerse en cuenta que para transmisiones de datos pequeñas (como se da en la mayoría de transmisiones celulares inalámbricas) es preferible que la ventana de inicio se mayor para que la transmisión sea eficiente.

- Análisis de la modificación del umbral del número de DUPACKS.
  La utilización de un número determinado para el umbral DUPACK para la retransmisión es fundamental, por el análisis de los experimentos a través del emulador NS-2, la opción más conveniente es de 4, pero se debe tener precaución en redes que tengan un nivel de latencia alto.

- Análisis de la utilización de mecanismos gestores de encolamiento activo *Active Queue Management* (AQM).
  La implementación de este tipo de mecanismos de gestión de colas, son dirigidos a la parte de enrutadores, pasarelas, y gateways, en las que el trafico es mayor. Como resultado del análisis de este tipo de mecanismos se puede concluir que es deseable tener una implementación de este tipo en la gateway de soporte del dispositivo móvil que implemente TCP, puesto que esto permite la optimización de los recursos de la red inalámbrica.

- Análisis de la habilitación de la opción SACK en transferencias de datos entre dos nodos.

  La opción SACK de TCP es una de las opciones que son necesarias ya que este mecanismo ahorra paquetes de retransmisiones innecesarias.

- Análisis de la utilización del mecanismo FACK en un enlace TCP entre dos nodos.

  El algoritmo FACK utiliza la información adicional dada por la opción SACK para mantener una medida explicita del numero total de bytes de los datos transmitidos a través de la red, aunque esto es deseable ya que puede retransmitir solo los bytes perdidos puede volverse un mecanismo no conveniente debido a que puede hacer retransmisiones innecesarias si se implementa en redes con alto nivel de latencia.

  No se recomienda que se haga la implementación de FACK si el enlace presenta problemas de latencia, lo cual es muy común en redes inalámbricas grandes.

- Análisis de la modificación del tamaño del paquete TCP (Maximum Size Segment MSS) debido a la compresión de cabecera.

  La implementación de la compresión de la cabecera TCP es muy conveniente en enlaces con poco ancho de banda, ya que elimina el envio de información innecesaria y se puede optimizar el rendimiento del canal, transmitiendo mas segmentos con información útil (payload) y menos información repetida, sin embargo si hay mucha perdida de paquetes en la transmisión, este mecanismo afectará el desempeño del canal ya que un paquete es reconstruido con la información del paquete anterior, cuando un paquete se pierde, se pierde la secuencia y se descartaran los paquetes hasta tanto no se envie un nuevo paquete completo que permita sincronizar la transmisión. Se puede decir entonces que si el enlace tienes pocas perdidas, este mecanismo optimizará la transmisión considerablemente.

CAPITULO VI

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES

En las pruebas efectuadas con las ventana de inicio de transmisión se pudo apreciar que el trafico FTP no varia en mucho del trafico con distribución exponencial, además las modificaciones de la ventana de inicio tienen diferentes comportamientos al ir aumentando su tamaño.

Las graficas generadas al modificar el tamaño de la ventana de inicio varían en el numero de paquetes perdidos debido al tamaño de la cola del *buffer* de transmisión, para las simulaciones con una ventana de transmisión mayores a 4 las pérdidas incrementan significativamente, llegando a comportamientos erráticos en una ventana de 64 lo que sugiere que el eliminar este mecanismo y dejar fija la ventana de inicio en un tamaño grande trae consigo pérdidas por congestión y pérdidas por exceso de paquetes en el *buffer* de salida. Podemos concluir del análisis del aumento del tamaño de la ventana de inicio de transmisión que, la importancia de la eficiencia de un mecanismo de encolamiento de paquetes dentro de la gateway es vital si se pretende aumentar el tamaño de la ventana de inicio, ya que las “ráfagas” de paquetes al inicio de la transmisión hasta llegar a un punto de estabilización de la transmisión son altas en comparación con ventanas de inicio de tamaño pequeño. Podemos recomendar con este trabajo que para microceldas celulares con bajas pérdidas y gateways con alta eficiencia de recepción el aumento de la ventana es un buen mecanismo de optimización del protocolo de transporte.

Con las pruebas efectuadas al aumentar el tamaño del umbral de DUPACKS para que se active la retransmisión de paquetes se tuvo un resultado evidente, en el cual al aumentar el umbral de DUPACKs el mecanismo de reenvió de paquetes se activa mas tarde, lo cual no es tan ventajoso en enlaces inalámbricos con un novel de latencia alto.
Como conclusión dentro del estudio utilizando las graficas generadas, podemos afirmar que un retraso en la retransmisión de paquetes puede ser un buen mecanismo de optimización en

medios de transmisión (interfaz aérea) con pérdidas, ya que no se retransmitirán paquetes los cuales no se han perdido debido a congestión sino a pérdida.

Existen muchos estudios sobre mecanismos de gestión de colas activas, uno de los más utilizados e implementados es RED, por lo cual se hizo la simulación a través de NS-2. El resultado de la simulación nos da un comportamiento de la cola más estable que con el mecanismo por defecto que es Drop Tail, el cual según las graficas estudiadas no es un buen método para un dispositivo que soporte múltiples clientes en una red inalámbrica. Se hizo también un estudio de un gestor de colas activas (RED con análisis de colas con lógica difusa) pero a nivel teórico y el cual se cree que pueda manejar de forma mas eficiente las colas en un trafico TCP.

La utilización de SACK en Reno es una opción que según los resultados y su ya conocida implementación es conveniente para enlaces inalámbricos con perdidas de paquetes. La implementación de esta opción esta ya desarrollada dentro del modulo de TCP en el núcleo de muchas distribuciones de Linux.

En la compresión de la cabecera de TCP podemos tener varias conclusiones, la primera y la principal es que la mayoría de los campos de TCP como de IP pueden ser fácilmente comprimidos puestos que ellos nunca o casi nunca cambian, y además algunos de ellos pueden ser deducidos a partir de la información de otros campos. Solo algunos campos necesitan mecanismos mas sofisticados de compresión. Estos campos son:

Identificación IPv4 (16 bits)		IP-ID
Numero de Secuencia TCP (32 bits)		SN
ACK TCP (32 bits)		ACKN
Reservado TCP (4 bits)		
Bandera ECN TCP (2 bits)		ECN
Ventana TCP (16 bits)		WINDOW
Opciones TCP	SACK permitido (2 octetos)	
	TCP SACK	SACK
	Timestamp TCP (32 bits)	TS

Tabla 6. 1. Campos que necesitan mecanismos mas sofisticados de compresión

Otra de las conclusiones con la opción de la compresión de los datos de cabecera de los protocolos TCP e IP son que algunos de los campos, y algunos valores de contexto pueden ser fácilmente compartidos puesto que ellos cambian rara vez o cambian de forma predecible. Estos campos son:

Cabecera IPv4

Campo	**Tipo**	**Es un campo con información compartible?**
Longitud de la cabecera	Estático – conocido	Si
ToS	Variable	Si
Longitud del paquete	Se puede inferir	-
Identificación	Variable	Si
Tiempo de vida	Variable	Si
Protocolo	Estático	-
Cheksum de la cabecera	Se puede inferir	-
Dirección fuente	Estático - definido	-
Dirección destino	Estático- definido	-

Tabla 6. 2. Cabecera IPv4

Cabecera TCP

Campo	**Tipo**	**Es un campo con información compartible?**
Puerto de la fuente	Estático – conocido	Si
Puerto destino	Estático – conocido	Si
Data offset	Se puede inferir	-
Ventana	Variable	Si
Bits reservados	Variable	Si
Ventana de inicio	Variable	Si

Tabla 6. 3. Cabecera TCP

Opciones TCP

Opción	**Solo en el evento de sincronía SYN?**	**Es un campo con información compartible?**
Opción MSS	Si	Si
Opción de la escala de la ventana	Si	Si
Opción de SACK permitida	Si	Si
Opción Timestamp	No	Si

Tabla 6. 4. Opciones TCP

6.2 Trabajos futuros

Como trabajos futuros está el analizar los retrasos que generan el tratamiento de los paquetes en nodos intermedios, la seguridad de este tratamiento de paquetes en dichos nodos y la eficiencia del modelo TCP planteado en esta investigación.

Creemos que las modificaciones que se plantean en este trabajo no van en contra de las políticas que se plantean en la recomendación original del Protocolo de Control de Transporte TCP (RFC0793), ya que cumple con los objetivos para los cuales se creo este protocolo. Por lo tanto como trabajo futuro se puede ver la opción de la modificación de otros protocolos (del conjunto de protocolos recomendados en la especificación WAP 2.0) cuando son utilizados en enlaces inalámbricos, con características similares a las de las redes inalámbricas celulares.

La utilización de métodos de compresión de las cabeceras de los paquetes es aun insipiente a nivel comercial, y creemos que es un método de optimizar recursos como ancho de banda de los enlaces. La compresión tanto de datos de cabecera como de la carga útil de los paquetes de una forma automática dentro de una red es un método eficiente de ahorro de recursos de red en una transmisión de datos, creemos que este tipo de compresiones de paquetes y de datos a nivel de otros protocolos comprendidos en la especificación WAP 2.0 puede ser una solución a problemas de utilización eficiente del ancho de banda en un enlace inalámbrico.

GLOSARIO

Acceso cableado: El servicio de enlace bidireccional por medio de cableados entre una red pública de telecomunicaciones y el usuario para la transmisión de signos, señales, escritos, imágenes, voz, sonido o información de cualquier naturaleza. Cada servicio de telecomunicaciones que se preste al usuario final se sujetará a las disposiciones legales, reglamentarias y administrativas aplicables.

Acceso inalámbrico: El servicio de enlace radioeléctrico bidireccional entre una red pública de telecomunicaciones y el usuario para la transmisión de signos, señales, escritos, imágenes, voz, sonido o información de cualquier naturaleza. Cada servicio de telecomunicaciones que se preste al usuario final se sujetará a las disposiciones legales, reglamentarias y administrativas aplicables.

Ancho de banda necesario para una emisión: Para una cierta clase de emisión, el ancho de la banda de frecuencia que es apenas suficiente para garantizar la transmisión de información a la velocidad y con la calidad requeridas bajo condiciones específicas.

Enlace: Medio de transmisión con características específicas, entre dos puntos, esto puede ser mediante canal o circuito Conjunto de instalaciones terminales y red de interconexión que funciona en un modo particular a fin de permitir el intercambio de información entre equipos terminales

Estación base: Estación terrestre para proporcionar el Servicio móvil terrestre

Estación móvil: Estación de servicio móvil destinada a ser utilizada en movimiento o mientras esté detenida en puntos no determinados.

Segmento: un segmento es cualquier paquete de datos o acuses de recibo (o ambos) de los protocolos TCP/IP.

Tamaño de segmento máximo del emisor (*SENDER MAXIMUM SEGMENT SIZE - SMSS*): el SMSS es el tamaño del segmento más grande que el emisor puede transmitir.

Tamaño de segmento máximo del receptor (RECEIVER *MAXIMUM SEGMENT SIZE - SMSS*): es el tamaño del segmento más grande que el receptor esta dispuesto a aceptar. Este es el valor especificado en la opción MSS enviada por el receptor durante el inicio de la conexión. Si la opción MSS no es utilizada el tamaño es de 536 bytes.

Ventana de congestión (CONGESTION WINDOW - CWND): una variable de estado que limita la cantidad de datos que TCP puede enviar. En cualquier momento, TCP no debe enviar datos con un número de secuencia más grande que la suma del número de secuencia acusado de recibo más grande y el mínimo valor entre CWND y RWND (ventana de recepción)

Ventana inicial (INITIAL WINDOW - IW): la ventana inicial es el tamaño de la ventana de congestión del emisor después de que se haya completado el proceso de *handshake*.

FLIGHT SIZE: es la cantidad de datos que se han enviado pero que no han sido acusados de recibo.

WAP: Protocolo de Aplicaciones Inalámbrico, es una especificación abierta que facilita a los usuarios móviles con dispositivos móviles de acceso fácil e interactivo con información y servicios instantáneos.

HTTP: Protocolo de transporte de hipertexto.

FTP: Protocolo de transferencia de archivos.

UDP (User Datagram Protocol): Protocolo del datagrama del usuario, protocolo falto de conexión (que no demanda conexión directa entre el remitente y el destinatario), permite el envío de paquetes de datos por medio del Internet.

Gateway: puerta en comunicaciones de red, es una combinación de programa y hardware que comunica dos tipos diferentes de redes.

SACK: Acuses de Recibo Selectivos (Selective Acknowledgments, SACK)

Datagrama: es el tamaño de la Cabecera IP. la Cabecera TCP y los Datos.

MSS: Tamaño Máximo de Segmento. Se compone de la Cabecera TCP y los Datos.

BIG-ENDIAN: Almacenamiento de bytes en la memoria del disco, el bit más significativo corresponde a la memoria más alta y el menos significativo a la memoria más baja.

FCC: Federal Communications Commission

FIFO: First In First Out – EL Primero en llegar es el primero en salir. Es una lógica de colas.

GUI: Graphic User Interface – Interfaz Gráfica de Usuario

LAN: Local Area Network – Redes de Área local

LITTLE-ENDIAN: Almacenamiento de bytes en la memoria del disco, el bit más significativo corresponde a la memoria más baja y el menos significativo a la memoria más alta.

3G: Tercera Generación

RTT : Rond Trip Time, tiempo que un paquete se demora en ir desde el emisor hasta el receptor.

DUPACK : Acuse de recibo duplicado, *Duplicated* ACK.

BIBLIOGRAFIA

[ACBR 03]	Iara Augustin, Adenauer Correa Yamin, Jorge Luis Victoria Barbosa, Cláudio Fernando Resin Geyer, Towards Taxonomy For Mobile Applications With Adaptive Behavior. 2003
[MM, JS, M, 97]	M. Mathis, J. Semske, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication, Review*, 27(3), July 1997.
[JP, VF, JK, 98]	J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, 1998.
[SF, VJ, 97]	S. Floyd and V. Jacobson. Random Early Detection gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1997
[NS-2]	http://www.isi.edu/nsnam/ns/, Network Simulator 2
[HI,TI 00]	H. Inamura y T. Ishikawa, NTT DOCOMO. draft-inamura-docomo-00.txt
[FK, SS 99]	Feng, Kandlur, Saha, & Shin, 1999b, “TCP enhancements for an Integrated Services Internet”
[CF98]	Clark & Fang, 1998, “Explicit Allocation of Best Effort Packet Delivery Service”
[AFP98]	Allman, M., Floyd, S. y C. Partridge, "Increasing TCP's Initial Window Size”, RFC 2414, September 1998
[RFC0793]	DARPA INTERNET PROGRAM, TRANSMISSION CONTROL PROTOCOL.1981.
[RFC2001]	W. Stevens, NOAO .Network Working Group. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. 2001
[RFC2581]	M. Allman, NASA Glenn/Sterling Software; V. Paxson, ACIRI / ICSI; W. Stevens, Consultant. Network Working Group. TCP Congestion Control, 1999.
[RFC1144]	V. Jacobson, Compressing TCP/IP Headers for Low-Speed Serial Links. 1990
[RFC1323]	V. Jacobson, LBL; R. Braden, ISI; D. Borman, Cray Research. TCP Extensions for High Performance. 1992.
[RFC2507]	M. Degermark, Lulea University of Technology/SICS; B. Nordgren, Lulea University of Technology/Telia Research AB; S. Pink, Lulea University of Technology/SICS. IP Header Compression. 1999
[ROHC-TCP]	Qian Zhang, HongBin Liao, Wenwu Zhu, Ya-Qin Zhang. Microsoft Research Asia. ROHC-TCP: Unidirectional Operation Enhancements. 2003.
[DB, MC, 00]	Daniel P. Bovet, Marco Cesati. Understanding the Linux Kernel. First Edition October. O’Reilly. 2000
WAP-210-WAPArch-20010712	http://www.wapforum.org/. Wireless Application Protocol, Architecture Specification.
WAP-225-TCP- -	http://www.wapforum.org/. Wireless Profiled TCP

20010331-a

WAP-236-WAESpec-20020207-a — http://www.wapforum.org/. Wireless Application Environment Specification

WAP-230-WSP — http://www.wapforum.org/. Wireless Session Protocol Specification.

WAP-224-WTP-20010710-a — http://www.wapforum.org/. Wireless Transaction Protocol.

WAP-259-WDP-20010614 — http://www.wapforum.org/. Wireless Datagram Protocol

WAP-261-WTLS-20010406-a — http://www.wapforum.org/. Wireless Transport Layer Security.

WAP-159-WDPWCMPAdapt-20010713-a — http://www.wapforum.org/. WDP and WCMP Wireless Data Gateway Adaptation