FACULTAD DE CIENCIAS EXACTAS
UTC - UNICEN

TRABAJO FINAL

"Implementación Modular DCCP en Java"

Tutor:
Ing. Guillermo Rigotti M.Sc.

Autores:
Benavidez Luis
Pástor Israel

Mayo, 2007
Universidad Nacional del Centro de la Provincia de Buenos Aires
Tandil – Argentina

Dedicatoria

A Dios todopoderoso y eterno;

A nuestras Madres;

A nuestros hermanos;

A nuestro profesor guía;

A nuestros amigos y amigas.

La humildad trae gracia y felicidad a la vida; permite acomodarse a las situaciones difíciles sin pensar en lo que se está dejando ó renunciando, nos vuelve más sencillos y naturales, permite que nos concentremos en lo que estamos haciendo, y que lo hagamos correctamente.

La humildad hace que podamos ver los beneficios en cada escena de la vida, haciendo que nuestras interacciones giren en un ambiente más agradable, así logramos un lugar en el corazón de todos, eliminando en un segundo aquello que nos hiere y no nos deja crecer.

Siendo humildes comprenderemos que aún tenemos mucho por mejorar, mucho que aprender y que podemos ocuparnos en la tarea de crecer.

Luis Benavidez, Israel Pástor.

Agradecimientos

No debo dejar pasar esta oportunidad sin agradecer a todas aquellas personas que en mayor o menor medida han ayudado a que esta tesis se desarrolle:

Primeramente quiero agradecer a DIOS por haber permitido que viva hasta esta etapa de la vida y darme la oportunidad de realizar mis estudios universitarios.

Quiero agradecer a mi MADRE Y HERMANA Patricia, por que han sido un pilar importante de mi vida, mi fuente de inspiración y las personas que más amo en el mundo, dándome su cariño, su apoyo y su comprensión.

Quiero agradecer a mi profesor guía Ing. Guillermo Rigotti M.Sc., que gracias a el, a su interés, su dedicación y su paciencia se ha hecho posible que esta tesis saliera adelante.

Quiero agradecer a mi familia en general, por estar siempre a mi lado tanto en los tiempos buenos como también en los malos.

Quiero agradecer a mis amigas(os) que en todo momento de mi tesis siempre me han apoyado y aconsejado.

Con mucho cariño, humildemente Israel Pástor

Agradecimientos

Muchas veces me he planteado la opción de comenzar y acabar esta sección únicamente con la palabra “Gracias.”... y nada más. Sencillamente por evitar el resumir en una sola página la mención de tantas personas a las que debo mucho:

Agradezco a Dios nuestro señor por la oportunidad que he tenido de aprender, mejorar y de crecer junto a personas tan especiales para mí.

Agradezco principalmente a mi MADRE que con su esfuerzo y dedicación me permitió salir adelante en todo el transcurso de mi vida y apoyarme en todo lo que estuvo a su alcance.

Agradecimiento especial para mi Profesor Ing. Guillermo Rigotti M.Sc., por su amistad, paciencia y su constante apoyo durante el desarrollo de esta tesis.

A mis hermanos los cuales me supieron brindar su apoyo incondicional.

A mis compañeros y amigos por compartir las angustias y gratificaciones, a todos ellos gracias.

Con mucho cariño, humildemente Luis Benavidez.

Índice general

CAPÍTULO 4:

CAPÍTULO 5:

Índice de gráficos

Índice de tablas

Abstract

In the last years the requirements to sent information, part of application in the fields of the Internet, has diversified by all meanings. The protocols of the transport which is traditionally used for these meanings exist in TCP and UDP, which are recommendations to provide the mechanics for the transfer of information in any type of service. But nothing of these new requirements have been applied in real time.

Which is one of the missing tools is the control of congestion en UDP, which potentially could be a risky protocol. At this point the only alternative is using the protocol of transfer from TCP, which provides tools of control de congestion, but the mechanic of new transmissions is a disadvantage. This means that from going to A to B is what is causing this. DCCP combines the best of the two protocols in the context of the transmission in terms of communication, supporting tools controlling the congestion, without new transmissions.

For these and other reasons there is a prototype created with the characteristics of the protocol DCCP written in the [RFC 4340] which can add a required function in an easy and structured way, to evaluate the functions in new conditions, which is, communicating two parts of the protocol (customer related) through the Internet.

An important characteristic inside this application is the opportunity of realizing the implementation of the prototype DCCP at a new user level, without interfering the operational system. For this final reason, the language used is Java.

Resumen

En los últimos años los requerimientos de envío de la información por parte de las aplicaciones en el ámbito de la Internet se han diversificado significativamente. Los protocolos de transporte que se ha utilizado tradicionalmente para esos fines han sido *TCP* y *UDP* que son los encargados de proveer los mecanismos para la transferencia de los datos proporcionando cada uno un tipo de servicio diferente. Sin embargo, ninguno de los dos tipos de servicio provisto por estos protocolos se adapta satisfactoriamente a estos nuevos requerimientos de las aplicaciones de tiempo real.

La carencia de mecanismos de control de congestión en *UDP* hace de este protocolo una opción aventurada. En este momento la única alternativa es usar el protocolo de transporte de *TCP*, que proporciona mecanismos de control de congestión, pero el mecanismo de nueva transmisión es una desventaja debido a la alta tardanza de punta a punta que esto causa. *DCCP* combina lo mejor de los dos protocolos dentro del contexto de transmisión de medios de comunicación, apoyando mecanismos de control de congestión sin nuevas transmisiones.

Por estas y otras razones, se desarrollo un prototipo con las características del protocolo DCCP descrito en el [RFC 4340] al cual se le pueda agregar de manera clara y modular, la funcionalidad requerida, para evaluar su funcionamiento en condiciones reales, es decir, comunicando las dos partes del protocolo ("*cliente-servidor*") a través de la Internet.

Una característica importante dentro de esta aplicación es la portabilidad la cual se la realizo mediante la implementación del prototipo *DCCP* a nivel de usuario, es decir sin inferir en el sistema operativo, por este último motivo, el lenguaje utilizado fue *Java*.

Capítulo 1

Introducción

En un principio (años 60), la conexión entre ordenadores sólo era posible entre modelos de una misma marca, llegándose incluso a dar el caso de que diferentes modelos de un mismo fabricante no podían comunicarse entre sí. Esta limitación venía dada por el hardware que componía la máquina y como ésta era gobernada por el software que la hacía funcionar. Además, esta comunicación sólo era posible entre dos ordenadores (vía línea telefónica, comunicación por un puerto serie o paralelo, etc.). De esta forma, la incompatibilidad entre diferentes fabricantes de ordenadores era casi total, lo que implicaba tres serias limitaciones a la expansión de la informática.

Se debían comprar todas las máquinas y periféricos al mismo fabricante. Esto permitía un monopolio absoluto de las pocas marcas existentes:

- El precio de estos primeros ordenadores era muy elevado, de tal forma que tan sólo eran tomados para gobiernos, universidades y grandes empresas.

- Las posibilidades de conexión entre diferentes equipos estaban limitadas a ordenadores de la misma marca, llegándose incluso al caso de ser sólo compatibles con su mismo modelo y no con el resto de ordenadores de la misma casa.

A partir de los años 70, las redes de interconexión de ordenadores ya habían avanzado bastante, permitiendo la comunicación de diferentes tipos de máquinas. Para conseguir este objetivo, fue básico que organismos internacionales como el Instituto de Ingenieros Eléctricos y Electrónicos o The Institute of Electrical and Electronics Engineers, (*IEEE*) y la Organización Internacional para la Estandarización o International Standards Organization (*ISO*) regularan los mecanismos y formas en que se debían realizar las comunicaciones entre equipos informáticos. Se definió un modelo teórico denominado modelo o Interconexión de Sistemas Abiertos u Open Systems Interconection (*OSI*) [6], que especificaba un conjunto de 7 capas que permitían aislar el ordenador de la red a la que se encontraba conectado, permitiendo la interconexión de diferentes ordenadores a una misma red.

Una vez conseguida la conexión de ordenadores diferentes en una misma red, se pasó a una segunda fase que consistió en la comunicación entre los diferentes tipos de redes existentes hasta formar la gran red de redes que actualmente denominamos Internet[1].

La red Internet utiliza una familia de protocolos[2] denominada Protocolo de Control de Transmisión de Internet o Transmisión Control Protocol/Internet (*TCP/IP*) [1] (entre los que podemos destacar Protocolo de Control de Transmisión o Transmisión Control Protocol (*TCP*) [2], Protocolo de Datagramas del Usuario o User Datagram Protocol (*UDP*) [3], Protocolo de Control de Congestión de Datagramas o Datagram Congestion Control Protocol (*DCCP*) [4] y Protocolo de Internet o Internet Protocol (*IP)* [5].

[1] Internet, Red informática de comunicación que permite el intercambio de todo tipo de información.
[2] Protocolos, Conjunto de normas y procedimientos útiles para transmisión de datos.

IP, es un protocolo de interconexión de redes heterogéneas que se encarga del transporte de los datagramas (paquetes de datos) a través de la red.

UDP, es un protocolo de comunicación entre ordenadores de nivel superior al *IP*, sin conexión y que no proporciona fiabilidad a la comunicación.

TCP, es también un protocolo de nivel superior al *IP*, pero orientado a conexión y fiable.

DCCP, es un protocolo de nivel de transporte que tiene la capacidad de una entrega rápida de datos que mantiene un mecanismo de control de congestión[3].

En los últimos años los requerimientos de envío de la información por parte de las aplicaciones en el ámbito de la Internet se han diversificado significativamente. Los protocolos de transporte que se ha utilizado tradicionalmente para esos fines han sido *TCP* y *UDP* que son los encargados de proveer los mecanismos para la transferencia de los datos proporcionando cada uno un tipo de servicio diferente. Sin embargo, ninguno de los dos tipos de servicio provisto por estos protocolos se adapta satisfactoriamente a estos nuevos requerimientos de las aplicaciones de tiempo real.

El trabajo se enmarcara en un protocolo en fase de desarrollo y experimentación, que se encuentra en el seno del Grupo de Trabajo en Ingeniería de Internet o Internet Engineering Task Force (*IETF*) descrito en [RFC4340]. Denominado *DCCP* mencionado anteriormente.

[3] Control de Congestion, conjunto de técnicas para detectar problemas que surgen en una red.

Como alternativa adicional a dichas implementaciones, consideramos que una implementación portable, y modular, seria de utilidad para poder evaluar este protocolo, desarrollando prototipos y experimentando con diferentes mecanismos de control de congestión de acuerdo al medioambiente de comunicación y al tipo de aplicación.

La modularidad dependerá del diseño de la aplicación, mientras que un alto grado de portabilidad sería posible utilizando lenguaje java[4], lo que conduce a pensar en una implementación a nivel aplicación, la cual interactúe con el protocolo *UDP* en lugar de *IP*. Se considera que esto no afectará significativamente la performance de los prototipos desarrollados.

[4] Java, es un lenguaje de programación de propósito general, orientado a objetos forman parte de una red.

Capítulo 2

Objetivos, alcances y limitaciones de la implementación

Antes de comenzar a describir con mayor detalle el desarrollo del trabajo de investigación, especificaremos cuál es el objetivo perseguido y aclarar cuáles son los alcances y limitaciones.

El objetivo fundamental del presente trabajo es obtener un prototipo[5]de la parte de conexión, desconexión y transmisión de datos del *DCCP*, tal como es definido en el [RFC 4340], al cual se le pueda agregar de manera clara y modular, la funcionalidad requerida, para evaluar su funcionamiento en condiciones reales, es decir, comunicando las dos partes del protocolo a través de la Internet.

Posteriormente se describen las características principales del desarrollo, remarcando sus alcances y limitaciones.

[5] Prototipo, un prototipo se refiere a cualquier tipo de máquina en pruebas, o un objeto diseñado para una demostración.

- ✓ Programación simple, modular y código fácil de modificar a quien desee incorporar nueva funcionalidad en el protocolo,

- ✓ Portabilidad, esta característica es importante en esta aplicación, ya que por su naturaleza, exige ser probada entre un par de equipos conectados a la Internet.

Dos aspectos fundamentales para lograr portabilidad son la implementación del prototipo a nivel usuario, es decir, sin interferir con el sistema operativo, y la elección de un lenguaje de amplia difusión. Por este último motivo, el lenguaje elegido fue java. Como se menciona en el siguiente párrafo, en un principio la elección de java trae un problema relacionado con el acceso al nivel de red, que constituye el soporte de comunicación para este prototipo.

Uso de *UDP* como soporte de comunicación. Esta característica resulta contraria al uso de las facilidades de comunicación que debería hacer un protocolo de nivel transporte, es decir, uso de *IP*, que esta en el nivel de red. Se decide utilizar *UDP*, debido a que, al estar el prototipo *DCCP* implementado en Java, no es posible disponer de librerías Standard para el acceso al nivel de red.

Estas no son provistas por java debido a que para algunos, resultaría en un problema de seguridad, mientras que otros argumentan que seria muy útil ya que pondría a java a la altura de otros lenguajes como *C++* en aspectos relativos a aplicaciones de monitoreo de red y similares.

Si bien esta característica impide la comunicación de nuestro prototipo, ya que en nuestro caso el paquete *DCCP* se encapsula en uno *UDP* y este en un *IP*, desde el

punto de vista de la evaluación, no representa mayores problemas ya que *UDP* es un protocolo que no interfiere en modo alguno con el envío de datos. La limitación que surge de esta característica es que se hace imposible comunicar nuestro prototipo *DCCP* con versiones reales.

Capítulo 3

Introducción a los protocolos de internet

3.1. El modelo OSI

3.1.1. Historia

En sus inicios, el desarrollo de redes sucedió con desorden en muchos sentidos. A principios de la década de 1980 se produjo un enorme crecimiento en la cantidad y el tamaño de las redes. A medida que las empresas tomaron conciencia de las ventajas de usar tecnología de networking[6], las redes se agregaban o expandían a casi la misma velocidad a la que se introducían las nuevas tecnologías de red.

En 1977 la *ISO* estableció un subcomité encargado de diseñar una arquitectura de comunicación. El resultado fue el modelo de referencia para la *OSI* adoptado en 1983, que establece unas bases que permiten conectar sistemas abiertos para procesamiento de aplicaciones distribuidas. Se trata de un marco de referencia para definir estándares que permitan comunicar ordenadores heterogéneos.

[6] Networking, consiste en establecer una conexión de red.

3.1.2. Descripción del modelo OSI

Interconexión de Sistemas Abiertos u Open Systems Interconection *OSI*, es usado para describir el uso de datos entre la conexión física de la red y la aplicación del usuario final. Este modelo es el más conocido y el más usado para describir los entornos de red.

Las capas *OSI* están numeradas de abajo hacia arriba. Las funciones más básicas, como el poner los bits de datos en el cable de la red están en la parte de abajo, mientras las funciones que atienden los detalles de las aplicaciones del usuario están arriba.

En el modelo *OSI* el propósito de cada capa es proveer los servicios para la siguiente capa superior, resguardando la capa de los detalles de como los servicios son implementados realmente.

Dicho modelo define una arquitectura de comunicación estructurada en siete niveles verticales. Cada nivel ejecuta un subconjunto de las funciones que se requieren para comunicar con el otro sistema. Para ello se apoya en los servicios que le ofrece el nivel inmediato inferior y ofrece sus servicios al nivel que está por encima de él.

El Modelo *OSI* es un lineamiento funcional para tareas de comunicaciones y, por consiguiente, no especifica un estándar de comunicación para dichas tareas. Sin embargo, muchos estándares y protocolos cumplen con los lineamientos del modelo *OSI*.

OSI nace de la necesidad de uniformizar los elementos que participan en la solución del problema de comunicación entre equipos de cómputo de diferentes fabricantes.

El modelo en sí mismo no puede ser considerado una arquitectura, ya que no especifica el protocolo que debe ser usado en cada capa, sino que suele hablarse de modelo de referencia.

El modelo *OSI* tiene siete capas. Los principios que se aplicaron para llegar a las siete capas son los siguientes:

1. Se debe crear una capa siempre que se necesite un nivel diferente de abstracción[7].
2. Cada capa debe realizar una función bien definida.
3. La función de cada capa se debe elegir pensando en la definición de protocolos estandarizados internacionalmente.
4. Los límites de las capas deben elegirse a modo de minimizar el flujo de información a través de las interfaces[8].
5. La cantidad de capas debe ser suficiente para no tener que agrupar funciones distintas en la misma capa y lo bastante pequeña para que la arquitectura no se vuelva inmanejable.

[7] Niveles de abstracción, es un mecanismo fundamental para la comprensión de problemas y fenómenos.
[8] Interfaces, Zona de contacto, conexión entre dos componentes de "hardware", entre dos aplicaciones o entre un usuario y una aplicación

3.1.3. Capas del modelo OSI

Este modelo está dividido en siete capas:

- ✓ Capa física
- ✓ Capa de enlace de datos
- ✓ Capa de red
- ✓ Capa de transporte
- ✓ Capa de sesión
- ✓ Capa de presentación
- ✓ Capa de aplicación

3.1.3.1. Capa física

La capa física del modelo de referencia *OSI* [6a] es la que se encarga de las conexiones físicas de la computadora hacia la red, tanto en lo que se refiere al medio físico (medios guiados y no guiados) y la forma en la que se transmite la información (codificación de señal, niveles de tensión/intensidad de corriente eléctrica, modulación, tasa binaria, etc.)

Medios guiados: cable coaxial, cable de par trenzado, fibra óptica y otros tipos de cables.

Medios no guiado: radio, infrarrojos, microondas, láser y otras redes inalámbricas); características del medio (tipo de cable o calidad del mismo; tipo de conectores normalizados o en su caso tipo de antena; etc.)

Es la encargada de transmitir los bits de información a través del medio utilizado para la transmisión. Se ocupa de las propiedades físicas y características eléctricas de los diversos componentes; de la velocidad de transmisión, si esta es uni o bidireccional (simplex, dúplex o full-duplex). También de aspectos mecánicos de las conexiones y terminales, incluyendo la interpretación de las señales eléctricas/electromagnéticas.

Se encarga de transformar una trama de datos proveniente del nivel de enlace en una señal adecuada al medio físico utilizado en la transmisión. Estos impulsos pueden ser eléctricos (transmisión por cable); o electromagnéticos.

Sus principales funciones se pueden resumir como:

- ✓ Definir conexiones físicas entre computadoras.
- ✓ Describir el aspecto mecánico de la interfase física.
- ✓ Describir el aspecto eléctrico de la interfase física.
- ✓ Describir el aspecto funcional de la interfase física.
- ✓ Definir la técnica de transmisión.
- ✓ Definir el tipo de transmisión.
- ✓ Definir la codificación de línea.
- ✓ Definir la velocidad de transmisión.
- ✓ Definir el modo de operación de la línea de datos.

3.1.3.2. Capa de enlace de datos

La capa de enlace de datos se ocupa del direccionamiento físico, de la topología de la red, del acceso a la red, de la notificación de errores, de la distribución ordenada de tramas y del control del flujo[9].

Este nivel proporciona facilidades para la transmisión de bloques de datos entre dos estaciones de red. Esto es, organiza los 1's y los 0's del nivel físico[10] en formatos o grupos lógicos de información. Para:

- ✓ Detectar errores en el nivel físico.
- ✓ Establecer esquema de detección de errores para las retransmisiones o reconfiguraciones de la red.
- ✓ Establecer el método de acceso que la computadora debe seguir para transmitir y recibir mensajes. Realizar la transferencia de datos a través del enlace físico.
- ✓ Enviar bloques de datos con el control necesario para la sincronía.
- ✓ En general controla el nivel y es la interfase con el nivel de red, al comunicarle a este una transmisión libre de errores.

3.1.3.3. Capa de red

El cometido de nivel de red es hacer que los datos lleguen desde el origen al destino, aún cuando ambos no estén conectados directamente. Es decir que se encarga de

[9] Control del flujo, corriente de datos que circulan dentro de una conexión de red.
[10] **N**ivel Físico, es la que se encarga de las conexiones físicas de la computadora hacia la red.

encontrar un camino manteniendo una tabla de enrutamiento y atravesando los equipos que sea necesario, para hacer llevar los datos al destino. Los equipos encargados de realizar este encaminamiento se denominan en castellano encaminadores, aunque es más frecuente encontrar el nombre en inglés que significa routers[11] y, en ocasiones enrutadores.

Adicionalmente el nivel de red debe gestionar la congestión de red, que es el fenómeno que se produce cuando una saturación de un nodo tira abajo toda la red.
Este nivel define el enrutamiento y el envío de paquetes entre redes. Es responsabilidad de este nivel establecer, mantener y terminar las conexiones.

Este nivel proporciona el enrutamiento de mensajes, determinando si un mensaje en particular deberá enviarse al nivel 4 (nivel de transporte) o bien al nivel 2 (enlace de datos). Este nivel conmuta, enruta y controla la congestión de los paquetes de información en una sub-red.

3.1.3.4. Capa de transporte

Su función básica es aceptar los datos enviados por las capas superiores, dividirlos en pequeñas partes si es necesario, y pasarlos a la capa de red. En el caso del modelo *OSI*, también se asegura que lleguen correctamente al otro lado de la comunicación.

[11] Routers o enrutadores, dispositivo que conecta dos o más redes

Este nivel actúa como un puente entre los tres niveles inferiores totalmente orientados a las comunicaciones y los tres niveles superiores totalmente orientados al procesamiento. Además, garantiza una entrega confiable de la información.

Otra característica a destacar es que debe aislar a las capas superiores de las distintas posibles implementaciones de tecnologías de red en las capas inferiores, lo que la convierte en el corazón de la comunicación. En esta capa se proveen servicios de conexión para la capa de sesión que serán utilizados finalmente por los usuarios de la red al enviar y recibir paquetes[12].

Este nivel define como direccionar la localidad física de los dispositivos de la red.
Asigna una dirección única de transporte a cada usuario.
Define una posible multicanalización. Esto es, puede soportar múltiples conexiones.
Define la manera de habilitar y deshabilitar las conexiones entre los nodos.
Determina el protocolo que garantiza el envío del mensaje.
Establece la transparencia de datos así como la confiabilidad en la transferencia de información entre dos sistemas.

3.1.3.5. Capa de sesión

Provee los servicios utilizados para la organización y sincronización del diálogo entre usuarios y el manejo e intercambio de datos.

Esta capa ofrece varios servicios que son cruciales para la comunicación, como son:

[12] Paquetes, conjunto de estructurado de bytes que forma la unidad básica de comunicación..

1.- Control de la sesión a establecer entre el emisor y el receptor (quién transmite, quién escucha y seguimiento de ésta).

2.- Control de la concurrencia (que dos comunicaciones a la misma operación crítica no se efectúen al mismo tiempo).

3.- Mantener puntos de verificación o (*checkpoints*), que sirven para que, ante una interrupción de transmisión por cualquier causa, la misma se pueda reanudar desde el último punto de verificación en lugar de repetirla desde el principio.

Por lo tanto, el servicio provisto por esta capa es la capacidad de asegurar que, dada una sesión establecida entre dos máquinas, la misma se pueda efectuar para las operaciones definidas de principio a fin, reanudándolas en caso de interrupción. En muchos casos, los servicios de la capa de sesión son parcialmente, o incluso, totalmente prescindibles.

3.1.3.6. Capa de presentación

La capa de presentación realiza ciertas funciones que se piden con suficiente frecuencia para justificar la búsqueda de una solución general; en lugar de dejar que cada usuario resuelva los problemas. En particular, y a diferencia de todas las capas inferiores que se interesan solo en mover bits de manera confiable de acá para allá, la capa de presentación se encarga de la sintaxis y la semántica[13] de la información que se transmite.

[13] Semántica, es un término que hace referencia al significado de las palabras.

Establece independencia a los procesos de aplicación considerando las diferencias en la representación de datos. Proporciona servicios para el nivel de aplicaciones al interpretar el significado de los datos intercambiados.

Por lo tanto, podemos resumir definiendo a esta capa como la encargada de manejar las estructuras de datos abstractas y realizar las conversiones de representación de datos necesarias para la correcta interpretación de los mismos. Esta capa también permite cifrar los datos y comprimirlos.

3.1.3.7. Capa de aplicación

La capa de presentación contiene varios protocolos que se necesitan con frecuencia. Esta capa proporciona servicios al usuario acerca del modelo *OSI* [6a] y la comunicación entre dos procesos de aplicación, tales como: programas de aplicación, aplicaciones de red, etc.

Aporta aspectos de comunicaciones para aplicaciones específicas entre usuarios de redes: manejo de la red, Protocolo de Transferencia de Ficheros o File Transfer Protocol (*FTP*), etc.

Ofrece a las aplicaciones (de usuario o no) la posibilidad de acceder a los servicios de las demás capas y define los protocolos que utilizan las aplicaciones para intercambiar datos, como correo electrónico Post Office Protocol (*POP*) y Protocolo Simple de Transferencia de Correo Electrónico o Simple Mail Transfer Protocol (*SMTP*), gestores de bases de datos y servidor de ficheros (*FTP*). Hay tantos protocolos como

aplicaciones distintas y puesto que continuamente se desarrollan nuevas aplicaciones el número de protocolos crece sin parar.

Cabe aclarar que el usuario normalmente no interactúa directamente con el nivel de aplicación. Suele interactuar con programas que a su vez interactúan con el nivel de aplicación pero ocultando la complejidad subyacente.

Nivel	**Nombre**	**Categoría**
Capa 7	Nivel de aplicación	Aplicación
Capa 6	Nivel de presentación	
Capa 5	Nivel de sesión	
Capa 4	Nivel de transporte	
Capa 3	Nivel de red	Transporte de datos
Capa 2	Nivel de enlace de datos	
Capa 1	Nivel físico	

Tabla 1: Capas del modelo OSI

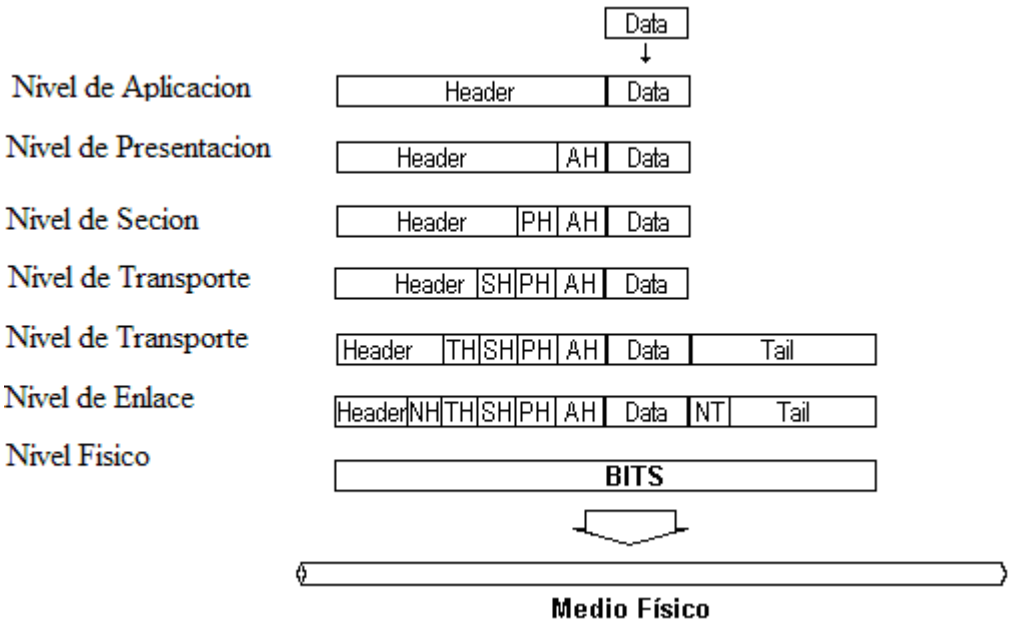

Gráfico 1: Transmisión de datos en el modelo *OSI*

3.2. Modelo de referencia TCP/IP

Las familias de protocolos *TCP/IP* permiten la comunicación entre diferentes tipos de ordenadores con independencia del fabricante, red a la que se encuentren conectados y al sistemas operativo utilizado.

Las familias de protocolos que gobiernan Internet, se caracterizan por haber sido construidos siguiendo un esquema de capas (layers). Cada capa es la responsable de cada una de las diferentes fases de la comunicación. De esta forma, se puede definir la familia de los protocolos *TCP/IP* como una combinación de cuatro capas. En este esquema la capa superior accede únicamente a los servicios prestados por la capa situada justo en el nivel inferior a ella.

APLICACIÓN (Application)	WWW,FTP…
TRANSPORTE (Transport)	TCP,UDP…
RED (Network)	IP
ENLACE (Link)	EEE 802.2, 802.3…

Tabla 2: Estructura de cuatro capas

3.2.1. Capas del TCP/IP

3.2.1.1. Capa de enlace (Link Layer)

También denominada capa de datos (data layer) o capa de acceso a la red (network interfase layer), incluye los mecanismos que permiten al sistema operativo enviar y recibir información a través de la red a la que esta conectado (Ethernet, RDSI…).

3.2.1.2. Capa de red (Network Layer)

También denominada capa de Internet (Internet layer), es la encargada de mover los paquetes a través de las diferentes redes para llegar a su destino. En esta capa encontramos los protocolos de más bajo nivel, destacando *IP*.

3.2.1.3. Capa de transporte (Transport Layer)

Es la encargada de proporcionar un flujo de datos entre dos ordenadores. Este flujo de datos puede ser fiable, *TCP* o no fiable, *UDP*.

3.2.1.4. Capa de aplicación (Application Layer)

Es la encargada de manejar los detalles particulares relativos a las diferentes aplicaciones (*WWW*, *TELNET*, *FTP*…).

Siguiendo el modelo de casa descrito anteriormente, vemos que la combinación entre dos ordenadores no se produce directamente. De esta forma, cada capa añade una información de control específica denominada cabecera (“*header”*) a los datos recibidos y los pasa a la capa inferior. Este proceso se repite en cada capa, hasta llegar a la capa de enlace, dónde se envían los datos por la red.

Análogamente, cuando se recibe una información, la capa receptora elimina la cabecera y pasa el resultado a la capa superior.

Este sistema permite una independencia entre las diferentes capas y obliga a que la comunicación entre los dos ordenadores se realice mediante una comunicación entre las capas de los dos ordenadores.

3.3. Diferencias entre OSI y TCP/IP

Los modelos de referencia *OSI* y *TCP/IP* tienen mucho en común. Ambos se basan en el concepto de un gran número de protocolos independientes. También la funcionalidad de las capas es muy similar. Por ejemplo, en ambos modelos las capas por encima de la de trasporte, incluido esta, están ahí para presentar un servicio de transporte de extremo a extremo, independiente de la red, a los procesos que deseen comunicarse.

OSI y *TCP/IP* muestra un intento de establecer una correspondencia entre las diferentes capas de las arquitecturas de *OSI* y *TCP/IP*, pero hay que ser consciente de las diferencias básicas explicadas más abajo.

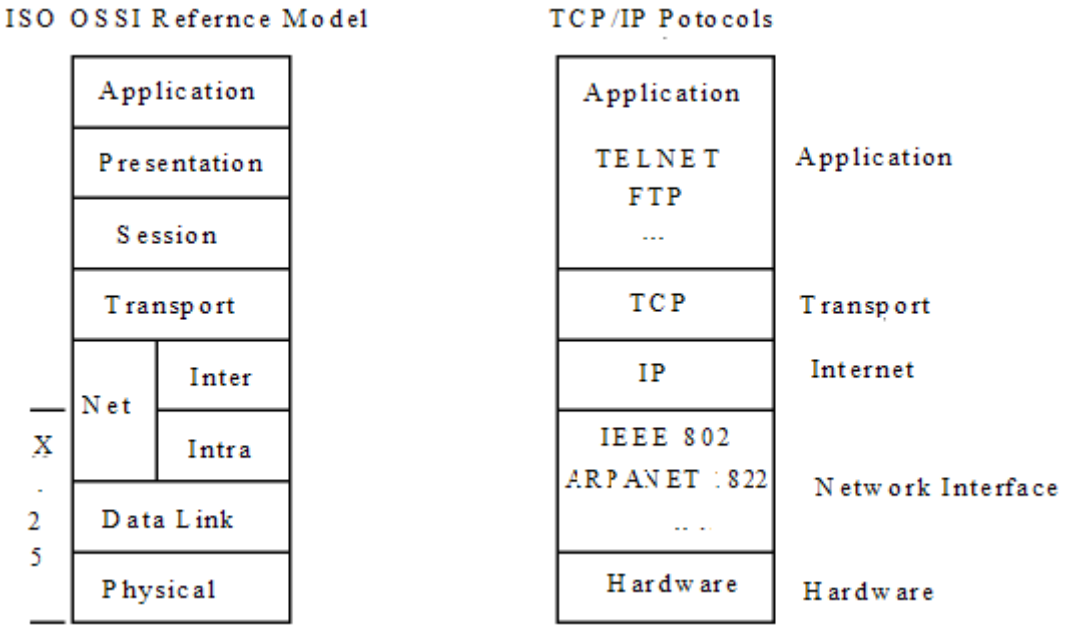

Gráfico 2: *OSI* y *TCP/IP* - Correspondencia funcional de las capas.

El modelo de Internet sólo puede equipararse funcionalmente al modelo *OSI* de *ISO*, ya que existen diferencias básicas tales como:

En la pila de protocolos de Internet, una capa representa un encapsulamiento de una función.

La perspectiva de *ISO*, por otro lado, trata a las capas como grupos funcionales bastante reducidos, intentando forzar la modularidad al requerir capas adicionales para funciones adicionales.

En los protocolos *TCP/IP*, señala que un protocolo dado puede ser usado por otros protocolos en la misma capa, mientras que en el modelo *OSI* se definiría dos capas en las mismas circunstancias. Ejemplos de estas "dependencias horizontales" son *FTP*, que usa la misma representación común que (*telnet*)[14] sobre la capa de aplicación, o El Protocolo de Control de Mensajes de Internet o Internet Control Message Protocol (*ICMP*), que usa *IP* para el envío de datagramas en el nivel de red.

A nivel práctico, lo que estamos discutiendo aquí es la diferencia entre un estándar "de jure", *OSI*, y uno "de facto", *TCP/IP*. El objetivo en el mundo de *TCP/IP* consiste en establecer de común acuerdo un protocolo estándar que pueda funcionar en una diversidad de redes heterogéneas; siempre se le ha dado mayor importancia al estándar en sí que a su implementación.

Eficiencia y viabilidad. Las normas de *OSI* tienden a ser prescriptitas (por ejemplo, la capa "*N*" debe atravesar todas las capas "por debajo" de ella), mientras que los protocolos *TCP*/*IP* tienden a ser descriptivos, y dejan un máximo de libertad a los implementadotes. Una de las ventajas del enfoque de *TCP/IP* es que cada

[14] **Telnet,** es una aplicación que permite desde nuestro sitio y con el teclado y la pantalla de nuestra computadora

implementación concreta puede explotar características dependientes del sistema, de lo que suele derivarse una mayor eficiencia (menos ciclos de la Unidad Central de Proceso o Central Procesor Unit "*CPU"*, mayor productividad para las mismas funciones), al mismo tiempo que se asegura la interoperabilidad con otras aplicaciones.

En general *OSI* es un estándar de referencia y consta de siete niveles los cuales no suelen implementarse todos.

Y *TCP/IP* suele explicarse en 4 niveles:

Acceso, que permite que los datagramas *IP* [RFC791] viajen por un determinado medio.

IP, es el protocolo de nivel 3, no orientado a la conexión.

TCP, Es el protocolo de transporte confiable y orientado a la conexión.

UDP, Es el protocolo de transporte ni fiable ni orientado a la conexión.

Aplicación, Engloba a cualquier usuario del nivel *TCP* o *UDP*.

3.4. Protocolo IP

El protocolo *IP* [5] es la pieza fundamental en la que sustenta el sistema *TCP/IP* y por tanto todo el funcionamiento de Internet. Su especificación esta recogida en [RFC791]. La unidad de datos del protocolo *IP* es el datagrama.

El protocolo *IP* facilita un sistema sin conexión (*connectionless*) y no fiable (*unreliable*) de entrega de datagramas entre dos ordenadores cualquiera conectados a Internet. *IP* da un servicio de entrega basado en el mejor intento (best enffort).

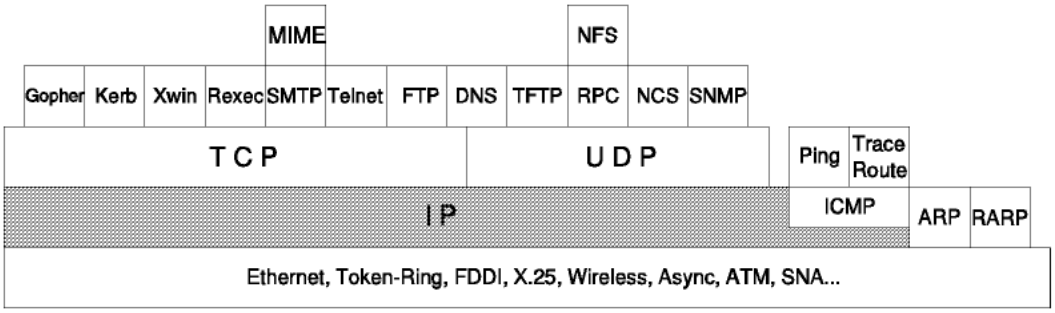

Gráfico 3: Estructura de un datagrama IP

3.5. Protocolos de red

Para [LA02] explica que el termino protocolo lo usamos para describir el intercambio de información entre procesos, es el conjunto de reglas que especifican el intercambio de datos u órdenes durante la comunicación entre las entidades que forman parte de una red.

Los protocolos pueden estar implementados bien en hardware (*tarjetas de red*), software (*drivers*), o una combinación de ambos.

Al hablar de protocolos no se puede generalizar, debido a la gran amplitud de campos que cubren, tanto en propósito, como en especificidad. No obstante, la mayoría de los protocolos especifican una o más de las siguientes propiedades:

Detección de la conexión física sobre la que se realiza la conexión (*cableada o sin cables*)

Los protocolos que son implementados en sistemas de comunicación que tienen un amplio impacto, suelen convertirse en estándares, debido a que la comunicación e intercambio de información (*datos*) es un factor fundamental en numerosos sistemas, y

para asegurar tal comunicación se vuelve necesario copiar el diseño y funcionamiento a partir del ejemplo pre-existente. Esto ocurre tanto de manera informal como deliberada.

3.6. Funciones principales de los protocolos

- ✓ Control de errores
- ✓ Control de Flujo y Congestión
- ✓ Estrategias de encaminamiento

Control de Errores:

Protege integridad de los datos del usuario y de los mensajes de control.

Control de Flujo y Congestión:

Permite a la red compartir sus recursos entre un gran número de usuarios, entregando a cada uno un servicio satisfactorio sin que sus operaciones corran peligro.

Estrategias de Encaminamiento:

Permite optimizar la utilización de los recursos de la red, aumentando la disponibilidad de los servicios de la red al proveer caminos alternativos entre nodos terminales.

3.7. Datagramas

Un datagrama es un fragmento de paquete que es enviado con la suficiente información como para que la red pueda simplemente encaminar el fragmento hacia el ordenador receptor, de manera independiente a los fragmentos restantes.

Esto puede provocar una recomposición desordenada o incompleta del paquete en el ordenador destino.

La estructura de un datagrama es: cabecera y datos.

Los datagramas tienen cabida en los servicios de red no orientados a la conexión o Datagrama Agrupación lógica de información que se envía como una unidad de capa de red a través de un medio de transmisión sin establecer con anterioridad un circuito virtual.

Es un paquete autosuficiente (*análogo a un telegrama*) el cual contiene información suficiente para ser transportado a destino sin necesidad de, previamente, establecer un circuito.

No se provee confirmación de recepción por el destinatario, pero puede existir un aviso de no entrega por parte de la red.

3.8. Niveles de los procesos de comunicación

Un par de procesos no necesita conocer la estructura interna de su sistema de comunicaciones, sólo se comunica con el, a través de una interfaz. P1[15] y P2[16] podrían ser capaces de soportar varios procesos simultáneamente, proporcionando funciones de multiplexación.

Si P3[17]-P1 no están instalados en el mismo procesador, deberá usarse un protocolo para implementar la interfaz.

Además si existe una línea física (*P3-P1*), propensa a errores, se deberá usar un protocolo a nivel de línea para asegurar la corrección de los mensajes intercambiados por los procesos. Este protocolo no afectará la estructura general, solo reemplazará una conexión directa.

3.9. Jerarquías de protocolos

Cuando los dispositivos que van a intervenir en la comunicación son ordenadores hace falta un alto grado de cooperación entre ellos. Por ejemplo. En una transmisión de ficheros, además de las tareas usuales relacionadas con la comunicación de datos, es necesario asegurarse que el sistema destino está listo para recibir datos, para aceptar y almacenar el fichero, si el formato de ficheros que usan lo dos sistemas es incompatible uno u otro debe realizar una transformación, etc.

[15] P1, proceso primero o 1.
[16] P2, proceso segundo o 2.
[17] P3, proceso tercero o 3.

3.10. Arquitectura de comunicación

Los protocolos se utilizan para la comunicación entre entidades de diferentes sistemas. Los términos entidad y sistema se utilizan aquí en un sentido muy general. Ejemplos de entidades son programas de aplicación de usuario, paquetes de transferencia de ficheros, sistemas de manejo de bases de datos[18] y terminales.

Ejemplos de sistemas son ordenadores, terminales y sensores remotos. En general, una entidad es algo capaz de enviar o de recibir información, y un sistema es un objeto que contiene una o más entidades. Para que dos entidades puedan comunicarse deben hablar el mismo idioma.

Qué se comunica, cómo se comunica y cuándo se comunica debe cumplir ciertas convenciones entre las entidades involucradas. Este conjunto de convenciones constituye un protocolo, que puede definirse como el conjunto de reglas que gobiernan el intercambio de datos entre dos entidades.

La tarea de la comunicación entre dos entidades de diferentes sistemas es demasiado complicada para ser manejada por un simple proceso o módulo. En lugar de manejar un único protocolo, implementaremos las funciones de comunicación mediante un conjunto de protocolos estructurados.

La organización de estos protocolos se realiza mediante una serie de capas o niveles, con objeto de reducir la complejidad de sus diseños. El número de capas, el nombre,

[18] Bases de datos, conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso

contenido y función de cada una varían de una red a otra. Sin embargo, en cualquier red, el propósito de capa es ofrecer ciertos servicios a las capas superiores, liberándolas del conocimiento detallado sobre cómo se realizan dichos servicios.

3.11. Protocolo TCP (Protocolo de Control de Transferencia)

El (*TCP*) descrito en el [RFC793] dice que un protocolo de red de la capa de transporte, orientado a la conexión que proporciona el envió de grandes cantidades de información de una manera fiable, ejecuta control de congestión, lo que impide que el emisor introduzca demasiado trafico en la red; *TCP* es un protocolo punta a punta, esto es, provee un servicio de transmisión entre dos *hosts*. Por esta razón, el *TCP* se mantiene como la capa de transporte predominante en la Internet; sin embargo pierde eficiencia y es incapaz de satisfacer los requerimientos de flujos en tiempo real.

3.12. Protocolo UDP (Protocolo de Datagrama del Usuario)

El (*UDP)* es un protocolo de transporte sin conexión, no fiable, no porque sea particularmente malo, sino porque no verifica que los paquetes lleguen a su destino, y no da garantías de que lleguen en orden. *UDP* es usado normalmente para el tipo de aplicaciones de tiempo real que hemos mencionado (*audio, video, etcétera*) donde la llegada a tiempo de los paquetes es más importante que la fiabilidad, o para aplicaciones simples de tipo petición/respuesta, es más rápido que *TCP* pero menos confiable. Si bien el ritmo de transmisión es controlable por la aplicación, no se provee control de congestión, lo que puede producir saturación de los recursos de red, ni se provee

conexión, característica deseable en flujos de una duración considerable, como los producidos por las aplicaciones mencionadas.

Como consecuencia de lo anterior, surge la necesidad de contar con un protocolo a nivel transporte que se adapte al tipo de aplicaciones que nos ocupa, reuniendo las características deseables de *TCP* y de *UDP*.

3.13. Protocolo DCCP (Protocolo de Control Congestión del Datagrama)

DCCP es un protocolo de nivel de transporte que tiene la capacidad de una entrega rápida de datos que mantiene un mecanismo de control de congestión, *DCCP* está pensado para aplicaciones que no necesitan la entrega en orden ni la confiabilidad que ofrece *TCP*; hasta la fecha con la aparición de este estándar la mayoría de las aplicaciones han estado usando *TCP*, con los problemas arriba mencionados, o en su defecto han estado usando, *UDP* no implementa control de congestión, la motivación original para el desarrollo del *DCCP* es proveer a este tipo de aplicaciones una vía para acceder a los mecanismos de control de congestión sin tener que implementarlos en el nivel de aplicación.

Capítulo 4

Protocolo del control de la congestión del datagrama (DCCP)

4.1. Introducción protocolo DCCP

El protocolo del control de la congestión del datagrama (*DCCP*) es un nuevo protocolo de transporte diseñado por el (*IETF*). El bosquejo (*draft-ietf-dccp-spec-03.txt*), escrito por KOHLER. E, HANDLEY. M, FLOYD. S, & J PADHYE especificado en mayo de 2003.

DCCP fue publicada como [RFC 4340], en la categoría de estándar propuesto, por el *IETF* en marzo del 2006, esta en la fase de diseño y de prueba en este momento.
DCCP es un nuevo protocolo del transporte que esta diseñado por el *IETF* sustituir el *UDP* como el protocolo que se utilizará transportar tráfico en tiempo real, pero no limitado a este tipo de tráfico.

Entre las aplicaciones implementar el *DCCP* se incluyen aquellas que tienen necesidad de entrega rápida de datos, y que combinada con la implementación de algún método de control de congestión, resulta generalmente en el arribo extemporáneo de la

información, convirtiéndose en inútil. Esta categoría de aplicaciones incluye la telefonía en Internet y multimedia en tiempo real, entre otras. El control de congestión es la forma en que el protocolo de red descubre la capacidad disponible de la red para una ruta en particular. La motivación original.

Para el desarrollo del *DCCP* es proveer a este tipo de aplicaciones una vía para acceder a los mecanismos estándar de control de congestión sin tener que implementarlos en el nivel de aplicación.

DCCP está pensado para aplicaciones que requieren un estudio basado en el flujo de *TCP* pero no necesitan la entrega en orden ni la confiabilidad que ofrece *TCP*, o que quieren un control de congestión dinámico distinto del de *TCP*.

Hasta la fecha de aparición de este estándar, la mayoría de las aplicaciones han estado usando *TCP*, con los problemas arriba mencionados, o en su defecto han estado usando *UDP* implementando sus propios mecanismos de control de congestión, o no implementando ninguno.

El propósito de *DCCP* es proveer a las aplicaciones que utilizan una red, de una metodología uniforme para la negociación del control de congestión, y la implementación de tal mecanismo.

4.2. Terminología

Las palabras claves "*MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", y "OPTIONAL"* en esta trabajo de investigación serán interpretadas según lo descrito dentro del [RFC2119].

4.2.1. Anatomía de una conexión de DCCP

Según lo descrito en el [RFC4340] se dice que cada conexión de *DCCP* funciona entre dos anfitriones o (*host*), que a menudo nombramos "*DCCP A"* y "*DCCP B"*. Cada conexión es iniciada activamente por uno de los anfitriones (*host*), que llamamos el cliente; el otro, inicialmente anfitrión pasivo lo llamaremos servidor. El término "*punto final de DCCP*" lo utilizaremos para referirnos a cualquiera de los dos anfitriones nombrados explícitamente por la conexión "*el cliente y el servidor".* El término "*DCCP procesor*" se refiere más generalmente a cualquier anfitrión o (*host*) que pudiera necesitar procesar la cabecera de *DCCP*.

Las conexiones de *DCCP* son bidireccionales y los datos pueden pasar de cualquier *punto final* al otro. Esto significa que los datos y los reconocimientos pueden fluir en ambas direcciones simultáneamente, sin embargo, una conexión de *DCCP* consiste en dos conexiones unidireccionales separadas, que las llamaremos la "*mitad de conexión"*. Cada *mitad de conexión* consiste en los datos de la aplicación enviadas por un *punto final* y que corresponden a los reconocimientos enviados por el *otro punto final*.

En el contexto de una sola *mitad de conexión*, los términos "*HC-Remitente*" y el "*HC-Receptor*" denotan los *puntos finales* que envían datos y reconocimientos de la aplicación, respectivamente.

La conexión de *DCCP* entre *DCCP A* y *DCCP B* es ilustrada a continuación en el Gráfico 4.

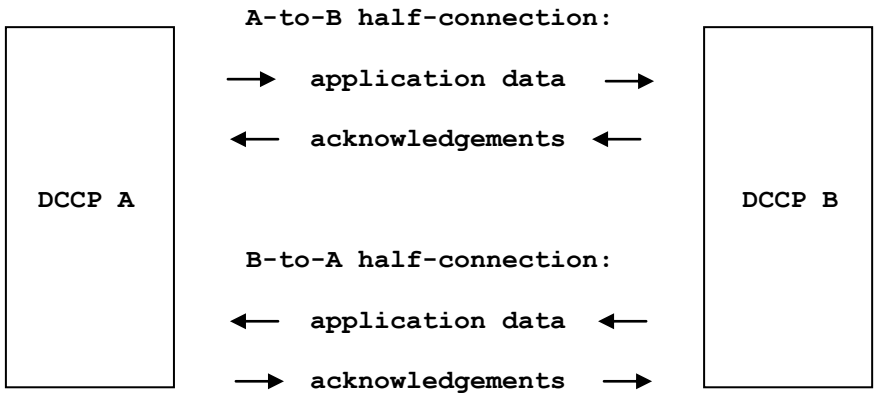

Gráfico 4: Anatomía de una conexión de DCCP

4.2.2. Características

Dentro de una conexión los dos puntos finales se ponen de acuerdo o convienen entre si, esta es una característica muy marcada dentro del *DCCP*.

Algunas propiedades de la conexión *DCCP* son controladas por estas características, incluyendo los mecanismos de control de congestión, los *puntos finales* alcanzan un acuerdo con el intercambio de opciones dentro de esta negociación.

Las características de *DCCP* son identificadas por un número de característica. La notación "*F/X*" representa la característica con el número *F* situado en la punto final *X*. de *DCCP*. Cada número válido corresponde a dos características, que se negocian por separado y no necesitan tener el mismo valor. Los dos *puntos finales* acceden al valor de cada característica válida.

DCCP A es la "*características de localización*" para todas las características *F/A*, y la "*característica remota*" para todas las características *F/B*.

4.2.3. Tiempos de ida y vuelta o RTT

El [RFC4340] describe, que los "tiempos de ida y vuelta" o (*RTT*) del *DCCP* son realizados por los mecanismos de control de la congestión, los diversos mecanismos pueden medir los *tiempos de ida y vuelta* de diversas maneras, o no medirlos. Sin embargo, el protocolo *DCCP* utiliza *tiempos de ida y vuelta* de vez en cuando, por ejemplo en los valores iniciales para ciertos contadores de tiempo. Cada implementación en *DCCP* es definida cuando no hay estimación disponible. Este parámetro debe omitir no menos que **0.2 segundos**, en un *tiempo de ida y vuelta* para las conexiones *TCP* en el Internet.

El comportamiento del protocolo especificado en términos de valores del *tiempos de ida y vuelta* se refiere realmente a una estimación actual del *tiempo de ida y vuelta* tomados por algún *CCID*, o, si no hay estimaciones disponibles, el parámetro del *tiempo de ida y vuelta* será tomado por defecto.

El tiempo de la vida máximo de un segmento, o (*MSL*), es la longitud del tiempo máxima que un paquete puede sobrevivir en la red. El *DCCP MSL* debe igualar al del *TCP*, que es normalmente de **2** minutos.

4.2.4. Principio de la robustez

El [RFC4340] muestra que, las implementaciones del protocolo *DCCP* seguirán "*el principio general de robustez*" del *TCP* [RFC793] "*ser conservador en lo que haces, y ser liberal en lo que aceptas de otros".*

4.2.5. Números y campos

El [RFC4340] menciona que, todas las cantidades numéricas del *multi-byte* en *DCCP*, tales como números de acceso, números de secuencia, opciones y argumentos, son transmitidas en orden dentro de la red ("el primer byte es el mas significativo"). Nosotros ocasionalmente nos referimos a la posición de un byte de izquierda a derecha. Mostrando el bit mas significativo desde la izquierda, y el byte menos significativo hacia la derecha.

Los números al azar o randómicos en *DCCP* se utilizan para definir las características de la seguridad y se deben elegir según los parámetros definidos en el [RFC4086].
Todas las operaciones con números de secuencia en *DCCP* utilizan el módulo aritmético circular **2^48**, al igual que las comparaciones tales como "*mayor*" y "*menor que".*

Esta forma de aritmética guarda relación entre los números de secuencia, los mismos que se encuentran en el rango desde **2^48 - 1 a 0.**

La implantación de los números de secuencia de *DCCP* se asemejará a los números de secuencia del *TCP* y los números de serie del nombre del Dominio del Servicio o Domain Name Service (*DNS)* descritos en el [RFC1982].

4.3. Formatos del paquete

El [RFC4340] menciona que, la *cabecera* del *DCCP* puede tener de **12 a 1200** bytes de longitud, la parte inicial de la *cabecera* tiene la misma *semántica* para todos los tipos de paquete definidos, luego se define cualquier campo de longitud fija adicional requerido por el tipo del paquete.

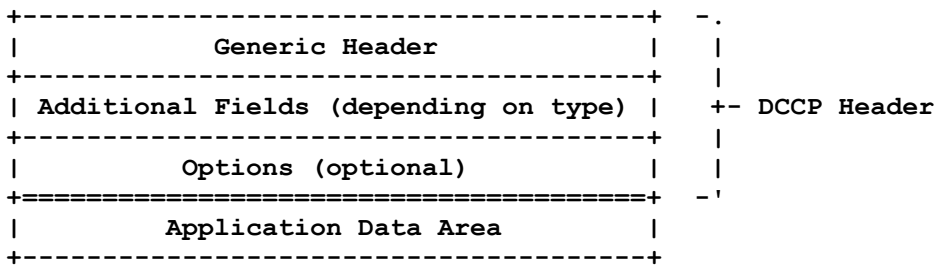

Gráfico 5: Cabecera del DCCP

4.3.1. Cabecera genérica DCCP.

El [RFC4340] describe que, la *cabecera genérica* de *DCCP* toma diversas formas dependiendo del valor de *X*, el bit mas largo de los números de serie. Si *X* es **1**, el campo del número de secuencia es de 48 bits de longitud, y la *cabecera* toma 16 bytes, como se muestra a continuación en el gráfico 6:

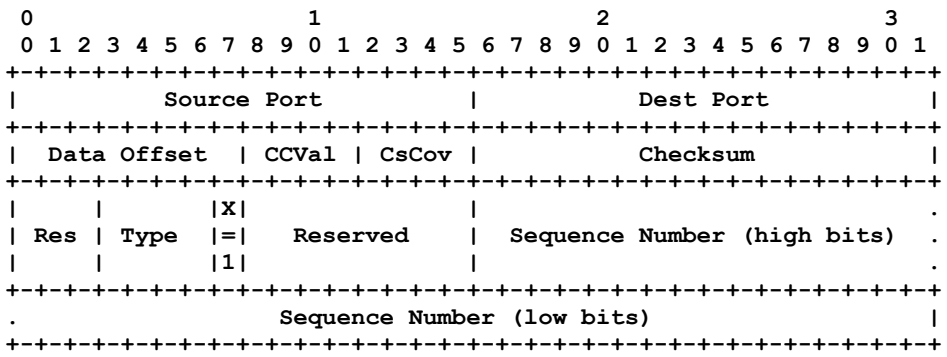

Gráfico 6: Cabecera Genérica cuando X= 0

Si *X* es **0**, solamente los **24** bits mas pequemos de los números de secuencia son transmitidos y la *cabecera* tiene una longitud de **12** bytes.

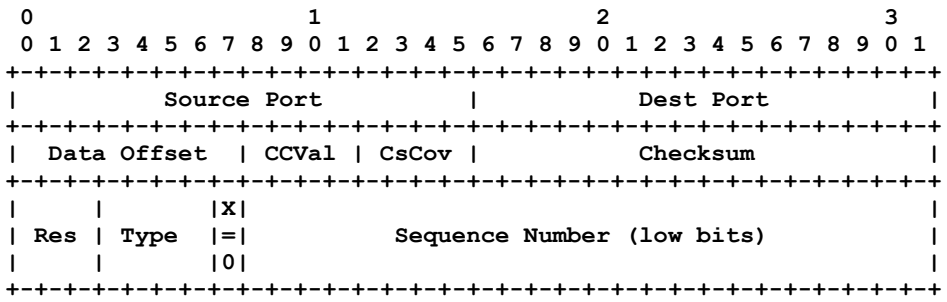

Gráfico 7: Cabecera Genérica cuando X= 1

4.3.2. Tipos de Paquete.

El [RFC4340] describe que, para implementar las funciones de este protocolo se utilizaran diez tipos de paquetes, los mismos serán detallados a continuación.

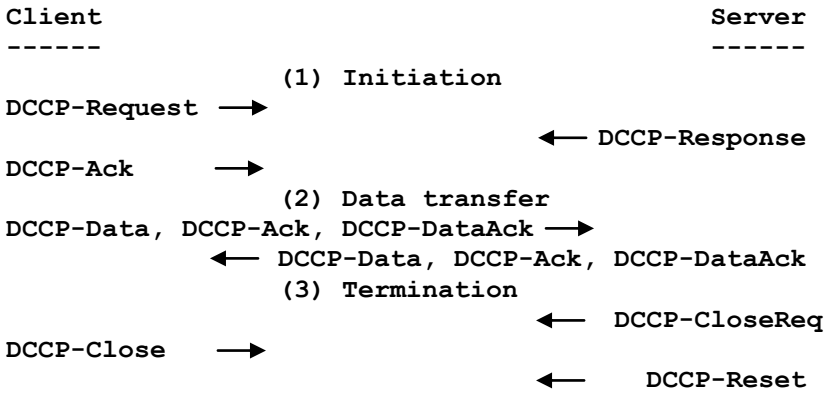

Gráfico 8: Tipos de Paquetes DCCP

3.3.2.1. DCCP-Request o DCCP para Solicitud.

Es la solicitud que envía el cliente para iniciar una conexión, esta es la primera parte del "*three-way handshake*[19]".

Este paquete lo envía el cliente al iniciar una conexión de *DCCP* enviando paquete de respuesta o ("*DCCP-Request*"). Estos paquetes pueden contener datos de aplicación y utilizan números de secuencia de **48** bits (*X=1*).

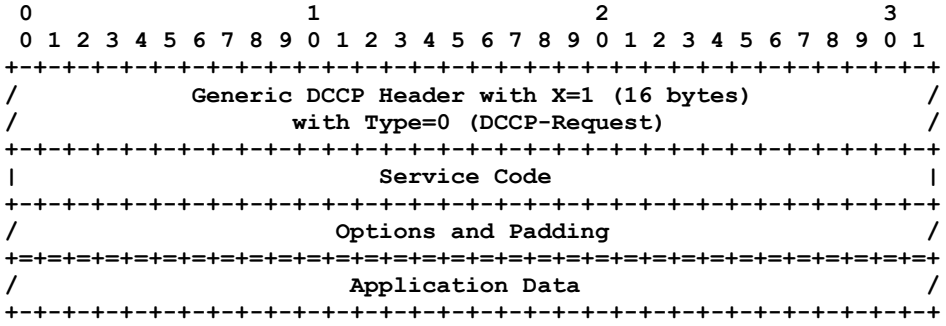

Gráfico 9: DCCP-Request o DCCP para Solicitud

[19] three-way handshake, comienzo de una conexión segura en el protocolo.

4.3.2.2. DCCP-Response o DCCP de Respuesta.

Este paquete lo envía el servidor en respuesta a un paquete de petición ("*DCCP-Request*"), es la segunda parte del *three-way handshake.*

Los paquetes de respuesta o ("*DCCP-Response*"), pueden contener datos de la aplicación y deben utilizar números de serie de **48** bits (*X=1*).

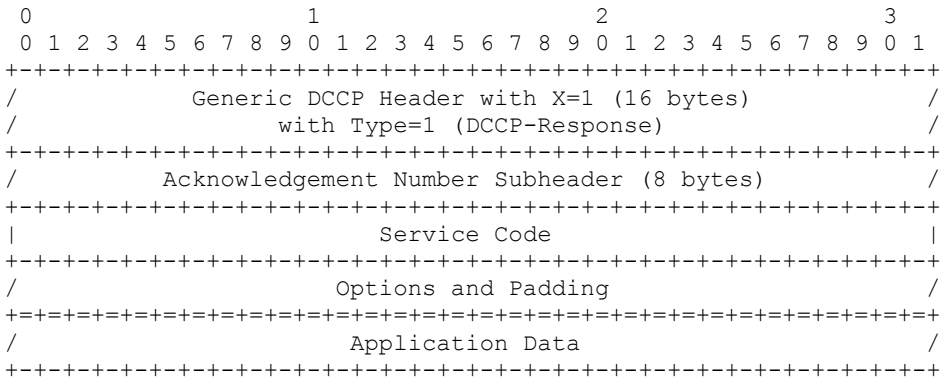

Gráfico 10: DCCP-Response o DCCP de Respuesta

4.3.2.3. Paquetes DCCP-Data, DCCP-Ack, y DCCP-DataAck

La transferencia de datos de cada conexión de *DCCP* utiliza los paquetes de datos o ("*DCCP-Data*"), paquetes de reconocimiento o ("*DCCP-Ack*"), y paquetes de datos con reconocimientos o ("*DCCP-DataAck*"). Estos paquetes pueden utilizar números de secuencia de **24** bits, dependiendo del valor de los números de secuencia. Los paquetes de los *DCCP-Data* llevan datos de aplicación sin reconocimientos.

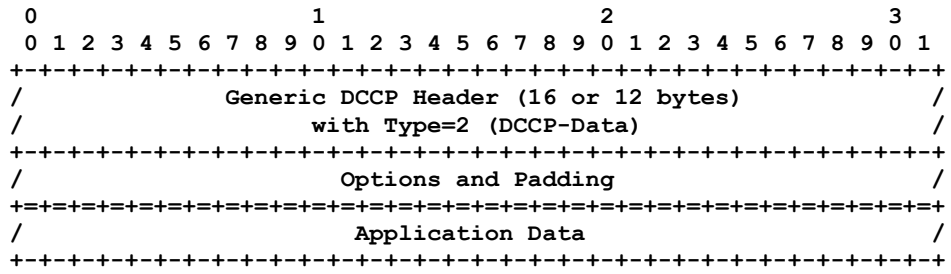

Gráfico 11: Paquetes DCCP-Data

Los paquetes *DCCP-Ack* son usados para la transmisión datos en una aplicación y son usados para reconocimientos puros.

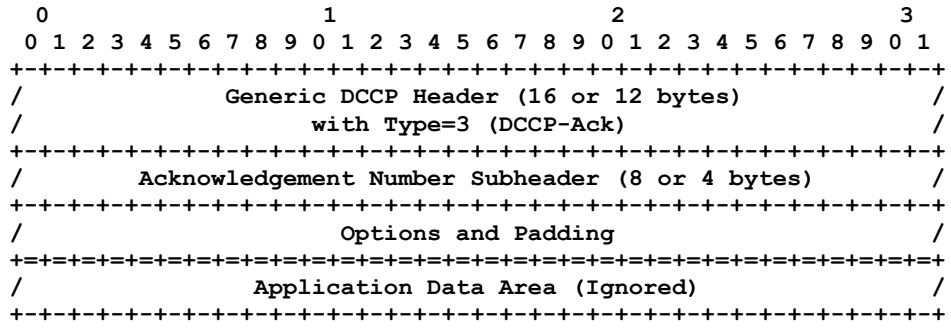

Gráfico 12: Paquetes DCCP-Ack

Los paquetes de *DCCP-DataAck* llevan datos de aplicación y un número del reconocimiento. Estos contienen la información del reconocimiento sobre un paquete de datos.

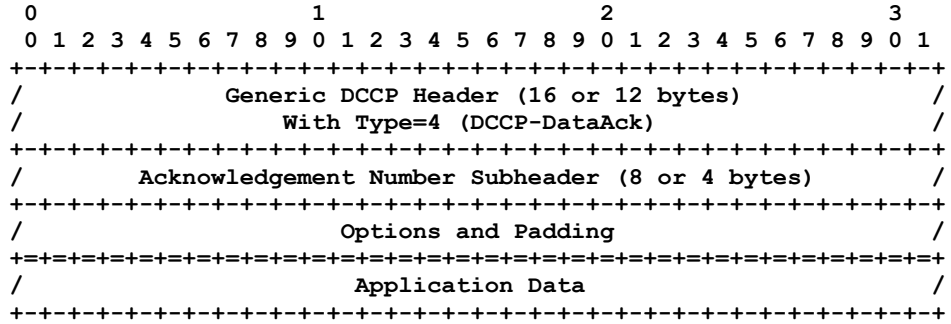

Gráfico 13: Paquetes DCCP-DataAck

Un paquete *DCCP-Data* o *DCCP-DataAck* pueden tener un área de datos de aplicación con longitud **0**, que indican que la aplicación envió un datagrama de longitud **0.** Esto los diferencia de los paquetes *DCCP-Request* y *DCCP-Respond*, donde un área de datos vacía de la aplicación indica la ausencia de los datos dentro de la misma.

La Interfaz de Programación de Aplicaciones o Application Programming Interface *(API)* debe generalizar cualquier datagrama recibido con longitud **0**. Un paquete *DCCP-Ack* puede tener un área de datos de longitud distinta de **0**, que esencialmente completa el paquete *DCCP-Ack* a una longitud deseada. Los receptores deben pasar por alto el contenido del área de datos en los paquetes *DCCP-Ack*. Los paquetes *DCCP-Ack* y *DCCP-DataAck* incluyen a menudo adicionales opciones del reconocimiento, tales como el "*vector del Ack*", según los requisitos del mecanismo del control de congestión que se encuentre en funcionamiento.

4.3.2.4. DCCP-CloseReq y DCCP-Close

Los paquetes de cierre de respuesta o ("*DCCP-CloseReq*") y los paquetes de cierre de conexión ("*DCCP-Close*") comienzan y terminan normalmente una conexión. El cliente o el servidor pueden enviar el paquete *DCCP-Close*, que obtendrá un paquete de reinicio o ("*DCCP-Reset*"). Solamente el servidor puede enviar un paquete de *DCCP-CloseReq*, que indica que el servidor desea cerrar la conexión pero no desea llevar a cabo su estado de tiempo de espera ("*TIMEWAIT*"). Ambos tipos del paquete deben utilizar números de secuencia de **48** bits (*X=1*).

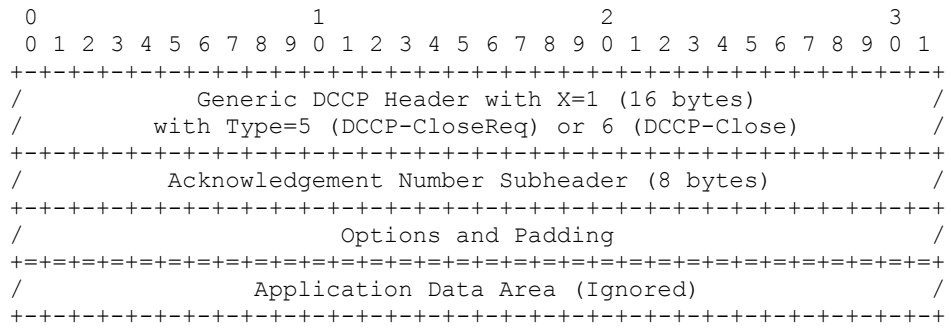

Gráfico 14: Paquetes DCCP-CloseReq y DCCP-Close

Como con los paquetes *DCCP-Ack*, *DCCP-CloseReq* y *DCCP-Close* pueden tener áreas de datos de aplicación de longitud distinta de **0**, en este caso el receptor debe pasarlos por alto.

4.3.2.5. Paquetes del DCCP-Reset

Los paquetes del *DCCP-Reset* cierran una conexión. Las conexiones terminan normalmente con un *DCCP-Reset*, pero los reinicios se pueden enviar por otras razones, incluyendo malos números de acceso. Los *DCCP-Reset* deben utilizar números de secuencia de **48** bits respectivamente (*X=1*).

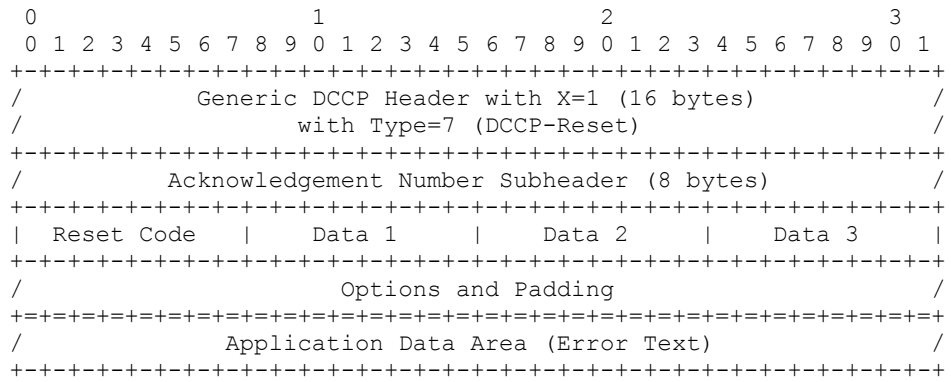

Gráfico 15: Paquetes del DCCP-Reset

4.3.2.6. DCCP-Sync, DCCP-SyncAck

Son utilizados para resincronizar los números de serie después de grandes pérdidas de datos. Ambos tipos del paquete deben utilizar números de secuencia de **48** bits respectivamente (*X=1*).

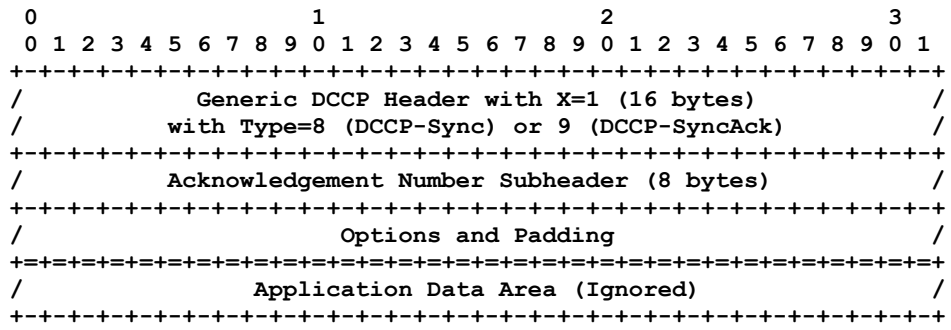

Gráfico 16: DCCP-Sync, DCCP-SyncAck

4.3.3. Estados

El [RFC4340] describe que, los *puntos finales* de *DCCP* están definidos por diversos estados durante el curso de una conexión, correspondiendo al *three-way handshake* de iniciación, transferencia de datos, y terminación. La figura de abajo demuestra el progreso típico a través de estos estados para un cliente y un servidor.

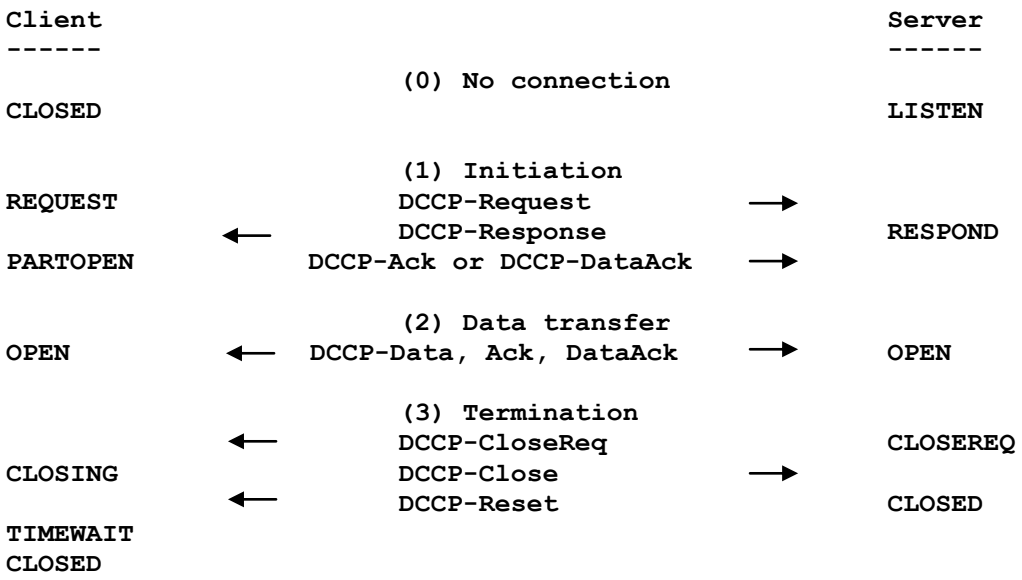

Gráfico 17: Estados del DCCP

Los nueve estados posibles se los indicamos a continuación.

CLOSED o Cierre de Conexión

Este estado representa conexiones no existentes dentro del protocolo.

LISTEN o en Escucha de conexión.

Este estado representa los sockets del servidor en el estado pasivo o ("*Passive-open*"), en escucha o ("*LISTEN*") y cerrado o ("*CLOSED*") no se asocian a ninguna conexión particular de *DCCP*.

REQUEST o Petición de conexión.

El socket del cliente incorpora este estado, de *CLOSED*, después de enviar un paquete *DCCP-Request* el paquete para intentar iniciar una conexión.

RESPOND o Responder

El socket del servidor incorpora este estado, desde *LISTEN*, después de recibir un paquete *DCCP-Request* de un cliente.

PARTOPEN

El socket del cliente incorpora este estado, desde *REQUEST*, después de recibir un paquete *DCCP-Request* del servidor. Este estado representa la tercera fase del *three-way handshake*. El cliente puede enviar datos de la aplicación en este estado, pero debe incluir un número del reconocimiento en todos sus paquetes.

OPEN o Abierto

Los sockets del cliente y del servidor incorporan este estado de apertura de un *punto final* o ("*PARTOPEN*") y respuesta o ("*RESPOND*"), respectivamente. Hablamos a veces de estados de servidor abierto o ("*SERVER-OPEN*") y de cliente abierto o ("*CLIENT-OPEN*"), correspondiendo al estado abierto o ("*OPEN*") del servidor y al estado *OPEN* del cliente.

CLOSEREQ o Petición de cierre

Un *socket*[20] del servidor incorpora este estado, desde *SERVER-OPEN*, para ordenar al cliente cerrar la conexión y llevar a cabo el estado de *TIMEWAIT*.

CLOSING o CIERRE de conexión

Los sockets del servidor y del cliente pueden incorporan este estado para cerrar la conexión.

[20] Socket, punto final de un enlace de comunicación de dos vías entre dos programas

TIMEWAIT o tiempo de espera.

El *socket* del servidor o del cliente permanece en este estado para **2** (*MSL*) o tiempo de la vida máximo de un segmento que equivale a **4** minutos; después de que la conexión se haya roto o finalizado, para prevenir los errores debido a la entrega de viejos paquetes. Solamente uno de las *puntos finales* tiene que incorporar el estado de *TIMEWAIT* el otro *punto final* puede incorporar el estado *CERRADO* inmediatamente, y el servidor puede solicitar al cliente llevar a cabo el estado de *TIMEWAIT* usando el tipo del paquete de *DCCP-CloseReq*.

4.3.4. Mecanismos del control de la congestión

El [RFC4340] describe que, el protocolo *DCCP* tiene control de congestión en todas sus conexiones, a diferencia del *TCP* las aplicaciones de *DCCP* proporcionan una selección de los mecanismos de control de congestión, de hecho las *dos mitades de conexión* se pueden controlar por diversos mecanismos de control de congestión. Los mecanismos de control de congestión son mostrados en los *CCIDs*.

Los *puntos finales* negocian su *CCIDs* durante la iniciación de la conexión. El *CCIDs 2 y 3* se definen actualmente; Los *CCIDs* desde el **0,** hasta el **1,** y del **4** al **255** son reservados, otros *CCID*s se pueden definir en el futuro.

El *CCID 2* proporciona control de la congestión (*TCP-like*) o congestion control for layered multicast data transfer, que es similar al de *TCP* [RFC793]. En el *CCID 2* el remitente mantiene una ventana de la congestión y envía los paquetes hasta que esa ventana este llena mientras que los paquetes (*ECN)* Notificación Explicita de

Congestión o Explicit Congestion Notification explicados en el [RFC3168] indiquen la congestión; la respuesta a la congestión parte en dos la ventana de congestión y a los reconocimientos. El *CCID 2* contiene los números de secuencia de todos los paquetes recibidos dentro de alguna ventana, similares a un reconocimiento selectivo (*TCP Sack*[21]) descrito en el [RFC2018].

El *CCID 3* en cambio proporciona el control "*TCP-Friendly*" Receptor de Transferencia o ttransferring Receptor (*TFRC*), una forma ecuación-basada de control de la congestión prevista para responder a la congestión más amigablemente que *CCID 2.*

El *CCID 3* se comportara de forma distinta al *TCP*. En un futuro próximo, se diseñara para funcionar con el *TCP*. Los comportamientos de *CCIDs 2 y 3* se definen completamente en los documentos separados en los [RFC4341, RFC4342].

4.4. Números de secuencia

El [RFC4340] describe que el "*número de secuencia*" normalmente es usado para impedir la repetición al enviar un paquete, o impedir el reprocesamiento múltiple de un paquete.

El protocolo *DCCP* utiliza *números de secuencia* para ordenar los paquetes en serie, detecta pérdidas y duplicados de la red y los protege contra accesos no autorizados.

Cada paquete lleva un *número de secuencia*; la mayoría de los tipos del paquete llevan un *número del reconocimiento* también.

[21] TCP SACK, soluciona problemas de congestión relacionados con la perdida de paquetes múltiples

Los *números de secuencia* se establecen a los paquetes *DCCP*. Es decir, los *números de secuencia* generados por cada *punto final* se aumentan en **1**, con un módulo **2^48**, por paquete, incluso los paquetes *DCCP-Ack* y *DCCP-Sync*, y otros paquetes no lleven datos de usuario, puesto que *DCCP* es un protocolo no fiable, no hay retransmisiones verdaderas, pero las retransmisiones eficaces tales como la de los paquetes *DCCP-Request*, también incrementan el *número de secuencia* .

Estas implementaciones *DCCP* nos permitirán detectar duplicados en la red retransmisiones, y perdida de reconocimientos.

4.4.1. Variables

El [RFC4340] menciona que, los puntos finales de *DCCP* mantienen un *número de secuencia* para cada conexión.

Las variables que se usan para los números de secuencia las mostramos a continuación en la tabla 4:

Variable	Descripción
ISS	Es el *número de secuencia* inicial enviado por el *punto final*. Esto iguala el *número de secuencia* del primer *DCCP-Request* o *DCCP-Response*.
ISR	Es el *número de secuencia* inicial recibido del otro *punto final*. Esto iguala el *número de secuencia* del primer *DCCP-Request* o *DCCP-Response* recibido.
GSS	Es el *número de secuencia* más grande enviado por el *punto final*
GSR	Es el *número de secuencia* más grande recibido del *otro punto final* en un reconocimiento de paquete.
GAR	Es el número más grande del reconocimiento recibido del *otro punto final*

	en un reconocimiento de paquete que no es un *DCCP-sync*
SWL y SWH	Es la (Ventana del *número de secuencia* baja y alta), para los *números de secuencia* de los paquetes recibidos.
LEZNA y AWH	Es la (Ventana del *número del reconocimiento* bajo y alto), para el reconocimiento, estos numeran los paquetes recibidos.

Tabla 3: Variables

4.4.2. Números de secuencia iniciales

Los *números de secuencia* iniciales de los *puntos finales* son fijados por los primeros paquetes *DCCP-Request* y *DCCP-Response* enviados.

Los *números de secuencia* iniciales presentan dos problemas:

- ✓ La entrega de los viejos paquetes, donde los paquetes viejos existentes en la red se entregan a una nueva conexión con las mismas direcciones y números de acceso.

- ✓ La secuencia de números de acceso, donde un acceso puede suponer los *números de secuencia* en una conexión futura.

4.4.3. Validez del número de secuencia

Los *puntos finales* de *DCCP* deben ignorar a los paquetes con los *números de secuencia* inválidos, que pueden presentarse si la red entrega reconocimientos de un paquete muy viejo. En el protocolo *DCCP*, los *números de serie* cambian con cada paquete enviado.

DCCP utiliza mecanismos de ventana para identificar las pérdidas, si el *número de secuencia* de un paquete dado es válido. Cada *HC-Remitente* da a *HC-Receptor* correspondiente a una anchura "*W"* de la ventana de la pérdida. Solamente el remitente puede anticipar este número. Una buena pauta es fijarlo a cerca de **3** o **4** veces el número máximo de paquetes que el remitente espera enviar en cualquier (*Round-Trip Time o RTT).* Los valores demasiado pequeños aumentan el riesgo de que los *puntos finales* salgan de la *Sync*. Después de la pérdida de datos; los valores demasiado grandes aumentan el riesgo de violación de la conexión.

El *HC-Receptor* instala una ventana de pérdida con los *números de secuencia* consecutivos de *W* que contienen un (*GSN*) o Red Global de Seismographic Global Seismographic Network,

El receptor puede agrupar la ventana de pérdida *GSN*, o arreglarlo asimétricamente. Los *números de secuencia* fuera de esta ventana son inválidos., a menos que ambas condiciones sean verdaderas.

1.- No se ha recibido ningún paquete válido recientemente.

2.-El paquete incluirá una identificación correcta, y un número válido de reconocimiento.

El *DCCP* de recepción debe pasar por alto los paquetes inválidos. Sin embargo, puede enviar un paquete *DCCP-Ack* al remitente. Este paquete debe contener el ultimo número de serie válido recibido. El otro *DCCP* enviará una opción de la identificación a la *RESYNC*.

4.4.4. Números del reconocimiento

El [RFC4340] menciona que, los reconocimientos acumulativos no tienen sentido en un protocolo no fiable. Por lo tanto, el campo del número del reconocimiento de *DCCP* tiene un parecido significativo con los *TCP*`s.

4.4.5. Validez y sincronización

Cualquier *punto final* de *DCCP* puede recibir los paquetes que no son realmente parte de la conexión actual. Si la red entrego un paquete viejo, un acceso no autorizado pude procurar invadir una conexión, o puede chocar con el *otro punto final*, y causar un *half-open*[22] en la conexión.

DCCP y *TCP*, utilizan una verificación del *número de serie* para detectar estos casos. Los paquetes y *números de secuencia* y/o reconocimiento que están fuera de rango se denominan ("secuencia inválida") y son procesados de modo anormal.

[22] Half-open, es una conexión que esta medio abierta

A comparación del *TCP*, el *DCCP* requiere de un mecanismo de sincronización y recuperación de grandes pérdidas. Un *punto final* pude enviar muchos paquetes durante una de la pérdida que cuando uno de sus paquetes finalmente llego, el *otro punto final* marca su *número de secuencia* como invalido.

4.4.6. Secuencia y número de la ventana de reconocimiento

Cada *punto final* de *DCCP* especifica las ventanas con una secuencia que tiende a bajar los espacios de la serie y del *número del reconocimiento*. Estas ventanas corresponden a los paquetes que el *punto final* espera recibir en los *tiempos ida-vuelta* próximos.

Las ventanas de la secuencia y el *número del reconocimiento* contienen siempre *GSR* y *GSS*, respectivamente. La longitud de las ventanas son controladas por las características de secuencia de la ventana para las dos *mitades de conexión*.

Los *números de secuencia* validos de la ventana para los paquetes de *DCCP B* son *SWL*, *SWH*, esta ventana contiene siempre *GSR*, y el número de serie más grande recibido en un paquete de secuencia válido de *DCCP B*, donde *W* es el valor de la secuencia en la ventana/B. Un cuarto de la ventana de la secuencia, es inferior o igual *GSR*, y los tres-cuartos son mayor que *GSR*. (Esta colocación asimétrica asume que las pérdidas son más comunes en la red.)

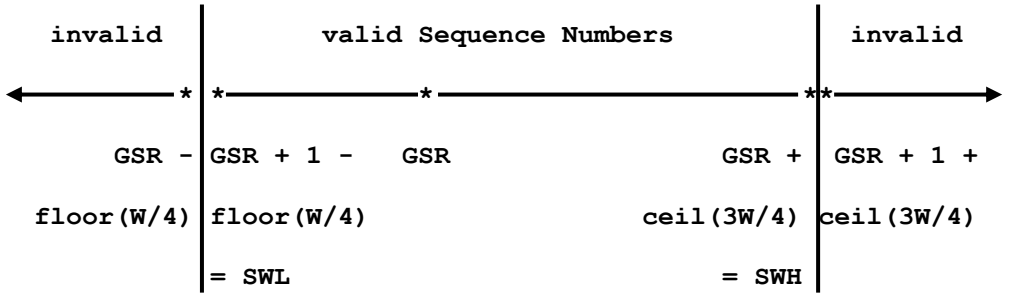

Gráfico 18: Secuencia y número de la ventana de reconocimiento

Los *números de secuencia* validos de la ventana para los paquetes de *DCCP B* son *LEZNA, AWH,* el extremo de la ventana, *AWH* es igual a *GS*S, que es el número de serie más grande enviado por *DCCP A*; la ventana *W,* es para paquetes más extensos, donde el valor *W,* tiene la secuencia de la ventana/A

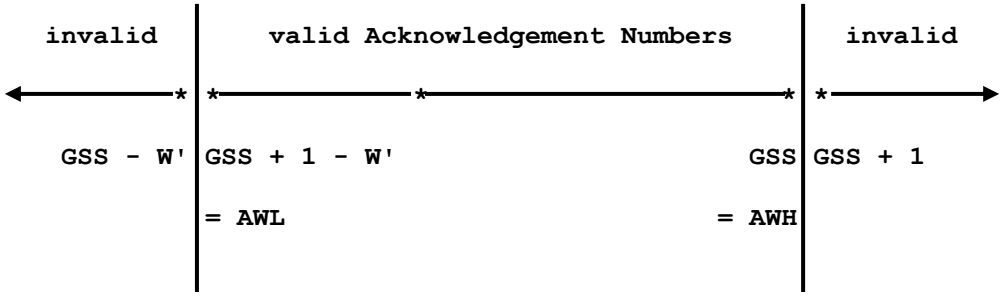

Gráfico 19: Secuencia y número de la ventana de reconocimiento1

SWL: = max (*GSR* + 1 – floor (*W*/4), *ISR*),

AW: = max (*GSS* + 1 - *W*', *ISS*).

Estos ajustes se deben aplicar solamente al principio de la conexión, (Las conexiones duraderas devuelven *números de secuencia* de modo que aparenten ser menores que *ISR* o *ISS*; los ajustes no se deben aplicar en estos casos.)

4.4.7. Números de secuencia cortos

El [RFC4340] menciona que, los números de serie de *DCCP* son de **48** bits de longitud. Esta secuencia de espacios largos protege las conexiones *DCCP* contra algunos accesos no autorizados ocultos, tales como la introducción de *DCCP-Reajustes* en la conexión. Sin embargo, los paquetes de *DCCP-Datos*, del *DCCP-Ack*, y de *DCCP-DataAck*, componen el cuerpo de cualquier conexión de *DCCP*, pueden reducir el espacio de la cabecera transmitiendo solamente **24** bits menos de los números principales de la

secuencia y del reconocimiento. Los *puntos finales* de recepción ampliarán estos números para usar **48** bits usando el siguiente pseudo código.

Procedimiento Extendido *números secuencia* (*S*, referencia)

S Es una secuencia de números de **24** bits desde el paquete del Servidor.

*REF***:** Es una secuencia de números de referencia relevantes de **48** bits.

*GSS***:** Si *S* es un *número de reconocimiento*.

GSR: si S es un *número de secuencia*.

Set REF_low := low 24 bits of REF

Set REF_hi := high 24 bits of REF

If REF_low (<) S /* circular comparacion mod 2^24 */

and S |<| REF_low, /* convencional, non-circular comparacion */

Return (((REF_hi + 1) mod 2^24) << 24) | S

Otherwise, if S (<) REF_low and REF_low |<| S,

Return (((REF_hi - 1) mod 2^24) << 24) | S

Otherwise,

Return (REF_hi << 24) | S

Las dos diferencias, si comparamos en si los estados detectados cuando el orden de secuencia baja de bits tienen espacios circulares. (La comparación circular ·*REF low* (<) *S*") retorna un valor verdadero si y únicamente si (*S – REF low) esta usando un calculo de dos complementos aritméticos estos están representados con un único número menor o igual a* ***2^23*** *(mod* ***2^24****).*) Cuando esto sucede el orden de bits son incrementados o decrementados como se crea necesario.

4.4.8. NDP Conteo y detección de pérdidas en aplicaciones

Las secuencias de números incrementados para cada uno de los paquetes *DCCP`s*, incluyendo (*NDP*) "*non_data packets"* o paquetes que no llevan datos de Aplicación, forman los *números de secuencia DCCP* para la detección de perdida en la red, pero no para detectar las pérdidas de datos en aplicaciones.

La *NDP* cuenta con una cantidad de reportes de cada uno de los paquetes *non-data*, que permiten al receptor *DCCP's* que determine cuando hay perdida de datos en la aplicación.

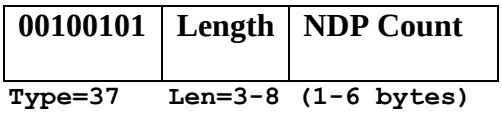

Gráfico 20: NDP Conteo y detección de pérdidas en aplicaciones

Los paquetes *non-data* constan de paquetes *DCCP*, *DCCP-Ack, DCCP-Close, DCCPCloseReq, DCCP-Reset, DCCP-Sync,* y *DCCP-SyncAck.* y otros tipos de paquetes como, *DCCP-Request, DCCP-Response, DCCP-Data, y DCCP-DataAck,* que también son considerados paquetes de datos, si bien no todos los paquetes *DCCP-Request y DCCP- Response* actualmente llevan datos de aplicación.

Los valores almacenados en *NDP* contienen igual el número de paquetes *non-data* consecutivos, estos paquetes no cuentan con la opción *NDP* y son considerados por tener *NDP* con cálculo de **0**.

La *NDP* cuenta con una opción que puede llevar uno o seis bytes de datos. Los formatos dan pequeñas opciones para almacenar la *NDP*, el receptor puede decir confiablemente que hubo una pérdida por lo menos de un paquete de datos.

4.5. Procesando Eventos

Aquí se describe como las conexiones *DCCP* se mueven entre estados, y que paquetes se deben enviar y cuando.

4.5.1. Establecimiento de la Conexión

La fase de iniciación de la conexión *DCCP* consiste en *three-way handshake*, inicializando un paquete *DCCP-Request* enviado por el cliente, a un *DCCP-Response* enviado por el servidor en respuesta, y finalmente un reconocimiento desde el cliente, usualmente por la vía de los paquetes *DCCP-Ack* o *DCCP-DataAck*.

El cliente se mueve desde el *REQUEST* hasta el estado de *PARTOPEN*, y finalmente a *OPEN*; el servidor se mueve desde *LISTEN* a *RESPOND*, y finalmente a *OPEN*.

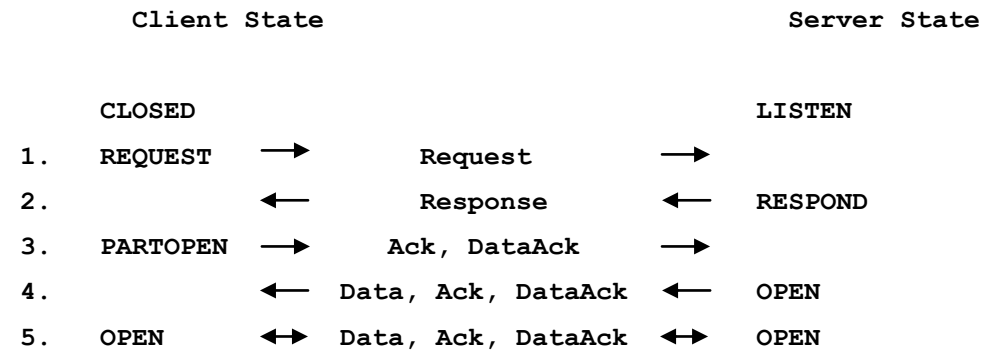

Gráfico 21: Establecimiento de la Conexión

4.5.1.1. Requerimientos del Cliente

El [RFC4340] describe, que cuando un cliente decide inicializar una conexión, debe lanzar un requerimiento de estado *REQUEST*, escogiendo un *número de secuencia* y enviando un paquete *DCCP-Request* para intentar comunicarse con el servidor.

Los paquetes *DCCP-Request* llevan características de negociación en común estos abren negociaciones con varios parámetros de conexión, llevando los datos de *cabecera*.

Un cliente en estado de *REQUEST* usa un tiempo exponencial ("*exponential-backoff")* timer al enviar un nuevo paquete *DCCP-Request* si no hay respuesta es recibido, la primera retransmisión sucede después de aproximadamente un segundo **1** segundo, garantizando que los paquetes no se queden fuera en **64** segundos; o se pueda usar cualquier estrategia de retransmisión que se ejecuta para retransmitir *TCP SYN`s*.

Cada nuevo *DCCP-Request* puede incrementar el *número de secuencia* por uno **1** y contienen un servicio de clave y datos de Aplicación estos originan el *DCCP-Request*.
Un cliente puede entregar estos *DCCP-Requests* después de **3** minutos.

4.5.1.2. Respuesta del Servidor

El [RFC4340] describe, que en la segunda fase de *three-way handshake*, el servidor se mueve desde el estado *LISTEN* al estado de *RESPOND* y envía un mensaje *DCCP-Response* para el cliente.

En esta fase, el servidor especificará a menudo las características que él quisiera utilizar, entre esas que el cliente solicite ser aceptado. El servidor responde con un paquete *DCCP-Request* y con un paquete *DCCP-Reset,* en este caso se considera la conexión.

Cuando el *DCCP-Request* solicita el puerto de destino y este no corresponde a un puerto de destino *DCCP* abierto para escuchar "*Código de servicio fallido*" el *DCCP-Request* solicita un código de servicio que no corresponde a los servicios de códigos registrados con el destino y el puerto. El servidor debe limitar el valor en el cual se generan estos reajustes; por ejemplo, no más a que **1024** bits por segundo.

El servidor no retransmite un paquete *DCCP-Response*; el cliente voluntariamente puede retransmitir el *DCCP-Request* si es necesario. Note que al retransmitir *DCCP-Request* tendrá un número diferente de secuencia desde el *DCCP-Request* original.
El servidor puede así distinguir verdaderas retransmisiones duplicadas en la red, y puede detectar esas retransmisiones *DCCP-Request* si la petición se dirige a conexiones existentes. Todos los *DCCP-Request* recibidos por el servidor están en estado de *RESPOND* y son extraídos de un nuevo *DCCP-Response*.

Para cada nueva *DCCP-Response,* el servidor incrementa la secuencia de números por uno **1** esto incluye datos de aplicaciones iguales al del original *DCCP-Response*.

El servidor no acepta más de una petición de *DCCP-Request* en una aplicación de datos para una conexión. En particular, las *DCCP-Response* enviadas hacia una retransmisión *DCCP-Request* con datos de aplicación contienen un valor para la pérdida de datos en las cuales los datos retransmitidos *DCCP-Request* contienen un valor de **0**.

La petición original *DCCP-Request* debe también informar las perdidas de datos en un bloque normal (*si el servidor acepta o no los datos*), si tienen un valor de **0** (*el servidor rechaza los datos*).

El servidor sale del estado de *RESPOND* para *OPEN* cuando recibe un *DCCP-Ack* valido del cliente, terminada la negociación de las *three-way handshake* deja el estado de *RESPOND* para *CLOSED* después de una espera de **4** *MSL* (**8** minutos).y este envía un paquete *DCCP-Reset* con valor de **2** "*Aborted*" para limpiar el estado en que se encuentra el cliente.

4.5.2. Completando la negociación

Cuando el cliente recibe un *DCCP-Response* del servidor se desplaza del estado de *REQUEST* al estado de *PARTOPEN* y termina la negociación de las *three-way handshake* enviando un paquete *DCCP-Ack* al servidor.
El cliente permanece en *PARTOPEN* hasta que este seguro que el servidor ha recibido el paquete que el cliente envió (*DCCP-Ack del ultimo paquete*).

Los clientes en estado *PARTOPEN* envían datos usando los paquetes *DCCP-DataAck* no los paquetes *DCCP-Data* esto se da por que los paquetes *DCCP-Data* carecen de números de reconocimiento así que el servidor no puede explicar un paquete *DCCP-Data* si el cliente tuviese un paquete *DCCP-Response*.

El enviar un *DCCP-Ack* cuando se intenta incorporar el estado de *PARTOPEN* podría causar la caída de la red. El cliente debe asegurarse de obtener estos paquetes con las características necesarias.

De preferencia con un mecanismo que contenga un contador de tiempo o "*timer*" de **200** milisegundos fijados en un paquete que se transmite en estado de *PARTOPEN* si se apaga este *timer* el cliente esta aun en *PARTOPEN* el cliente genera otro *DCCP-Ack* y retrocede el *timer*.

Si el cliente permanece en *PARTOPEN* por más que **4** *MSL* (***8*** minutos), debe reajustar la conexión con el código **2** del reajuste, "*Aborted*".

El cliente deja el estado de *PARTOPEN* para *OPEN* cuando recibe un paquete válido con excepción de *DCCP- Response*, de *DCCP-Reset*, o de *DCCP-Sync*. Del servidor.

4.5.3. Transferencia de datos

Se recalca que en la fase de transferencia de datos de la conexión, el servidor y el cliente están en el estado *OPEN*.

DCCP A envía los paquetes de los *DCCP-Data* y de *DCCP-DataAck* a *DCCP B* debido a los eventos en el host *A*. Estos paquetes son controlados por un mecanismo de congestion *CCID* para "*Uno-b"* (o la mitad de conexión).

En cambio si los paquetes *DCCP-Ack* enviados por *DCCP A* son controlados por el *CCID* para "*b-Uno"* o (la otra mitad de conexión). Generalmente, *DCCP A* llevará la

información del reconocimiento sobre los paquetes *DCCP-Data*, creando los paquetes de *DCCP-DataAck*.

Si se utilizan los paquetes *DCCP-Ack* cuando no hay datos a enviar de *DCCP A* al *DCCP B*, el estado de la congestión del *Uno-b* a *CCID* no permitirá el envió de datos. Los paquetes de *DCCP-Sync* y de *DCCP-SyncAck* pueden también aparecer en la fase de transferencia de datos.

Una distinción importante entre *DCCP- Sync* y otros tipos de paquetes, es que, *DCCP-Sync* saca un reconocimiento inmediato.

Si al recibir un paquete *DCCP-Sync* válido, en un *punto final DCCP* se debe generar y enviar inmediatamente una respuesta de *DCCP-SyncAck* (*conforme a los límites de la tarifa que se ponga en práctica*); el *número del reconocimiento* en ese *DCCP-SyncAck* debe igualar el *número de secuencia* de la *DCCP-Sync*. Una característica particular es que puede decidir iniciar una negociación unas ves que el estado *OPEN* fue alcanzado. Caso contrario puede ser que no permita la transferencia de datos, hasta que transcurra un cierto tiempo.

Los datos recibidos durante ese tiempo se deben rechazar usando datos de bloqueo para la perdida de paquetes con el valor **0**.

4.6. DCCP Diagrama de Estados

Lo discutido anterior mente se puede resumir en el diagrama de estado siguiente. El diagrama es ilustrativo.

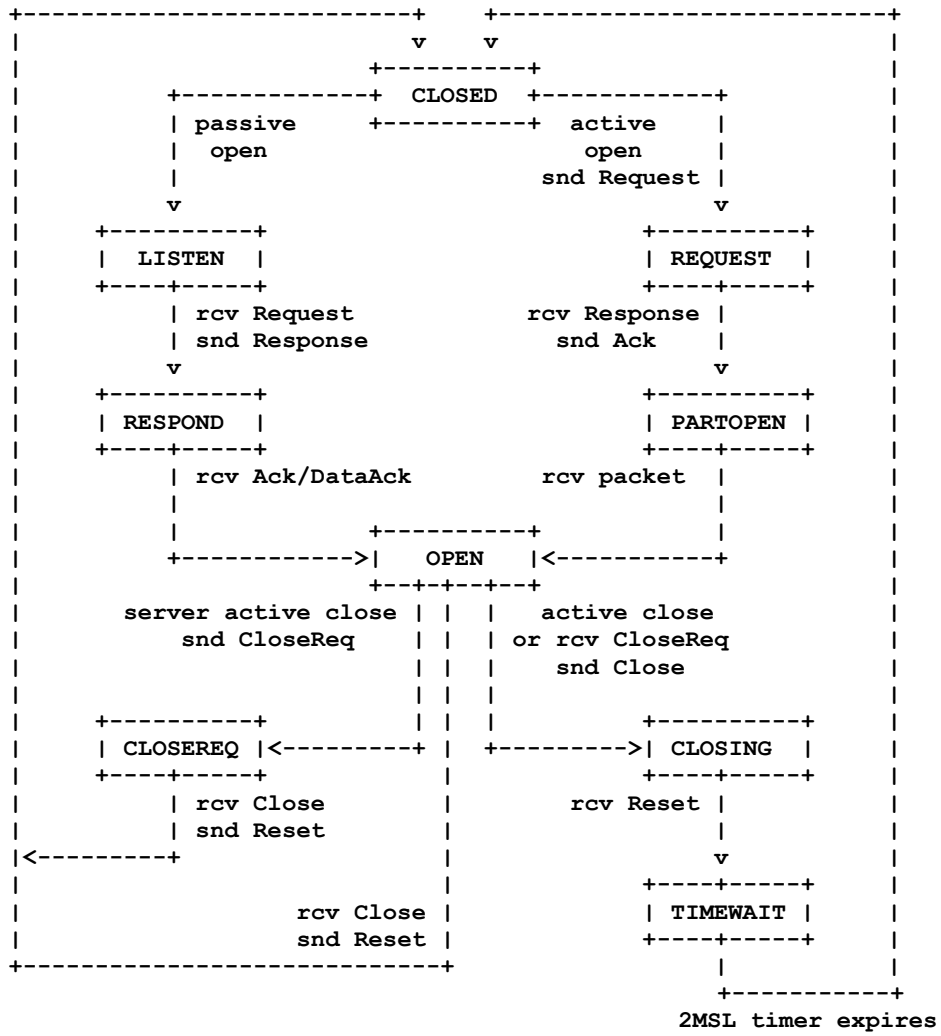

Gráfico 22: Diagrama de Estados DCCP

4.7. Pseudos-código

Esta sección se describe los pasos que un *punto final DCCP* debe pasar para recibir un paquete. El paquete recibido de define como *P*, el socket se define con la letra *S.*

Las variables que se utilizan son:

Variables	Descripción
S.SWL	número de la ventana serie baja
S.SWH	número de la ventana serie alto
S.AWL	número de la ventana de reconocimiento bajo
S.AWH	número de la ventana de reconocimiento alto

S.ISS	número de serie inicial enviado
S.ISR	número de serie inicial recibido
S.OSR	primer número de serie ABIERTO recibido
S.GSS	número de serie más grande enviado
S.GSR	número de serie válido más grande recibido
S.GAR	número válido más grande del reconocimiento recibido en una no-Sync

Tabla 4: Variables definidas para el Pseudos-código.

Paso 1:

Este paso comprueba si hay paquetes dañados.

Si la longitud del paquete es menor de **12** bytes, es dado de baja y debe volver.

Si P.*type* es desconocido, debe ser dado de baja y debe retornar

Si P.*Data* es más pequeño o mas grandes que los tipos de paquetes fijados por la cabecera *DCCP* el paquete es dado de baja y debe retornar

Si P.*type* no es un dato, *Ack*, o *DataAck* y p.x == 0 (*el paquete tiene números de serie cortos*), en este caso el paquete es dados de baja y debe retornar.

Si la suma de comprobación de la cabecera *DCCP* es incorrecta, el paquete es dado de baja y debe retornar

Paso 2:

En este paso se comprueba los puertos y los procesos del *TIMEWAIT*, los flujos ID son "*src addr, src port, dst addr, dst port* "

Si no existe un *sockets* y si el *S.state == TIMEWAIT*, los números de secuencia de reconocimiento se toman del paquete de entrada se reinicia la conexión, se da de baja el paquete y se lo devuelve.

Paso 3:

Aquí se describe los procesos en el caso de estar en el estado *LISTEN*.

Si el *S.state == LISTEN*, y si el *P.type == Request* o el *P* contiene una opción valida *init cookie*, aquí se visualiza las opciones de paquete. Algunos "*init cookies"* son procesados aquí, sin embargo, otras opciones son procesadas en el paso 8. Estas visualizaciones son realizadas solo si el punto final utiliza init cookies.

Generar un nuevo socket y un switch en el socket.

S:= nuevo *socket* para el par de puertos.

S.state = RESPOND, se elige S.*ISS* (*número de secuencia inicial*) luego inicializamos *S.GAR*:= S.*ISS*, colocamos. S.*ISR*, S.*GSR*, S.*SWL*, S.*SWH* desde el paquete y Continuamos con *S.state == RESPOND*

El paquete de respuesta será generado en el paso 11

Se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Paso 4:

Preparando la secuencia de números en estado *Request*

Si el *S.state == Request*, y Si (*P.type* == RESPONSE o P.type == *Reset*) y *S.AWL* <= *P.ackno* <= *S.AWH*, colocamos las variables del número de serie que corresponden al otro punto final, el P pasara al **paso 6***. Colocamos el *S.GSR*, *S.ISR*, *S.SWL*, *S.SWH*/* El proceso de la respuesta continuara en el **paso 10**; El proceso del reajuste continuara en el **paso 9**.

Solamente la respuesta y el reajuste son válidos en estado de *Request*, de otra manera se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Paso 5:

Preparando la Secuencia de números para Sync

Si el *P.type == Sync* o *P.type == SyncAck*, y si el *S.AWL <= P.ackno <= S.AWH* y *P.seqno >= S.SWL*, el *P* es válido. Después de esto, *P* pasará al **paso 6**. Un *SyncAck* se genera en caso de ser necesario en el **paso 15**. Actualizamos el *S.GSR, S.SWL, S.SWH*.

De otra manera se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Paso 6:

Comprobación de los números de secuencia.

Si *P.X* == 0, la característica de *Seqnos* corta es 0, el paquete tiene *Seqnos* cortas permitidas, caso contrario se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Si *P.X* == 0, se extender *P.seqno* y *P.ackno* para **48** bits.

Mantener *LSWL = S.SWL y LAWL = S.AWL*, si el P.type == *CloseReq* o P.type == *CLOSE* o *P.type ==Reset, LSWL:= S.GSR + 1, LAWL := S.GAR, si LSWL <= P.seqno <= S.SWH* y (*P.ackno no existe o LAWL <= P.ackno <= S.AWH*), actualizar el *S.GSR, S.SWL, S.SWH* si *P.type != Sync*, actualizamos el *S.GAR*, caso contrario .Si *P.type == Reset*, colocamos el paquete de reconocimiento *Sync* en el *S.GSR* enviando un paquete Sync de reconocimiento *P.seqno*, caso contrario se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Paso 7:

Comprobar si hay tipos inesperados de paquetes

Si (*S.is_server y P.type == CloseReq*) o (*S.is_server y P.type == Response*) o (*S.is_client y P.type == Request*) o (*S.state >= OPEN y P.type == Request y P.seqno >= S.OSR) o (S.state >= OPEN y P.type == Response P.seqno >= S.OSR) o (S.state == RESPOND y P.type == Data*), enviamos un paquete *Sync* de reconocimiento *P.seqno*. Caso contrario se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Paso 8:

Procesando las opciones y marcas de reconocimiento.

Las opciones de procesamiento no están específicamente descritas aquí.

Ciertas opciones, tales como mandataria, pueden causar que la conexión sea reinicializada, en este caso se pasa al **paso 9** y los siguientes no serán ejecutados.

Marcar el paquete de reconocimiento en *ECN*, (*en el vector ack, recibiendo un ECN*)

Paso 9:

Procesando el Reinicio

Si el *P.type == Reset*, Damos de baja la conexión, el *S.state:= TIMEWAIT*, colocamos un *TIMEWAIT* timer

Caso contrario se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Paso 10:

Procesando el estado *Request* de la segunda parte de la conexión.

Si el *S.state == Request*, obtenemos, P en Respuesta desde el servidor (*mirar el **paso 4***), y mover al estado *PARTOPEN*. Enviamos un Ack desde el estado PARTOPEN, enviamos paquetes de datos y retransmitimos Acks periódicamente, siempre se incluye algunos *init cookies* desde la respuesta.

El *S.state:= PARTOPEN*, envía un *PARTOPEN* timer enviamos un *S.state == PARTOPEN*

El **paso 12** enviará el *Ack* que termina el "*three-way handshake*"

Paso11:

Procesando el estado RESPOND

Si el *S.state == RESPOND*, y si *P.type == Request*, enviamos un Response, posiblemente conteniendo un *init cookie*, si un *init Cookie* es enviado, se desactiva el *S* y retorna.

El **paso 3** se creará otro *socket* cuando el cliente termina el "*three-way handshake*" caso contrario, el *S.OSR := P.seqno o el S.state := OPEN.*

Paso12:

Procesando el estado *PARTOPEN*

Si *S.state == PARTOPEN*, y si *P.type == RESPONSE*, enviamos un *Ack* caso contrario si *P.type!= Sync, el S.OSR:= P.seqno y el S.state := OPEN*.

Paso 13:

Procesando *CloseReq*

Si *P.type == CloseReq* y el *S.state < CloseReq*, aquí se Genera un Cierre de conexión el *S.state := CLOSING*, envía un *CLOSING* timer.

Paso 14:

Procesando Close

Si *el P.type == Close*, se Genera un Reinicio o (CLOSED). Dando de baja la conexión. Caso contrario se genera un reinicio de la conexión se da de baja el paquete y se lo devuelve.

Paso 15:

Procesando el Sync

Si P.type == Sync, Genera un SyncAck

Paso 16:

Procesando datos

En este punto cualquier dato de la aplicación sobre *P* pueden ser pasadas desde la aplicación, salvo que la aplicación no reciba datos de más de una *Request* o *Response*.

4.8. Control de la congestión

A cada mecanismo del control de la congestión apoyado por *DCCP* se le asigna un identificador de control de la congestión, o *CCID*, un número a partir de la (**0** a **255**). Durante el convenio de la conexión, y opcionalmente después de eso, *los puntos finales* negocian sus mecanismos del control de la congestión con los valores para la identificación del control de la congestión.

La identificación del control de la congestión tiene la característica número **1**. El valor de *CCID/A* iguala el *CCID* funcionando para *Uno-b* o la mitad-conexión. *DCCP B* envía una opción de cambio o "*R (CCID, K)*" para pedir *DCCP/A* y utilizar (*CCID, K)* para sus paquetes de datos.

El *CCID* es una característica de prioridad del servidor, así que las opciones de la negociación de *CCID* se pueden enumerar *CCID's* múltiples aceptables, clasificados en orden de prioridad descendente.

Actualmente se asigna los *CCIDs* de la siguiente manera.

CCID	Meaning	Reference
0-1	Reserved	
2	TCP-like Congestion Control	[RFC4341]
3	TCP-Friendly Rate Control	[RFC4342]
4-255	Reserved	

.

Tabla 5: Identificadores de control de congestión *DCCP*.

Las nuevas conexiones comienzan con *CCID 2* para los *puntos finales*. Si esto es inaceptable para un *punto final* de *DCCP*, ese *punto final* debe enviar opciones obligatorias del cambio de (*CCID*) en sus primeros paquetes. Todos los *CCID's* estandardizados para las implementaciones con *DCCP* corresponderán a los mecanismos del control de la congestión estandardizados previamente por el *IETF*.

4.8.1. Control de la congestión TCP-Like.

El [RFC 4341] describe que, el *TCP-like* como control *CCID* de la congestión este envía datos usando una variante cercana de los mecanismos del control de la congestión del *TCP*, Este incorpora una variante de los reconocimientos selectivos (*TCP Sack*) descritos en los [RFC2018, RFC3517]. El *CCID 2* es eficaz para los remitentes que pueden adaptarse a los cambios precipitados adentro ventana de la congestión típica del control Multiplicativo de la congestión y particularmente útil, para los remitentes que quisieran aprovechar el ancho de banda disponible en un ambiente con condiciones cambiantes.

4.8.2. Control de la congestión TFRC

TFRC es apropiado para los flujos que prefieran reducir al mínimo los cambios precipitados en la cantidad de datos que se envía, incluyendo las aplicaciones que usan buffers pequeños. *TCP-like*, parte en dos la cantidad de datos que envía en respuesta a cada acontecimiento de la congestión. Según lo explicado dentro del [RFC3448].

TFRC en *CCID 3* en cambio responde más lento a los cambios en ancho de banda disponible, los mecanismos *TCP-like* y, *CCID 3* se deben utilizar solamente para las

aplicaciones con rendimiento de procesamiento simple. Para las aplicaciones que necesitan simplemente transferir tantos datos como sea posible, se recomienda usar *TCP-like* control de la congestión, *CCID 2*.

Capítulo 5

Implementación del prototipo DCCP

5.1. Introducción

El Protocolo de Control de Congestión de Datagrama *DCCP*, es un nuevo protocolo de transporte siendo definido en el *IETF* que proporciona un flujo controlado por congestión de datagramas no fiables. En aplicaciones sensibles a tardanza, donde la transmisión óptima de medios de comunicación es el objetivo principal. En tiempo real la transmisión de medios de comunicación implica la generación de un flujo de paquete, donde cada paquete generado debería ser entregado cuanto antes o no entregado en absoluto. Las exigencias de tiempo son más importantes que la fiabilidad

Pero la carencia de mecanismos de control de congestión en *UDP* hace este protocolo una opción aventurada. En este momento la única alternativa es usar el protocolo de transporte de *TCP*, que proporciona mecanismos de control de congestión, pero el mecanismo de nueva transmisión es una desventaja debido a la alta tardanza de punta a punta que esto causa. *DCCP* combina lo mejor de los dos protocolos dentro del contexto de transmisión de medios de comunicación, apoyando mecanismos de control de congestión sin nuevas transmisiones.

Por esta y otras razones nos hemos visto en la necesidad de implementar un prototipo del protocolo de transporte *DCCP* la parte de conexión y transferencia de datos, el cual funcione de tal manera que en el se pueda experimentar con diferentes mecanismos de control de congestión.

5.2. Descripción del prototipo DCCP

En esta sección describimos el prototipo del protocolo de transporte *DCCP* el mismo que lo hemos dividido en dos partes:

La primera, es la parte dinámica del protocolo refiriéndonos con esto al proceso de manejo datos dentro de la aplicación.

El funcionamiento del prototipo *DCCP* esta materializado por la actividad de tres *threads* definidos por el *DCCP*, activados al comienzo del funcionamiento del protocolo y un número arbitrario de *threads* de ejecución definidos por la aplicación.

Para lo cual hemos incluido diferentes "*threads*" de ejecución, que serán descritos posteriormente.

La segunda, es la parte de funcionalidad de este prototipo que se lo ha dividido en módulos, que muestran diferentes aspectos a considerar referentes al comportamiento del protocolo *DCCP*. Esta división nos va a permitir experimentar con varios algoritmos de control de congestión, y desarrollar variantes de los mismos, para de esta manera evaluarlos posteriormente.

En la implementación, cada módulo se puede probar manualmente, introduciendo datos de prueba, independientemente de los demás módulos.

En este trabajo investigativo, se pone mas énfasis en el la parte de funcionalidad del protocolo *DCCP* descrito en el [RFC 4340], que nos sirvió como base para realizar el prototipo *DCCP*, cuyo objetivo es mejorar el desempeño de los protocolos de transporte como *UDP* y *TCP* , principalmente en lo que se refiere al control de congestión, control de flujo y transmisión de datos en tiempo real, por cuanto nos hemos visto en la necesidad de adaptar al *DCCP* descrito en el [RFC 4340] a cambios en la tecnología de transmisión.

5.3. Arquitectura del prototipo DCCP

Para darle una fácil funcionalidad al prototipo *DCCP* básico, hemos tratado de conservar la simplicidad lo cual nos llevo a definir los principales elementos y componentes del prototipo de protocolo de tal forma que se pueda descomponer de forma separada.

En el siguiente Gráfico de muestra la arquitectura del prototipo DCCP.

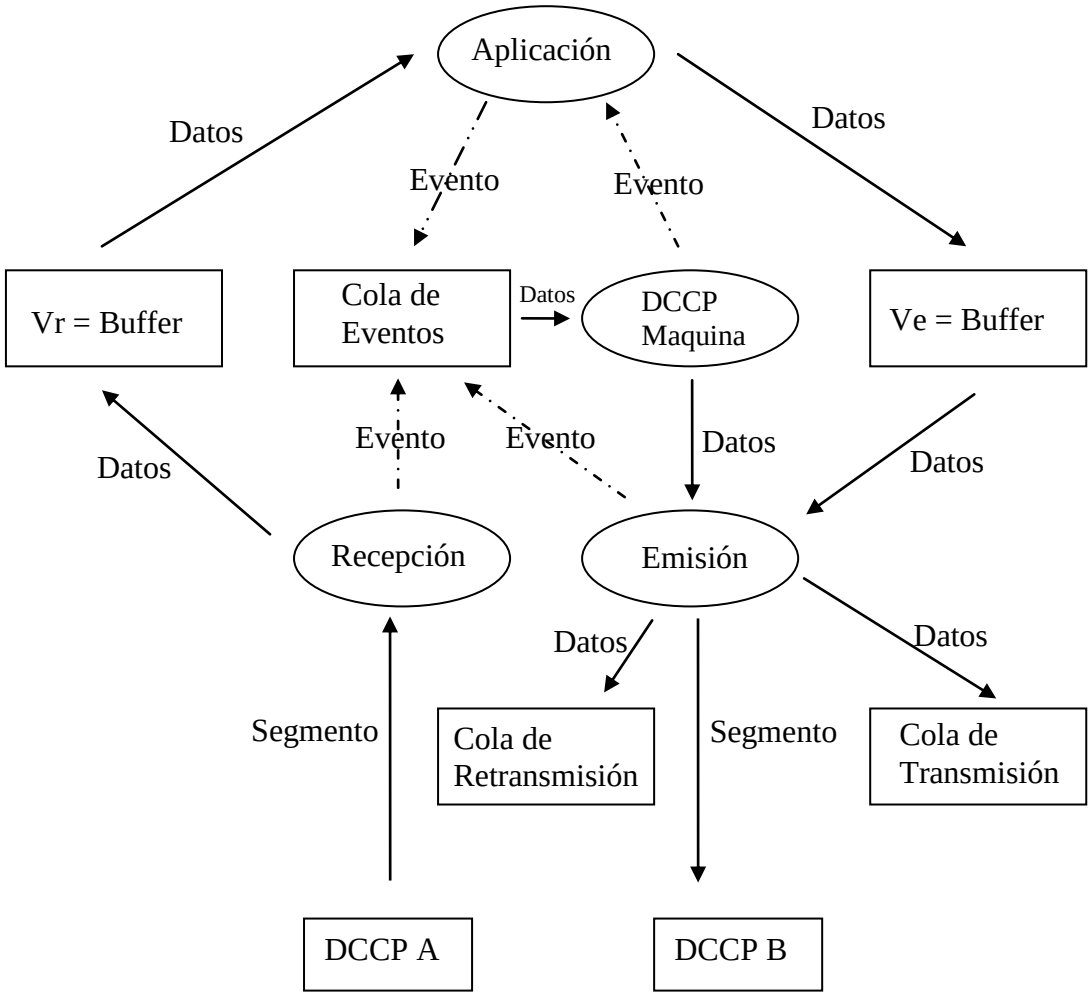

Gráfico 23: Arquitectura del prototipo DCCP

Los elementos más importantes que componen esta implementación los describiremos a continuación:

5.4. Ventana de emisión y el buffer de emisión

Para esta implementación la ventana de emisión involucra dos elementos relacionados con el envío de datos, el primero, un buffer de tamaño máximo determinado, destinado a recibir los bytes que la aplicación solicita enviar, segundo, una ventana de emisión que es un conjunto de bytes contiguos que se almacenan dentro del buffer y que han sido

enviados por *DCCP*, pero de los cuales aún no se ha recibido confirmación de recepción.

En nuestro caso el tamaño del buffer de emisión está dado por la cantidad de memoria que *DCCP* asigna a la conexión. Esto puede ser configurado por las aplicaciones. El tamaño de la ventana de emisión, estará dado, en cada momento, por el método de control de congestión utilizado.

Mantiene variables relacionadas con el envío de datos, tal como se define en el [RFC 4340], (*ENV_GSR***)** *número de secuencia* más grande recibido del *otro punto final*, (*ENV_ISS*) *número de secuencia* inicial enviado por el *punto final*, esto iguala el *número de secuencia* del primer *DCCP-Request* o *DCCP-Response*. (*ENV_ISR*) *número de secuencia* inicial recibido *del otro punto final*. Esto iguala el *número de secuencia* del primer *DCCP-Request* o *DCCP-Response* recibido. (*ENV_GSS) número de secuencia* más grande enviado por el *punto final*. (*ENV_GAR***),** Es el número más grande del reconocimiento recibido del *otro punto final* en un reconocimiento de paquete que no es un *DCCP-sync*. (*ENV_SWH*) Ventana del número de secuencia baja y alta.

5.5. Ventana de recepción y buffer de recepción

Como en el caso anterior, la ventana de recepción involucra dos elementos relacionados con el envío de datos de la misma manera como en el caso anterior, el buffer de recepción y la ventana de recepción. El primero es el espacio asignado a la aplicación por *DCCP*, donde este almacena los datos recibidos hasta que la aplicación los lea,

momento en el cual será liberado ese espacio del buffer. El tamaño de este buffer, que coincide con el tamaño máximo de la ventana de recepción, se configura como parámetro. La ventana de recepción puede considerarse como desplazándose sobre el buffer, sobre los lugares no ocupados, e indica los *números de secuencia* que pueden ser aceptados. El tamaño de la ventana puede llegar a ser igual al valor máximo del buffer, en el caso de que la aplicación no lea al ritmo de llenado del buffer, su tamaño puede disminuir a cero. Se mantienen aquí las variables definidas en [RFC 4340] relacionadas con la recepción.

5.6. Colas

Una "*cola*" es una estructura lineal, en la cual los elementos sólo pueden ser adicionados por uno de sus extremos y eliminados o consultados por el otro. Este tipo de estructuras lineales se conocen como estructuras (*FIFO),* o sea, el primero en llegar es el primero es salir o (*First-In-First-Out*). También hay que tener presente, que el único elemento visible en una *cola* es el primero y mientras éste no haya salido (*eliminado*), no es posible tener acceso al siguiente.

5.6.1. Colas de transmisión y retransmisión

En esta implementación los paquetes enviados por el *DCCP* remoto por primera vez o como consecuencia de una retransmisión son colocados en las *colas*, la *cola* de transmisión es alimentada por la Máquina de Estados *DCCP* y por el *thread* de envío, cuando extrae bytes de la ventana de emisión para generar los paquetes de datos

correspondientes. Este *thread* es el que se encarga de transmitir los paquetes de la *cola*, en orden. La *cola* de retransmisión recibe paquetes por parte del *thread* de envío, quien los coloca en ella cuando son transmitidos por primera vez, siempre que dicha transmisión este sujeta a reintentos. Una vez en la *cola* de retransmisión, las retransmisiones son disparadas por timers asociados a los paquetes. Esta *cola* es modificada cuando se recibe un *Ack*, eliminando los paquetes confirmados.

5.6.2. Cola de eventos

En esta *cola* se encuentran los objetos que representan los eventos que debe procesar la Máquina de Estados *DCCP*. Cada evento está compuesto por un código que identifica su tipo, y un objeto que contiene la información correspondiente, los eventos son colocados en la *cola* por los diferentes *threads* de la aplicación, el propio *thread* de la Máquina de Estados *DCCP* y los timers también pueden generar eventos, el orden de atención de los mismos por parte de la Máquina de Estados *DCCP* es *FIFO*.

5.7. Maquina de estados DCCP

Esta maquina es una entidad activa, implementada mediante un thread, que representa la funcionalidad básica de *DCCP*, a través de 9 estados. Esta funcionalidad permite el establecimiento de conexión y la terminación ordenada de la misma y posee un estado *OPEN*, correspondiente al estado conectado del protocolo, en el cual se lleva a cabo la transferencia de los datos. La máquina de estados *DCCP* funciona extrayendo eventos de una cola destinada a ese efecto y procesándolos. Los eventos son los especificados en [RFC 4340].

En esta implementación se manejan los siguientes estados:

Número	Estado
1	CLOSED
2	LISTEN
3	REQUEST
4	RESPOND
5	OPEN
6	PARTOPEN
7	CLOSING
8	TIMEWAIT
9	CLOSEREQ

Tabla 6: Estados DCCP

La maquina de estados *DCCP* maneja los siguientes mensajes:

Passive-open, (“conexión en estado abierto pasivo”)

Active-open, (“conexión en estado abierto activo”)

Receive-Ack, (“recibir un reconocimiento”)

Receive-reset, (“recibir un reinicio”)

Server-active-close, (“servidor activo cerrado”)

Active-close, (“cerrado activo”)

Receive-packet, (“recibir paquete”)

Receive-response, (“recibir respuesta”)

Receive-request, (“recibir solicitud”)

Timer-expires, (“tiempo expirado”)

Receive-close, (“recibir cierre de conexión”)

Número	Estado	Mensaje	Estado
1	CLOSED	Passive-open →	LISTEN
2	LISTEN	Receive-request →	RESPOND
3	RESPOND	Receive-ack →	OPEN
4	CLOSING	Receive-reset →	TIMEWAIT
5	OPEN	Server-active-close →	CLOSEREQ
6	CLOSEREQ	Receive-close →	CLOSED
7	OPEN	Active-close →	CLOSING
8	REQUEST	Receive-response →	PARTOPEN
9	PARTOPEN	Receive-packet →	OPEN
10	CLOSED	Active-open →	REQUEST
11	CLOSED	Timer-expires →	CLOSED
12	OPEN	Receive-close →	CLOSED

Tabla 7: Estados de la maquina DCCP

5.8. Timers en el prototipo DCCP

El mecanismo de manejo de timers se realizó de manera simple, utilizando las facilidades provistas por Java.

Se implementan timers para controlar los tiempos máximos de espera de una reacción del *otro punto final*, para el mecanismo de envío de datos (*piggybacking y retransmisión*), para determinar si el *DCCP* remoto aun está funcionando, y para la finalización de la conexión. Adicionalmente, la implementación define otro para que la aplicación pueda especificar en ciertos casos el tiempo máximo a esperar por una respuesta del *punto final*. Para una explicación detallada de las funciones de los timers, referirse a [7].

5.9. Procesos ligeros o Threads

Un "*thread*", también llamado "*proceso ligero*", es una unidad básica de utilización del *CPU*, y consiste en un contador de programa, un juego de registros y un espacio en la pila. Un thread comparte, con sus iguales, la sección de código, sección de datos y recursos del sistema operativo como archivos abiertos y señales, lo que se denomina colectivamente como una tarea. Podemos decir que un proceso es como una tarea con un solo thread. Debido a que son "*procesos ligeros*", la conmutación del *CPU* entre *threads* y su creación es poco costosa en comparación con el cambio de contexto entre procesos.

En esta implementación los *threads* trabajan de manera independiente, comunicándose a través de eventos dirigidos a la Máquina de Estados DCCP, ésta utiliza señales de control para indicar a alguno de los *thread* de la aplicación realice una función determinada.

5.9.1. Thread de recepción

En nuestro prototipo de protocolo se utilizo este *thread* que es el mas simple, que es el encargado de inicializar el socket *UDP* y que constituye el soporte de comunicación, durante el resto de la vida del proceso, este lee los bloques del *UDP* que llegan provenientes del lado remoto, entregándolos a la Máquina de Estados *DCCP*, en forma de eventos.

5.9.2. Thread de Emisión

Este *thread* es el encargado de realizar diversas tareas relacionadas con el envío de los Paquetes *DCCP*. Se encarga de la inicialización y posterior mantenimiento de las *colas* de transmisión y retransmisión. Para eso se definen dos métodos, uno que permite agregar paquetes a la *cola* de transmisión, y otro que, al recibirse un *Ack*, permitiendo eliminar de la *cola* de retransmisión los paquetes totalmente confirmados.

5.9.3. Thread de la Máquina de Estados DCCP

Este *thread* representa la Máquina de Estados *DCCP*, tal como se define en el [RFC 4340]. Está implementada, como un doble switch, a través de la cual se procesan los eventos en función del estado de la Máquina de Estados *DCCP*. Los estados definidos son los **12** especificados en el [RFC 4340], al igual que los eventos. Los primeros abarcan la fase de establecimiento de conexión, intercambio de información y finalización de la misma, mientras que los eventos pueden provenir de acciones solicitadas por la aplicación, de vencimiento de timers o de paquetes provenientes del *DCCP* remoto.

El código se trató de mantener simple, a través del uso de procedimientos comunes a diferentes combinaciones estado/evento. Debe destacarse que se sigue estrictamente la especificación del [RFC 4340], ya que esto facilita la comprensión del código y la facilidad para el agregado de módulos.

5.9.4. Threads de usuario

Debido a que las operaciones invocadas de *DCCP* son "*bloqueantes*", para respetar la filosofía de java, es posible que el usuario cree uno o más, para invocar a *DCCP*. Los detalles de la operación, se presentan en la sección de interfaz.

5.10. Interfaz DCCP

En nuestro trabajo de investigación una de las decisiones tomadas inicialmente, junto con la elección de Java como lenguaje, fue la de hacerlo compatible para desarrollar aplicaciones escritas en Java. Esto por supuesto no debe entenderse como que una aplicación Java escrita para utilizar *DCCP*.

El objetivo buscado en este aspecto, es implementar una interfaz con las mismas funciones que provee Java **2 v 1.4.2**. Este aspecto aún no ha sido completado, implementándose sólo la funcionalidad básica que permite establecer y terminar una conexión, e intercambiar datos.

Las invocaciones a *DCCP* son *bloqueantes,* produciendo un retorno exitoso con la entrega de la eventual información solicitada. Los posibles errores que puedan producirse (*dirección errónea al establecer conexión, datos nulos, tiempos vencidos, etc.*) son comunicados a la aplicación a través del mecanismo de excepciones provisto por Java. En este caso en particular, se define una nueva excepción, que contempla las situaciones anormales especificadas en el [RFC 4340] y otras propias de la implementación

Capítulo 6

Tecnologías usadas

Después de un proceso de investigación y pruebas de las diferentes soluciones propuestas, se seleccionaron las tecnologías para la implementación final del prototipo de protocolo DCCP descrito anteriormente.

6.1. Java

Java fue el lenguaje seleccionado para el desarrollo de esta aplicación, ya que al ser un lenguaje de programación orientado a objetos y que esta enfocado a la seguridad, *multi-plataforma*[23] e internacional, nos proporciono el medio necesario para cumplir con los objetivos planteados en este trabajo investigativo.

Java es el primer lenguaje de programación diseñado para el networking y para el desarrollo de aplicaciones Web. Cuando Sun desarrolló este lenguaje, en 1995, las posibilidades de las redes e incluso de Internet no eran las mismas que ahora y la enorme facilidad que ofrece Java para escribir páginas y aplicaciones Web hacen de él el lenguaje de Internet por excelencia. Por otro lado, Java tiene un gran futuro en lo que se refiere al desarrollo de software pues al ser un lenguaje independiente de la plataforma, no hace falta crear versiones distintas de un mismo programa para cada tipo

[23] multi-plataforma, esta orientado a trabajar sobre diferentes entornos o sistemas operativos.

de ordenador y esto permite reducir enormemente los costes de desarrollo de software. Sus características de universalidad y portabilidad hacen de Java el lenguaje idóneo para el desarrollo de aplicaciones para sistemas conectados en red. Además, los sistemas y las conexiones en red que se construyen actualmente son cada vez más sofisticados y Java es una herramienta para la gestión de la complejidad.

Java ha triunfado como lenguaje para el desarrollo de aplicaciones en el mercado. Una de sus principales características es que es universal, es decir, independiente de la plataforma, por lo que un programa hecho en Java se ejecutará igual en un ordenador con Windows que en un ordenador con Unix.

Esta portabilidad se consigue haciendo de Java un lenguaje interpretado, compilando el código fuente a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y el sistema operativo, finalmente, interpretando este lenguaje intermedio utilizando la máquina virtual de java.

6.2. Eclipse

Eclipse es un proyecto de desarrollo de software de código fuente abierto (*open source*) cuyo objetivo es la construcción de herramientas integradas para el desarrollo de aplicaciones.

Eclipse es una plataforma de software de código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "*aplicaciones de Cliente Enriquecido*", opuesto a las aplicaciones "*Cliente-liviano*" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos integrados de desarrollo

o integrated development environment (*IDE*), como el *IDE* de Java , llamado conjunto de herramientas para desarrollo en java o *Java Development Toolkit* (*JDT*) y el compilador de java interno de Eclipse o Eclipse Compiler for Java (*ECJ*) que se embarca como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas.

Eclipse fue desarrollado originalmente por la máquina de negocios internacionales o International Business Machines (*IBM)* como el sucesor de su familia de herramientas para "V*isualAge*". Eclipse es ahora desarrollado por la fundación eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Capítulo 7

Conclusiones y recomendaciones

El objetivo de este trabajo de investigación fue cumplido, logrando desarrollar un prototipo con la funcionalidad básica del protocolo DCCP descritas en el [RFC4340].

Nuestro desarrollo contempla las etapas de conexión, desconexión y transferencia de datos descritas en el [RFC4340], cabe aclarar que en lo que respecta a transferencia de datos, se implemento solo el esqueleto que permite una transferencia básica, habiendo diseñado un prototipo en el cual se pueda incorporar fácilmente y de manera modular los mecanismos de control de congestión del protocolo *DCCP* definidos en los [RFC4341] y [RFC4342].

El lenguaje Java, seleccionado al comienzo fue un pilar fundamental gracias a la portabilidad que brinda, demostró ser versátil, adaptándose a las características del prototipo de protocolo a implementar. El manejo de timers se lo realizamos utilizando las facilidades provistas por Java el cual nos permitió implementarlos fácilmente, los cuales controlan los tiempos máximos de espera, el envió de datos y la finalización de la conexión.

La capacidad del manejo de *threads* nos permitió trabajar de manera independiente comunicándose a través de eventos dirigidos a la Maquina de Estados la cual utiliza

señales de control para indicar a alguno de los *thread* de la aplicación realice una función determinada.

Entre las limitaciones del desarrollo, podemos mencionar un aspecto de performance, derivado del lenguaje utilizado, que sin embargo no se monto en las pruebas.
La característica de funcionar sobre UDP debe notarse que al no proveer java librerías Standard de manejo de sockets para el acceso directo a IP, cuando se desee comparar nuestra implementación con otros implementaciones que funcionan a nivel kernel, se deberá recurrir a librerías como la Jpcap, y posiblemente se encuentre alguna limitación en función del sistema operativo que se utilice.

Esta investigación constituye la base para una futura investigación sobre control de congestión para Apps (del tipo a las descritas en los [RFC4341] y [RFC4342]). Entre las posibles extensiones en el corto plazo, cabe mencionar:

- ✓ Incorporación de mecanismos de control de congestión
- ✓ Chequeo exhaustivo de la implementación
- ✓ Creación de conjuntos de datos de prueba y evaluación de los diferentes medioambiente de operación del protocolo DCCP descritas en el [RFC4340] y los requerimientos de las aplicaciones que lo utilizan.
- ✓ Completar la interfaz ofrecida a las aplicaciones, de manera de que las Apps que corran sobre el DCCP, puedan hacerlo sobre nuestro prototipo con mínimas modificaciones.
- ✓ Extensión del prototipo para que pueda funcionar directamente sobre IP, y de esta manera interactuar con las implementaciones del DCCP genuinas.

- ✓ En caso de que se haga necesario, mejorar la performance del código desarrollado.

Lista de Acrónimos

API	Application Programming Interface (Interfaz de Programación de Aplicaciones).
CCID	Congestion Control Identifier (Identificador de la Control de la Congestion).
CPU	Central Procesor Unit o (Unidad Central de Proceso).
DCCP	Datagram Congestion Control Protocol (Protocolo de Control de Congestion de Datagramas).
DNS	Domain Name Service (Nombre del Dominio del Servicio).
ECN	Explicit Congestion Notification (Notificación Explicita de Congestión).
FTP	File Transfer Protocol o (Protocolo de Transferencia de Ficheros).
GSN	Global Seismographic Network (Red Global de Seismographic).
ICMP	Internet Control Message Protocol o (Protocolo de Control de Mensajes de Internet).

IDE Integrated development environment o (entornos integrados de desarrollo).

IEEE The Institute of Electrical and Electronics Engineers o (Instituto de Ingenieros Eléctricos y Electrónicos).

IETF Internet Engineering Task Force **(**Grupo de Trabajo en Ingeniería de Internet).

ISO International Standards Organization o (Organización Internacional para la Estandarización).

JDT *Java Development Toolkit* o (conjunto de herramientas para desarrollo en java)

MSL Tiempo de la vida máximo de un segmento.

POP Post Office Protocol o (Protocolo Simple de Transferencia de Correo).

RFC Request For Comments (Petición de Comentarios).

RTT Round Trip Time o (tiempo de ida y vuelta).

SMTP Simple Mail Transfer Protocol (Protocolo Simple de Transferencia de Correo Electrónico).

TCP Transport Control Protocolo (Protocolo de Control de Transporte).

TFRC Transferrin Receptor (Receptor de Transferencia).

UDP User Datagram Protocol (Protocolo de Datagramas del Usuario).

Referencias y Bibliografía

[RFC2581] ALLMAN, M., PAXSON, V., &W. STEVENS, "*TCP Congestion Control*", RFC 2581, April 1999.

[RFC1812] BAKER, F., "*Requirements for IP Version 4 Routers*", RFC 1812, June 1995.

[RFC3711] BAUGHER, M., MCGREW, D., NASLUND, M., CARRARA, E., & K. NORRMAN, "*The Secure Real-time Transport Protocol (SRTP)"*, RFC 3711, March 2004.

[RFC1948] BELLOVIN, S., "*Defending Against Sequence Number Attacks*", RFC 1948, May 1996.

[RFC2460] DEERING, S. and R. HINDEN, "*Internet Protocol, Version6 (IPv6) Specification"*, RFC 2460, December 1998.

[RFC1982] ELZ, R. & R. BUSH, "*Serial Number Arithmetic*", RFC 1982, August 1996.

[RFC4340] FLOYD, S. & E. KOHLER, "*Datagram Congestion Control Protocol (DCCP)"* RFC4340, March 2006.

[RFC4341] FLOYD, S. & E. KOHLER, "*Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control*", RFC 4341, March 2006.

[RFC4342] FLOYD, S., KOHLER, E., & J. PADHYE, "*Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)*", RFC 4342, March 2006.

[4] FRANKLIN, St. "*Datagram Congestion Control Protocol (Dccp)*". Descargado el 03-02-2007, Disponible en web: http://babylon.com/definition/DCCP/Spanish.

[1] GOMEZ, A. "*Protocolo TCP/IP*". Descargado el 12-12-2006, disponible en web: http://beto_redes.galeon.com/defintcpip.html.

[RFC3448] HANDLEY, M., FLOYD, S., PADHYE, J., & J. WIDMER, "*TCP Friendly Rate Control (TFRC): Protocol Specification*", RFC 3448, January 2003.

[RFC3775] JOHNSON, D., PERKINS, C., & J. ARKKO, "*Mobility Support in IPv6*", RFC 3775, June 2004.

[RFC2401] KENT, S. & R. ATKINSON, "Security *Architecture for the Internet Protocol*", RFC 2401, November 1998.

[LA02] LARA, J. "*Protocolos de red*". Descargado el 30-03-2007. Disponible en web: http://www.monografias.com/trabajos12/redes/redes.shtml.

[RFC3828] LARZON, L-A., DEGERMARK, M., Pink, S., JONSSON, L-E.,& G. FAIRHURST, "*The Lightweight User Datagram Protocol (UDP-Lite)*", RFC 3828, July 2004.

[PMTUD] MATHIS, M. & J. HEFFNER, "*Path MTU Discovery*", Work in Progress, March 2006.

[3] NAHUAL, R. 1999. "*User Datagram Protocol (UDP)*". Descargado el 03-01-2007, disponible en web: http://www.raza-mexicana.org/textos/revista/txt/raza007.txt.

[RFC3692] NARTEN, T., "Assigning *Experimental and Testing Numbers Considered Useful*", BCP 82, RFC 3692, January 2004.

[RFC2434] NARTEN, T. & H. ALVESTRAND, "*Guidelines for Writingan IANA Considerations Section in RFCs*", BCP 26, RFC 2434, October 1998.

[RFC792] POSTEL, J., "*Internet Control Message Protocol"*, STD5, RFC 792, September 1981.

[RFC793] POSTEL, J., "*Transmission Control Protocol*", STD 7, RFC 793, September 1981.

[5] POSTEL, J. "*Internet Protocol*". RFC791. September 1981

[6a] PUIGDEMUNT, E & HARLAM, J. "*Modelo de Referencia OSI"*. Descargado el 02-01-2007 Disponible en web: http://pchardware.org/redes/redes_osi.php.

[RFC3168] RAMAKRISHNAN, K., FLOYD, S., & D. BLACK, "*The Addition of Explicit Congestion Notification (ECN) to IP*", RFC 3168, September 2001.

[RFC3550] SCHULZRINNE, H., CASNER, S., FREDERICK, R., & V. JACOBSON, "RTP: A *Transport Protocol for Real-Time Applications*", STD 64, RFC 3550, July 2003.

[RFC3540] SPRING, N., WETHERALL, D., & D. ELY, "*Robust Explicit Congestion Notification (ECN) Signaling with Nonces*", RFC 3540, June 2003.

[7] STEVENS, W, WESLEY A. "TCP*/IP Illustrated. Volume I: the Protocols*, 1994.

[RFC3309] STONE, J., STEWART, R., & D. OTIS, "*Stream Control Transmission Protocol (SCTP) Checksum Change"*, RFC 3309, September 2002.

[6] ZIMMERMANN, H. "*Reference Model — The ISO Model of Architecture for Open Systems Interconnection*" 1980.