Implementación de DCCP (Datagram Congestion Control Protocol) en espacio usuario

Guillermo Rigotti
UNICEN – Fac. de Ciencias Exactas-ISISTAN
Pje. Arroyo Seco, (7000) Tandil, Bs. As. Argentina
TE: +54-2293-439682 FAX: +54-2293-439681 Email: grigotti@exa.unicen.edu.ar

Abstract

Until recently time, TCP and UDP satisfied application´s requirements. This situation changed when real time multimedia applications appeared. As a consequence of its reliability, TCP does not enable applications to control the sending rate. This produces delays that are incompatible with the service to be offered by the application. UDP enables applications to control the sending rate; this is dangerous for the network, because it could result congested. To solve this situation, the IETF defined DCCP*(Datagram Congestion Control Protocol)*, a protocol specifically designed to support real time multimedia applications.
DCCP offers basic functionality: connection establishment and termination, support to incorporate congestion control strategies, and options negotiation. This allows it to adapt efficiently to the requirements of the different types of multimedia real time applications. Currently, a significant research activity related to DCCP is being carried out in the IETF. Among others, there are two main topics that deserve special attention, and that are being object of research: the definition of an application API and the incorporation and testing of congestion control mechanisms that can adapt to various types of applications. This paper presents an implementation of DCCP coded in Java, running in user space. Its main characteristics are portability and modularity. The goal of our work is to provide a way to experiment and to test the two aforementioned topics, application API and congestion control mechanisms. This alternative allows work with DCCP and the applications in any environment and without any risk for the operating system.
Keywords: DCCP, congestion control, real time multimedia applications.

Resumen

Hasta hace poco tiempo, el soporte de comunicaciones de TCP/IP, TCP y UDP, fue suficiente para satisfacer los requerimientos de las aplicaciones. Esto cambió con la aparición de las aplicaciones multimedia en tiempo real. TCP no permite a la aplicación controlar la tasa de envío, ocasionando demoras incompatibles con el servicio ofrecido. UDP permite controlar la tasa de envío, pero resulta peligroso ya que no provee control de congestión, pudiendo saturar de la red. Como respuesta a esta situación, la IETF definió DCCP (Datagram Congestion Control Protocol), un protocolo diseñado para soportar aplicaciones multimedia de tiempo real. DCCP se caracteriza por ofrecer mínima funcionalidad: conexión, soporte para control de congestión, y negociación de opciones de operación. Esto le permite adaptarse eficientemente a los requerimientos de los diferentes tipos de aplicaciones multimedia en tiempo real. Actualmente existe una actividad de investigación importante referida a DCCP en el ámbito de la IETF. Entre otros, hay dos aspectos relacionados con DCCP que merecen atención y que son objeto de investigación: la definición de una API a la aplicación y la incorporación y prueba de nuevos mecanismos de control de congestión. En este paper se presenta una implementación en Java de DCCP, en espacio usuario, cuyas principales características son la portabilidad y modularidad. El objetivo es posibilitar la rápida experimentación y prueba en los dos aspectos mencionados, API y métodos de control de congestión. Esta alternativa permite trabajar con DCCP y las aplicaciones en cualquier medioambiente y sin riesgos para el sistema operativo.
Palabras clave: DCCP, control de congestión, aplicaciones multimedia de tiempo real.

1. Introducción

DCCP (Datagram Congestion Control Protocol) [1] es un protocolo de nivel transporte que se originó en el ámbito de la Internet Engeneering Task Force (IETF) [2] como respuesta a la aparición de tipos de aplicaciones no tradicionales[1] que requieren servicios de transporte de datos diferentes a los ofrecidos por TCP[3] y UDP[4].

La actividad de investigación referida a DCCP se lleva a cabo principalmente en el DCCP Working Group, desde 2001. Actualmente se cuenta con un conjunto de standards (RFCs) que cubren diferentes aspectos del protocolo: especificación, alternativas de control de congestión, especificaciones de uso de DCCP con aplicaciones existentes y nuevas, y agregado de funcionalidad modular al protocolo, que lo haga atractivo a dichas aplicaciones.

El tipo de aplicaciones para el que fue diseñado DCCP, se caracteriza por generar flujos de información de larga duración, ser sensible a las demoras y tolerar pérdida o corrupción en los datos. El soporte tradicional ofrecido a nivel transporte, materializado por TCP y UDP no satisface estos requerimientos.

TCP se caracteriza por ser un protocolo que provee un servicio orientado a conexión, modo stream y confiable. Si bien el establecimiento de conexión es necesario para controlar el flujo de datos originado por las aplicaciones para las que fue concebido DCCP, la característica de confiabilidad y entrega ordenada de la información provista por TCP provoca efectos no deseados: por ejemplo, en una transmisión telefónica, es preferible recibir un datagram con datos erróneos, lo que provocará una interferencia no significativa para el receptor, que esperar el tiempo de recuperación impuesto por TCP, lo cual impediría la comunicación en tiempo real. Por otro lado, los mecanismos internos utilizados por TCP no son fácilmente adaptables a los requerimientos de DCCP, ya que los aspectos de confiabilidad y control de congestión se encuentran altamente relacionados.

UDP se caracteriza por ofrecer un servicio no orientado a conexión, no confiable. Las ventajas que presenta son su bajo overhead y la no interferencia con el envío de datos realizado por la aplicación, no imponiendo restricciones en la tasa de envío. Desde sus comienzos, UDP ha sido utilizado por aplicaciones que ocasionan poco tráfico en la red. Actualmente, es la única opción para soportar las aplicaciones multimedia de tiempo real, debido a las características no deseables de TCP. El problema que se presenta es que al no controlarse a nivel transporte el tráfico que se inyecta en la red, este tipo de aplicaciones pueden causar congestión, en la medida en que constituyan una parte significativa del tráfico en la Internet.

La alternativa de implementar control de congestión a nivel aplicación, presenta dos problemas, uno de ellos es que no siempre las aplicaciones lo implementarán, ya que es un aspecto complicado y sujeto a errores; por otra parte, en caso de que lo hicieran, seria un replica de la funcionalidad innecesaria.

Como respuesta a esta situación, surge DCCP, que es un protocolo orientado a conexión, de muy bajo overhead y adaptable a los diferentes requerimientos de las aplicaciones a través de mecanismos de negociación de opciones de operación. Entre las opciones, puede negociarse el tipo de control de congestión a utilizar. Si bien aún no esta ampliamente difundido, la expectativa es que sea el protocolo que reemplace a UDP para aplicaciones multimedia de tiempo real.

En la actualidad, DCCP se encuentra implementado en el kernel Linux 2.6.14 y posteriores [5] y en estado de desarrollo en BSD [6]. Existe además una implementación en lenguaje C, en espacio usuario en Linux [7], cuyo objetivo es ganar experiencia en varios aspectos del protocolo a través de un desarrollo rápido para luego portar la implementación al kernel.

En este trabajo se presenta una implementación de DCCP en espacio usuario, realizada en lenguaje Java, lo cual la hace portable a cualquier plataforma.

[1] Telefonía IP, multiconferencia, y en general aplicaciones multimedia de tiempo real.

Uno de los objetivos perseguidos consiste en proveer una base para experimentar con aplicaciones que por sus requerimientos sean candidatos a utilizar DCCP., en aspectos tales como la definición de APIs a nivel usuario (DCCP aplicación [8] [9]).
Otro aspecto de importancia lo constituye el desarrollo de heurísticas de control de congestión y su evaluación, en cuanto a su adaptabilidad a las aplicaciones y su compatibilidad con tráfico TCP.
Lo que distingue a esta implementación es su portabilidad, lo cual permite el uso y experimentación con DCCP entre cualquier par de equipos conectados a la Internet. Este aspecto es de importancia dada la escasa difusión que aún caracteriza a este protocolo..
El resto del trabajo está organizado de la siguiente manera: en la sección 2 se describen brevemente las características principales de DCCP, resaltando sus aspectos más importantes; en la sección 3 se describe la implementación presentada en este trabajo, de manera general; posteriormente, en las secciones 4, 5 y 6 se describen aspectos más específicos de la misma, respectivamente: threads de ejecución, interacción con las aplicaciones, y los módulos principales que componen la implementación. La sección 7 se refiere a las conclusiones, estado actual y continuación del trabajo. La sección 8 contiene las referencias bibliográficas.

2. Descripción de DCCP

DCCP es un protocolo de nivel transporte, al igual que TCP y UDP. Provee un servicio de transmisión de paquetes orientado a la conexión, no confiable y con control de congestión. Este protocolo provee la funcionalidad mínima necesaria, con el objeto de reducir al máximo el overhead y mantener un diseño simple. Deja a las aplicaciones funciones tales como FEC (forward error correction) y manejo de múltiples streams entre otras.
En lo que sigue de esta breve descripción, se remarcarán las diferencias con TCP; de esta manera se trata de aclarar la razón por la cual este protocolo no puede ser un subconjunto ni un derivado de aquél.

2.1 Formato de paquetes

DCCP define varios tipos de paquete, cada uno de ellos con funciones específicas, evitando el uso de los bits de flag, como hace TCP en su único formato de segmento.
Los paquetes DCCP tienen un formato simple, para su rápido proceso, permitiendo el agregado de opciones para diferentes funciones. Las opciones pueden extenderse hasta 1008 bytes. En este aspecto es más flexible que TCP, que tiene un campo de opciones limitado en longitud. En la figura 1 se muestra un paquete DCCP. Por razones de extensión, sólo se muestra el caso de números de secuencia extendidos, de 48 bits, en el caso de números de secuencia de 24 bits, se mantienen los campos mostrados. Los *ports* de origen y destino, al igual que en TCP, son los que juntamente con las direcciones IP determinan los endpoints de la conexión. El campo *data offset* indica dónde comienza, si hay, la zona de datos de la aplicación, determinando también la longitud del campo de opciones. *CCVal* es un valor de uso opcional, destinado al método de control de congestión, por lo que no se define aquí su función. *CsCov* indica qué parte del segmento DCCP está cubierta por el campo *checksum*:.además del header fijo y las opciones Puede especificarse qué parte de los datos de la aplicación deben ser cubiertos por el checksum, dependiendo de la sensibilidad de la misma a datos corruptos. El campo *Type* determina el tipo de paquete DCCP: hasta el momento hay definidos 10, los cuales se muestran en la figura 1. El campo *X* especifica si se usan números de secuencia cortos o largos, y por último, el campo *secuencia* especifica el número de secuencia del segmento DCCP.
En los segmentos DCCP_Request y DCCP_Response, se define un campo *Service Code*, que hace referencia a la aplicación a la cual sirve DCCP en el port (por ejemplo, RTP).
Todos los segmentos excepto DCCP_Request y DCCP_Data, llevan un subheader que contiene un número de Ack, que hace referencia al número de secuencia del paquete con mayor número y con header correcto que se ha recibido.

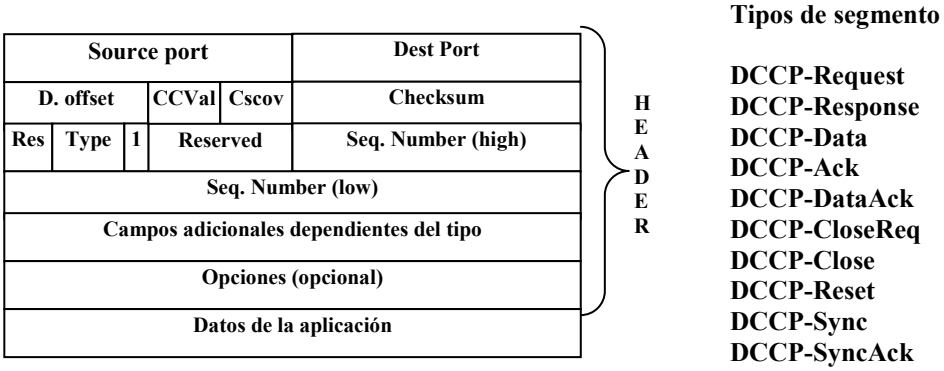

Fig. 1 Formato de segmentos DCCP

2,2 Funciones de los segmentos

Los segmentos que se utilizan para el establecimiento de la conexión son DCCP_Request, DCCP_Response y DCCP_Ack. Durante esta etapa se lleva a cabo normalmente la negociación de opciones de operación.

La terminación de la conexión se realiza, de manera normal, utilizando los segmentos DCCP_Close, DCCP_CloseReq (sólo el servidor) y DCCP_Reset. Esta último determina además el cierre incondicional de una conexión, debido a un error, por ejemplo la imposibilidad de negociar una opción (mandatory).

Los segmentos DCCP_Data, DCCP_DataAck .y DCCP_Ack se utilizan para el intercambio de datos, datos y asentimientos y asentimientos respectivamente.

Los segmentos DCCP_Sync y DCCP_SyncAck se utilizan para recuperar al DCCP de una situación de error, como conexiones parcialmente establecidas o grandes ráfagas de segmentos perdidos.

2.3 Manejo de conexiones

La conexión se establece a través del intercambio de paquetes específicos, (DCCP_Request, DCCP_Response y DCCP_Ack), no soportando conexión simultánea.

La terminación de la conexión involucra el intercambio de segmentos DCCP_CloseReq, DCCP_Close y DCCP_Reset. Ambos procedimientos son confiables.

Una conexión bidireccional es considerada separada en dos conexiones unidireccionales, comprendiendo cada una de ellas los datos que viajan en un sentido, y los asentimientos correspondientes a ellos, que viajan en el otro. Los datos y asentimientos de ambas conexiones unidireccionales se mezclan en los segmentos DCCP (piggybacking). La particularidad de considerar dos conexiones separadas es que cada una de ellas funciona independientemente de la otra en cuanto a los parámetros de operación y al método de control de congestión utilizado. Esta característica es de suna importancia en el tipo de aplicaciones al que está orientado DCCP. La opción *Init Cookie* permite al servidor no guardar estado de una conexión en vías de establecimiento.

2.4 Opciones

El manejo de opciones provisto por DCCP tiene una considerable importancia, ya que permite parametrizar el funcionamiento del protocolo, seleccionar el método de control de congestión a utilizar, y chequear el estado de la red, entre otras cosas. De esta manera, DCCP puede adaptarse a los diferentes tipos de aplicaciones que lo utilizan.

Desde el punto de vista del manejo de las opciones por parte del protocolo, podemos distinguir aquellas opciones que son procesadas por el receptor (por ejemplo el uso de checksum), aquellas que además implican una respuesta (a través del envío de otra opción – por ejemplo timestamp y timestamp echo-) por parte de este último, y aquellas que se refieres a negociación de parámetros de funcionamiento del protocolo (features).

Todos los intercambios de opciones pueden realizarse en cualquier fase del protocolo, pero los de negociación, generalmente se realizan en la fase de conexión..
Las características negociables del protocolo, denominadas features, se representan como valores almacenados y posibles de ser negociados con la otra parte y/o modificados a requerimiento de la aplicación. Por ejemplo, una feature se refiere al mecanismo de control de congestión a utilizar, teniendo almacenados dos valores, uno que representa el preferido, y otro alternativo. El mecanismo de negociación de opciones es simple y adaptado a las características de no confiabilidad del protocolo. Se basa en el intercambio de dos tipos de opción, Change, con la cual un lado indica al otro que desea cambiar el valor de una feature determinada y Confirm, que indica una respuesta (positiva o no) al Change. Las features son almacenadas en cada uno de los lados, con preferencias que pueden diferir. Las opciones Change y Confirm, deben referirse a una feature local al que inicia la solicitud de cambio, o remota, para lo cual se utilizan respectivamente los pares (Change_L, Confirm_R y Change_R, Confirm_L).
La confiabilidad de la negociación de opciones se basa en la repetición (limitada) de cada Change hasta obtener el correspondiente Confirm, y una protección contra opciones reordenadas en la red, basada en los números de secuencia de los segmentos.

2.5 Control de congestión

Como consecuencia de haber sido diseñado para soportar una diversidad de aplicaciones con diferentes requerimientos de control de congestión, DCCP soporta la negociación del método de control de congestión a ser utilizado, destinando además un porcentaje significativo (aproximadamente un 50%) de códigos de falla (reset), features y opciones para su uso por parte de métodos específicos de control de congestión. El método de control de congestión a utilizar se selecciona independientemente para cada conexión unidireccional de acuerdo a las necesidades de la aplicación.
Actualmente hay dos definidos : el identificado como CCID 2 [10] ofrece un control de congestión similar al de TCP, utilizando una ventana de congestión para mantener los paquetes enviados sin asentimiento. Cuando detecta congestión (paquetes descartados o marcados ECN) baja a la mitad la tasa de envío. Para determinar qué segmentos fueron bien recibidos en una ventana, se utiliza un mecanismo similar a SACKs.[11].
CCID 3 [12] aplica TCP-Friendly Rate Control (TFRC). El emisor mantiene una tasa de transmisión que va actualizando con las estimaciones del receptor acerca de paquetes erróneos y marcados como ECN.
Debe tenerse en cuenta que si bien ambos métodos de control de congestión son "TCP friendly"[2], esto no es un requisito impuesto por DCCP,

3. Generalidades de la implementación de DCCP

El trabajo presentado consiste en la implementación de la funcionalidad de DCCP de acuerdo al standard [1]. Esta implementación, realizada en Java, tiene las características de ser portable y modular.
La portabilidad fue un aspecto al que se dio fundamental importancia, ya que de esta manera, el protocolo puede ser ejecutado en cualquier plataforma que soporte Java, pudiendo por lo tanto probarse entre cualquier par de equipos conectados a la Internet, y no sólo aquellos con sistemas operativos que incluyan soporte DCCP (Linux con kernel 2.6 o BSD –experimental-)
La modularidad permite incluir fácilmente nuevos métodos de control de congestión y opciones[3], sin tener que modificar el código del DCCP.

[2] "TCP friendly" se llama a un protocolo que en el largo plazo no excede el ancho de banda normal usado por una conexión TCP en igules condiciones.
[3] La inclusión de este tipo de funciones es un aspecto importante del protocolo.

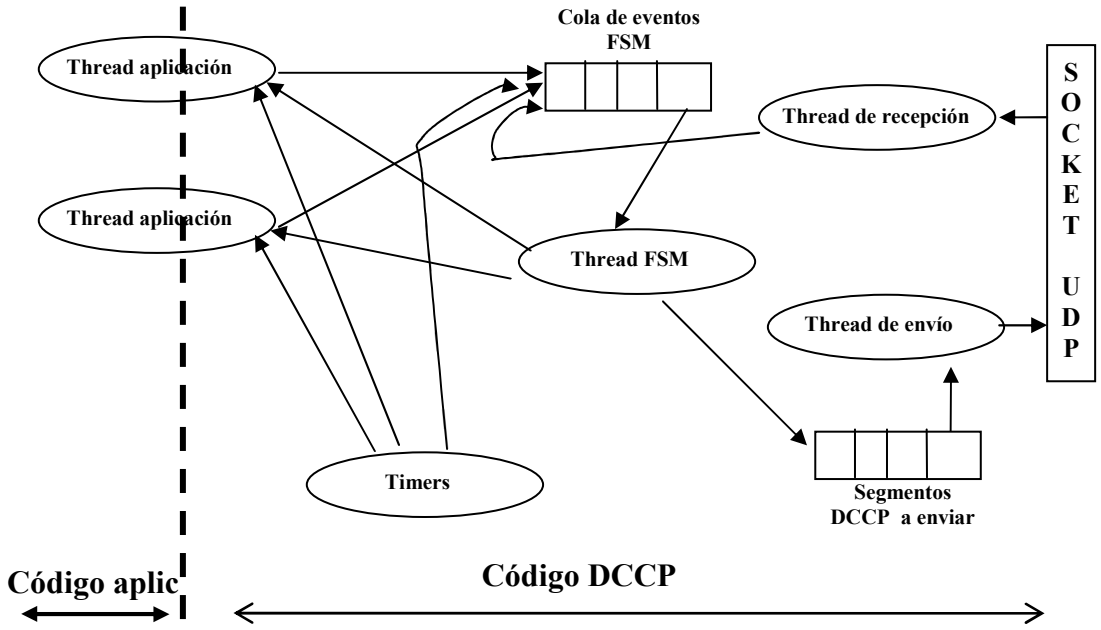

Fig. 2. Esquema simplificado de los threads que componen la totalidad del proceso (DCCP más aplicación). Los threads se muestran como elipses y sus interacciones con flechas. Las interacciones son sincronizadas. Los threads de la aplicación (uno o más) se bloquean en el código DCCP hasta que son satisfechos sus requerimientos o son informados de otras situaciones a través de excepciones

Al estar implementado en espacio usuario, las modificaciones realizadas no representan riesgos para el sistema, y pueden realizarse de manera rápida y sencilla. Posteriormente, dicha funcionalidad podrá ser portada al kernel[4].

Una vez completo, la implementación de DCCP permitirá investigar sobre aspectos de importancia del protocolo, casi sin abordar en la actualidad, como los siguientes:

Desarrollo de nuevos métodos de control de congestión orientados a necesidades de diferentes tipos de aplicaciones.

Evaluación de dichos métodos en cuanto a su adaptación a la aplicación, y en cuanto al uso que hacen de la red y de qué manera son compatibles con el tráfico TCP.

Desarrollo de una API a nivel usuario (DCCP/Aplicación). Este aspecto debe ser aún analizado ya que los requerimientos de las aplicaciones difieren significativamente, por ejemplo algunas necesitarán saber el estado en que llegaros ciertos bloques o si algunos fueron marcados como ECN, etc.

Experimentación y definición de la manera de utilizar DCCP por parte de otros protocolos, por ejemplo Real Time Protocol (RTP) [13]..

Se tomó la decisión de utilizar UDP como soporte de comunicaciones, debido a que este protocolo no impone restricciones de envío ni genera un overhead significativo. La otra alternativa, uso de IP en forma directa (raw sockets) fue considerada, pero se descartó ya que no es posible disponer de librerías Java 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. Un problema que surge de la utilización de UDP, es que a nivel sockets no se puede proveer una indicación de congestión (ECN), sin embargo, no todos los routers implementan dicha característica.

[4] En los casos en que se cuente con soporte DCCP en el sistema operativo.

Otra limitación que surge del uso de UDP es la imposibilidad de comunicación con versiones de DCCP standard Sin embargo, previendo la necesidad de esta prueba, se utilizará algún soporte de amplia difusión para acceso al nivel de red, tal como libpcap [14] desde un ambiente Linux, con una librería de acceso desde Java (Jpcap [15]), a efectos de poder interactuar directamente con IP y confrontar este desarrollo con implementaciones standard de DCCP.
A pesar de encapsular el segmento DCCP en UDP, se mantienen exactamente los formatos de paquetes en cuanto a la longitud de los campos, para no introducir mayor overhead y poder utilizar longitudes razonables para los segmentos DCCP,

4. Threads de ejecución

Para lograr una implementación clara y modular, se utilizó la capacidad de threads provista por Java. Se definió de esta manera un thread de lectura y otro de escritura sobre el socket UDP utilizado para comunicación, encargados respectivamente de recibir/desencapsular y encapsular/enviar los segmentos DCCP contenidos en los datagrams UDP.
Un thread adicional, implementa la FSM definida en [1], con sus 9 estados; este thread procesa eventos depositados por los demás threads en una cola de eventos en forma continua, dando lugar a los procesos DCCP (incluidas las invocaciones a métodos de control de congestión, negociación de opciones, etc).
Por último, uno o más threads definidos por la aplicación, ejecutan código DCCP (encargado de atender requerimientos de la aplicación) desencadenando eventos que son colocados en la cola de la FSM para su proceso. En ls figura 2 se muestra la interacción entre los threads mencionados.
Además de los threads mencionados, debemos considerar el proceso asincrónico realizado por los timers, ya que estos pueden afectar variables utilizadas por los primeros.

5. Interacción DCCP/Aplicación

En el caso standard, un DCCP implementado en espacio kernel, la interacción entre éste y la aplicación se realiza a través de sockets, como en TCP y UDP. Sin embargo, a diferencia de lo que ocurre en estos últimos, en el caso de DCCP dicha interacción se vuelve más compleja debido a los requerimientos especiales de cada tipo de aplicación.
Por ejemplo, en una aplicación que tolera la pérdida de paquetes, se podría requerir de DCCP una información detallada acerca de qué paquetes se han perdido, o bien la aplicación podría querer informar a DCCP qué paquetes ha descartado debido a su llegada fuera de término,
Otro aspecto de esta interacción, es el relacionado con la configuración inicial (incluido el método de control de congestión a utilizar) y le especificación de modos de operación por parte de la aplicación, que da lugar a la negociación de opciones durante el curso de la conexión.
Estos aspectos de la interacción no están aún definidos, debido a su complejidad y a la variedad de aplicaciones y requerimientos. Algunos intentos de definir esta API, que implica la modificación de los llamados a los sockets, pueden encontrarse en [16]
En el caso de la implementación que se presenta, estos aspectos se encuentran aún casi sin definir, habiendo implementado los mecanismos básicos que permiten una interacción simple de DCCP con las aplicaciones.
Para este mecanismo de interacción entre DCCP y la aplicación; se optó por proveer un conjunto de llamadas bloqueantes, con la misma filosofía de java, y el uso de excepciones (DCCPException) para informar al thread de la aplicación de situaciones anormales. En el caso de contar con un soporte DCCP en el kernel y librerías Java que soportaran un API apropiada sobre los sockets, la migración de las aplicaciones sería casi inmediata.
La funcionalidad definida hasta el momento, es la siguiente:

- Especificación de el o los Sevice Codes, a efectos de poder especificar el o los tipos de aplicación soportados por el port[5].

[5] Uno en el caso de un cliente, y uno o más en el caso de un servidor.

- Indicación a la aplicación de la señal de receptor lento (Slow Receiver) para que la aplicación emisora pueda aplicar un comportamiento acorde.
- Especificación por parte de la aplicación, de un número máximo de reintentos de negociación de opciones, para no crear ciclos.
- Especificación por parte de la aplicación de la semántica de descarte de paquetes
- Especificación por parte de la aplicación, del cambio de lista de preferencias para una opción negociable determinada.

6. Módulos que componen la implementación

En esta sección se describen los módulos de mayor importancia. Algunos de ellos se corresponden con threads descriptos anteriormente (por ejemplo la máquina de estados DCCP, y los módulos de emisión y recepción**).**

6.1 Inicialización e interfaz con la aplicación

Las funciones de inicialización se realizan al crear la instancia DCCP. Consisten en la creación de los threads, en la inicialización de variables del generales del DCCP y en la configuración del modo de operación por defecto; éste podrá ser posteriormente modificado por la aplicación.

La función más importante de este módulo es servir de nexo entre la aplicación y el DCCP, y sincronizar su operación. Las invocaciones de la aplicación recibidas por este módulo son tratadas de manera tal que sean compatibles con las invocaciones provistas por java al nivel de transporte, es decir, llamadas bloqueantes con anuncio de situaciones anormales a través del mecanismo de excepciones, en este caso, utilizando el tipo DCCPException.

En estas interacciones deben considerarse dos aspectos: el mecanismo de la interacción, que se encuentra totalmente definido, y las interacciones en sí, que irán siendo definidas a medida que se defina un API entre el DCCP y la aplicación; este aspecto es objeto de investigación [16].

La sincronización entre el thread de la aplicación y la FSM DCCP se realiza de la siguiente manera: el thread de la aplicación ejecuta código de la aplicación hasta que produce una invocación a DCCP (por ejemplo read, connect o cambio en la lista de preferencias para una feature); en este momento, se produce un evento en la FSM DCCP, de acuerdo a lo solicitado por la aplicación: por ejemplo la invocación a read produce un evento receive a ser procesado por la FSM. Una vez colocado el evento en la cola de eventos de la FSM, el thread originado por la aplicación quedará esperando la resolución de lo solicitado a DCCP. Dicha espera se produce a través del testeo de una variable incluida en el evento producido, pudiendo finalizar por éxito en lo solicitado o a través de una excepción, si se producen condiciones anormales[6]. Dicha excepción deberá ser tratada por código provisto por la aplicación.

Hay casos en los que la llamada a DCCP se resuelve inmediatamente, por ejemplo cuando se requiere información acerca del valor de una variable local[7]. En otros casos, el evento producido puede o no dar una respuesta a la aplicación. Por ejemplo, en el caso de un read, es posible que cuando se genera el evento correspondiente, la FSM tenga un frame de la aplicación listo para ser leído, que se devuelve a la aplicación, con lo cual finaliza la invocación con el fin del proceso del evento. Sin embargo, podría ocurrir que aún no hubiera un frame disponible, y como el llamado es bloqueante, se debería esperar hasta el arribo o hasta que venza el timer de espera de la aplicación. En ninguno de estos casos es posible demorar el evento de la FSM hasta que se produzca alguna de las alternativas. La solución que se adopta consiste en que el código invocado por la aplicación quede en un ciclo, en el cual realiza repetidas invocaciones a la FSM, a intervalos regulares, hasta que se tiene éxito o se produce una condición de error. De esta manera, los eventos se resuelven a ritmo de la FSM, sin interferir con otros que ella deba procesar. Los posibles casos en un read son la

[6] Se mantiene en todos los casos el mecanismo usual de retorno de llamadas a las funciones de comunicación utilizado por Java.

[7] En estos casos, en general, no es necesario producir un evento a la FSM, resolviéndose la invocación a través de una consulta a una variable.

recepción de los datos, retorno normal, o un error de recepción tardía, que sería indicado por la correspondiente DCCPException..
Otro caso lo constituye el de un open, ya que aquí hay un único evento producido como consecuencia de la aplicación, pero debe esperarse por una reacción del servidor DCCP. No es posible ni tiene sentido generar repetidos eventos open, sino que se debe memorizar el evento original, para luego poder entregar el resultado. Para lograr esto, se introduce una variable en el código de la FSM, que recuerda el evento open; luego, cuando se recibe el evento correspondiente, que determina si el open fue o no aceptado, la FSM lo anunciará al código de invocación de DCCP, quien procederá de la misma manera que la descripta en el caso anterior. Debe notarse que es posible tener múltiples invocaciones concurrentes en el caso de un read o write, pero se permite una única invocación a open[8].

6.2 Máquina de estados DCCP

Este módulo implementa la máquina de estados definida en [4340], compuesta de 9 estados que incluyen las fases de establecimiento y terminación de la conexión y la transferencia de datos. Se agregan estados y eventos propios de la implementación por razones de eficiencia.
Este código es ejecutado continuamente, procesando eventos almacenados en una cola a ese efecto; estos eventos corresponden a los siguientes sucesos:
Llegada de segmentos DCCP remotos, entregados por el thread de recepción. Estos son analizados y de acuerdo al estado de la FSM y al tipo de segmento, producen cambios y/o son entregados a otros módulos para continuar su proceso (por ejemplo al módulo de manejo de opciones o al de control de congestión, según contengan opciones o datos y/o asentimientos).
Vencimiento de timers directamente relacionados con la operación de la FSM (no incluye otros tales como timers de retransmisión para negociación de opciones, etc.). Estos eventos podrán producir, dependiendo del caso, cambio de estado de la FSM, notificaciones a la aplicación, o cierre de la conexión.
Eventos producidos por el módulo de interfaz con la aplicación como consecuencia de requerimientos de esta última. En algunos casos, estos eventos producirán cambios de estado en la FSM.
Además de los mencionados, se agregaron dos eventos propios de la implementación para mejorar la eficiencia y simplificar el código. Se utilizan en situaciones especiales, que como en todos los casos requieren un proceso ordenado de los eventos ya producidos y aún no ejecutados por la FSM. Estos eventos, que no alteran el comportamiento descrito en [1], son CANCEL_INCOMING , que es producido cuando desde la aplicación se cancela un Listen previo (sólo en el servidor), con lo cual se debe volver al estado inicial, y RESET que indica una condición excepcional que hará que la FSM prepare la terminación de la conexión, enviando la causa del reset a todos los eventos pendientes.
Cada evento consta de un código que lo identifica, un resultado de la operación que es colocado por la FSM, y un objeto dependiente del tipo de evento. En algunos casos de eventos generados por la aplicación, éstos conservan el resultado de la operación y son consultados por aquella luego de su ejecución por parte de la FSM (por ejemplo, establecimiento de la conexión).
En resumen, los eventos definidos hasta el momento, son los siguientes.
1-Los producidos por el thread de al aplicación, se corresponden con las llamadas que ésta hace a DCCP: por el momento. Algunos de estos eventos están relacionados con la API DCCP/aplicación.
2-Evento de recepción de segmentos, generado por el thread de recepción, que contiene el segmento DCCP recibido desde el lado remoto.
3-Los eventos internos de la implementación para proceso ordenado de los eventos ante un cierre de la conexión o cancelación de espera por requerimientos de conexión (RESET y CANCEL_INCOMING).

[8] Esto es controlado por la FSM, al analizar el par estado/evento.

4-El evento generado por el timer que controla, en ciertos casos de terminación de conexión, el tiempo de espera para la reutilización del port involucrado (TIMEWAIT).

6.3. Manejo de opciones

Como fue mencionado en la sección 2.4, una de las características de DCCP es su versatilidad en el manejo de opciones, lo que le permite, entre otras de sus características, adaptarse a diferentes tipos de aplicación. Una descripción detallada de las opciones definidas hasta el momento se encuentra en [1][9].

El manejo de opciones presenta características que justifican su implementación por separado del resto de los módulos:

- Opciones que requieren proceso dentro del módulo
- Opciones de negociación de features, que requieren timers y reintentos.
- Envío de opciones sujeto al control de flujo en uso pero independiente de los datos y/o segmentos a enviar.
- Interacción con la aplicación para la selección de valores de features.
- Previsión de agregado de nuevas opciones y features.
- Diferentes procesos de negociación de features.

A continuación se describen las interacciones entre el manejo de opciones y el resto de la implementación.
En el arranque de DCCP, se invocan los métodos de inicialización de opciones y de features; éstos se encargan de inicializar las opciones implementadas, con los valores por defecto. Posteriormente, se realizará la posible negociación de algunos valores (features); esta negociación se dará al comienzo debido a requerimientos de la aplicación (especificadas al solicitar la conexión a travès de la API a definir) o a diferencias de valores iniciales entre ambos DCCPs, o durante el transcurso de la conexión, debido a requerimientos de las aplicaciones o bien a la recepción de opciones de negociación del lado remoto. .
El uso de otras opciones (por ejemplo las de medición de tiempos), estará determinado por el propio DCCP, y por el método de control de congestión en uso.
Por el momento y debido a la funcionalidad de las opciones definidas en [1] se decidió implementarlas en el módulo de opciones y no por separado. Un caso especial es el método de control de congestión utilizado, que se implementa de manera independiente.
La comunicación entre el módulo de manejo de opciones y el envío de los segmentos, se produce a través de una lista de opciones a enviar.
Para el envío de opciones, la función de construcción de segmentos DCCP (invocada por la FSM y sujeta al control de congestión utilizado) consulta la lista de opciones a enviar, agregándolas en el segmento a enviar, siempre que sean aceptables para el tipo de segmento.
Las opciones, son generadas a través de requerimientos al módulo de opciones. Estos requerimientos son realizados por el módulo de negociación, por requerimientos del DCCP (por ejemplo si se ha configurado el uso de NDP), por requerimientos de la aplicación, que pueden dar lugar al cambio de la lista de preferencias para alguna opción negociable, o por el método de control de congestión en uso (opciones reservadas para el CCID).
A continuación se describen de manera simplificada las interacciones entre el módulo de manejo de opciones y el resto de la implementación

1. Llegada segmento DCCP (**recepción)**
   1.1 separación de opciones (**FSM DCCP**)

[9] Se prevé que al ir extendiéndose DCCP a soportar diferentes tipos de aplicación, se cree una cantidad considerable de nuevas opciones y features; esto se encuentra reflejado en el significativo porcentaje de valores reservados para los dos tipos.

1.2 proceso de opciones remotas (**opciones**)
1.2.1 Modificación de parámetros de funcionamiento (p.ej cambio de NDP)
1.2.2 Generación de opción de respuesta (p. ej. timestamp echo)(**opciones**)
1.2.3 Generación de opciones de respuesta a una negociación (Confirms) (**opciones**)
2. Vencimiento de timer de envío de Change
2.1 Generación de nuevo Change, si no esta en lista de opciones a enviar(**opciones**)
3. Modificación de lista de preferencias (aplicación u otro módulo)
3.1 Invocación a la feature, y luego posible generación de opción Change (**opciones)**
4. Envío de segmento (FSM). Extracción de opciones de la lista **(opciones)**

6.4. Emisión y recepción

Estos módulos se corresponden con los respectivos threads, y su función es la comunicación del DCCP local con el remoto.
El módulo de emisión inicializa el socket UDP de envío, y luego se ejecuta continuamente generando segmentos DCCP en función de datos e información de control suministrados ya sea por la FSM o por el método de control de congestión, y colocados en una cola de segmentos a enviar. De acuerdo las características del control de congestión, el envío puede ser bloqueado temporalmente.
El thread de recepción, sólo se encarga de desencapsular información recibida por el socket UDP, y entregarla, en forma de evento, a la FSM. Sería posible incluir en él un chequeo que permita descartar segmentos no válidos (independientemente de las condiciones del DCCP local) que de todas maneras serán ignorados, para mejorar la eficiencia.

6.5. Control de congestión

El módulo de control de congestión no se trata aquí debido a que depende del tipo de aplicación soportado. De acuerdo a sus necesidades, se selecciona el adecuado durante el establecimiento de la conexión. Los dos tipos de control de congestión definidos por el grupo de trabajo de DCCP [10] y [12] están actualmente siendo implementados.

7. Estado actual, conclusiones y continuación

Actualmente, se encuentran implementados y en funcionamiento los mecanismos de interacción DCCP/Aplicación, el establecimiento y la terminación de conexión, el manejo de opciones (incluido el código de las más simples) y el envío de segmentos DCCP. Se encuentran en estado de desarrollo los métodos de control de congestión ya mencionados, previéndose que cuando se los incorpore al protocolo se ajustará la interacción entre éste y los primeros.
La aplicación desarrollada será útil para experimentar con el API ofrecido a las aplicaciones y para probar las heurísticas existentes de control de congestión, y posiblemente desarrollar otras para nuevos requerimientos de las aplicaciones. Estos aspectos son de importancia ya que DCCP no está aún muy difundido.
Un problema de la implementación consiste en que DCCP se encapsula en UDP, no pudiendo de esta manera comunicarse con implementaciones DCCP en el kernel, que se encapsulan en IP. Otra consecuencia del uso de UDP es que no es posible acceder a información de IP, en particular a los bits ECN, con lo cual algunos métodos de control de congestión perderían información útil para tomar sus decisiones. Para salvar estos inconvenientes, sería posible utilizar librerías no standard Java, que permiten el acceso directo a IP [14] [15].
No sería razonable tratar de migrar la implementación presentada a espacio kernel, ya que la estructura de la misma no podría adaptarse fácilmente (además del cambio de Java a C) a operar en el entorno del sistema operativo. Sin embargo, el desarrollo y prueba rápida y sin riesgo para el sistema que posibilita esta implementación en espacio usuario, fundamentalmente de los aspectos mencionados (API y métodos de control de congestión) permite que sólo haya que modificar el kernel cuando la funcionalidad a agregar está suficientemente probada.

8. Referencias

[1] E. Kohler, M. Handley, S. Floyd, “Datagram Congestion Control Protocol (DCCP)”, RFC 4340, March 2006.
[2] http://www.ietf.org
[3] J. Postel, “Transmission Control Protocol, DARPA Internet Program Protocol Specification”, RFC 783, 1981
[4] J. Postel, “User Datagram Protocol”, RFC 768, 1980
[5] http://linux-net.osdl.org/index.php/DCCP
[6] http://www.jp.nishida.org/dccp
[7]Alkis, Evlogimenos, Khian Hao Lim, "On the Implementation of Datagram Congestion Control Protocol" 2002, www.cs.ucsd.edu/~tsohn/projects/dccp/index.html
[8] Damon Lanphear, “Datagram Congestion Control Protocol (DCCP) User Guide”, draft-ietf-dccp-user-guide-00, October 2002.
[9] T. Phelan, “Datagram Congestion Control Protocol (DCCP) User Guide”, draft-ietf-dccp-user-guide-02, July 2004
[10] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, March 2006.
[11]. M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, “TCP Selective Acknowledgment Options”, RFC 2018, October 1996
[12] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.
[13] C. Perkins, “RTP and the Datagram Congestion Control Protocol (DCCP)”, draft-ietf-dccp-rtp-07.txt, June 2007.
[14] libpcap Official Site, http://www.tcpdump.org/
[15]Jpcap Official Site, http://sourceforge.net/projects/jpcap/
[16] http://www.charles-guillemot.info/dccp.html#annexe