# DISEÑO E IMPLEMENTACIÓN DE UN PROTOCOLO DE TRANSPORTE MULTICAST PARA LA RÉPLICA MASIVA DE INFORMACIÓN

*P. Manzanares López* *J. Malgosa Sanahuja* *J. García Haro* *J.C. Sánchez Aarnoutse*

Departamento de Tecnologías de la Información y las Comunicaciones
Universidad Politécnica de Cartagena
{pilar.manzanares, josem.malgosa, joang.haro, juanc.sanchez}@upct.es

**ABSTRACT**

In this paper, we introduce a *multicast* transport protocol (MTP, *Must Transport Protocol*) used in a *multicast* application called MUST (*MUlticast Synchronous Transfer*) that allows the replication of hard disk partitions in a fast and efficient way. Using this protocol, the number of acknowledgement frames are dramatically reduced, allowing the employment of MUST in LAN and intra-campus (directly interconnected LANs via few routers) scenarios. Additionally, the application has been programmed in C language using standard linux *kernel* routines. Therefore, we can distribute our application world wide without incurring legal conflicts.

## 1. INTRODUCCIÓN

MUST es una aplicación *multicast* que permite la réplica rápida y eficiente del contenido de los discos duros de ordenadores conectados en entornos LAN e intra-campus.

Desafortunadamente, TCP no está preparado para ser utilizado en entornos multipunto; por ello, la complejidad de la aplicación MUST reside básicamente en la definición de su protocolo de transporte. El protocolo propuesto en este artículo (MTP, *MUST Transport Protocol*) es capaz de reducir sensiblemente el número de tramas de reconocimiento (ACK) que viajan por la red, independientemente del número de usuarios conectados. Además, también disminuye el efecto nocivo que las asimetrías en la carga de los enlaces de la red producen sobre el algoritmo de control de flujo.

MUST se ha programado en lenguaje C utilizando librerías estándares del *kernel* de Linux. No obstante, la metodología de trabajo es exportable a otros sistemas operativos propietarios.

## 2. MUST TRANSPORT PROTOCOL (MTP)

Cuando una aplicación *multicast* no presenta una gran complejidad, puede utilizarse UDP como único protocolo de transporte. En nuestro caso, aunque esta solución no es válida, hemos preferido codificar nuestro propio protocolo de transporte por encima de UDP [1]. Esta alternativa pueda parecer redundante, pero hay que tener presente que este esquema tiene la ventaja de que las tareas asociadas a la detección de errores las realiza automáticamente el *kernel*. Lógicamente, toda la complejidad clásica asociada al nivel de transporte (control de flujo, corrección de errores extremo a extremo, etc.) más la asociada a multipunto se programa en el nivel de transporte aquí presentado (MTP).

En las siguientes secciones se describirán sucintamente los algoritmos concretos propuestos para controlar el flujo de información, para reducir al máximo el número de reconocimientos (efecto *feedback implosion*) y para corregir los posibles errores en la transmisión de los paquetes. Posteriormente, se detallan las modificaciones más importantes que deben aplicarse al algoritmo base para que pueda trabajar también en un entorno intra-campus. Finalmente, en la última sección se describen las conclusiones más relevantes del presente trabajo.

### 2.1. Control de flujo

Las excelentes prestaciones que presentan las actuales redes LAN junto con la naturaleza *half-duplex* de sus algoritmos de control de acceso al medio asociados (CSMA/CD y CSMA/CA en redes inalámbricas), permiten la implementación de un control de flujo relativamente sencillo.

Un cliente podrá estar en estado R (*Receive*) esperando recibir un paquete MTP, o en estado A (*ACK*) intentando confirmar la recepción del paquete recibido anteriormente (este último proceso se lleva a cabo teniendo presente la correlación existente entre todos los clientes, ver sección 2.2).

El servidor, durante la transmisión de un paquete, estará en estado S (*Send*). Al finalizar la transmisión, pasa al estado W (*Wait*) esperando recibir una trama ACK que confirme el último paquete transmitido o hasta que expire un determinado temporizador (*time-out*). Cuando el estado W termina, el servidor vuelve al estado S e inicia la transmisión del siguiente paquete de información. Nótese que el algoritmo descarta todas las tramas de reconocimiento relativas a paquetes de información anteriores.

Si el temporizador expira, se aumenta su valor multiplicándolo por un factor $\alpha$ (con $\alpha>1$). Por contra, si el reconocimiento (trama ACK) se recibe mientras el sistema está en el estado W, el nuevo valor del *time-out* se reduce de la siguiente manera:

$$T_{out} = max\{T_{out}/2, default\_T_{out}\}$$

En consecuencia, en los instantes de tiempo en los cuales la red se congestione, el valor de $T_{out}$ se incrementará, disminuyendo por tanto la velocidad de transmisión. Cuando la congestión desaparece, la redefinición del $T_{out}$ permite aumentar nuevamente la velocidad de transferencia de paquetes.

### 2.2. Feedback Implosion

Para reducir el número de tramas ACK que debe procesar el servidor, el protocolo MTP envía los reconocimientos hacia todos los miembros del grupo y no únicamente hacia el servidor. De esta forma, se fuerza a que exista cierta correlación entre los ACKs de los clientes; así, basta con que uno de los clientes envíe la confirmación correspondiente.

En el algoritmo propuesto, justo antes de transmitir la trama ACK, el cliente debe esperar durante un tiempo aleatorio (llamado ARTP, *ACK Random Time Period*) y simultáneamente escuchar si otro cliente ha transmitido ya por el canal la misma trama de reconocimiento. En caso afirmativo, el cliente inhibe su propia transmisión.

Mediante este algoritmo se consigue enviar una única trama de reconocimiento hacia el servidor, independientemente del número de miembros que contenga el grupo *multicast*. Como veremos posteriormente, en entornos intra-campus el servidor sólo recibe un ACK por cada LAN interconectada.

### 2.3. Corrección de errores

Hay que tener presente que el protocolo de transporte propuesto trabaja sobre UDP. Como ya es sabido, cuando UDP recibe un paquete erróneo, éste simplemente lo descarta. Por este motivo, MTP no es capaz de detectar la presencia de ninguno de estos paquetes, por lo que los errores en transmisión equivalen, a nivel MTP, a recibir desordenadamente el resto de la información.

Este efecto no es en principio muy grave puesto que en nuestra propuesta, el número de secuencia del paquete MTP coincide con el número asociado al sector físico del disco. Ello permite que los clientes puedan almacenar un sector cualquiera en su correspondiente posición dentro del disco duro. En consecuencia, no es necesario que los paquetes de información MTP se reciban en un orden secuencial.

Cuando se haya transmitido la imagen completa del disco duro, los clientes analizan el contenido de su propio disco, detectando la presencia de sectores vacíos. En caso afirmativo, el cliente enviará al servidor una trama del tipo *Repair-Request* pidiendo la retransmisión completa del sector en cuestión. El servidor responderá con una trama del tipo *Repair-Reply*, en la que se encapsula el sector físico demandado.

Con este algoritmo, el caudal conseguido por la aplicación es máximo. Se ha seleccionado este algoritmo de control de flujo/errores de alto caudal debido a que, en las actuales redes LAN, la probabilidad de error por bit es extremadamente pequeña.

### 2.4. Cabecera propuesta

La cabecera propuesta contiene cuatro campos: SN (*Sequence Number*), TOP (*Type Of Packet*), BP (*Broken Packet*) y PL (*Payload*). El campo TOP permite distinguir entre un paquete de información, una trama ACK, un *Repair-Request* o un *Repair-Response*. El campo BP es necesario ya que es posible que un sector no quepa en un único paquete MTP (normalmente, el tamaño de un sector es de 512 bytes, pero podría ser también de 1024 bytes, y los datagramas UDP no acostumbran a superar los 512 bytes). El campo PL indica la longitud total (en bytes) del paquete MTP. Finalmente, la cabecera es seguida por los bytes de información.

## 3. EXTENSIONES PARA ENTORNOS INTRA-CAMPUS

En entornos intra-campus, el servidor deberá recibir un único ACK por cada una de las LAN. Para ello, deberá conocer el número total de redes LAN interconectadas; este parámetro puede ser averiguado por la propia aplicación en tiempo de ejecución o introducido por el usuario antes de iniciar el proceso de réplica.

También hay que tener presente que, puesto que los ACK son emitidos en modo *multicast*, éstos serán transmitidos hacia todos los clientes de la red intra-campus, lo que puede confundir al algoritmo propuesto para reducir el número de tramas de reconocimiento. La solución que se propone consiste en que todos los clientes deben extraer la dirección unicast de cualquier paquete ACK (esta tarea puede llevarse a cabo fácilmente puesto que la dirección fuente de todos los paquetes IP *multicast* tienen asignada la dirección *unicast* del cliente) y descartarlo siempre que éste contenga una dirección de red diferente a la del cliente.

Estas dos extensiones permiten sincronizar la velocidad de transferencia de información entre todas las redes LANs.

## 4. CONCLUSIONES

En este artículo se ha propuesto un protocolo de transporte para replicar información llamado MTP, el cual garantiza un esquema de réplica libre de errores y de elevado caudal. Los resultados obtenidos en entonos LAN han sido buenos. Por ejemplo, la réplica de un disco duro de 5 Gbytes en 10 clientes simultáneamente llevó aproximadamente 2 horas.

Sin embargo quedan aspectos que han de ser solventados en el futuro. Se debe encontrar, mediante simulaciones y análisis, valores óptimos de los parámetros $T_{out}$, $\alpha$ o ARTP para una amplia variedad de escenarios. Además, MTP debe ser desarrollado para obtener resultados eficientes en entornos WAN.

El hecho de que todos los clientes compartan la misma información puede ser aprovechado por las tareas de corrección de errores [2], de forma que cualquier cliente pudiera ayudar en este aspecto al servidor. Por tanto, probablemente la arquitectura cliente-servidor no sea la más adecuada para futuras versiones de MUST.

**Agradecimientos**

Este trabajo ha sido financiado por los proyectos nacionales FAR-IP (TIC2000-1734-C03-03) y MTCES (TIC2001-3339-C02-02).

**Referencias**

[1] J. Malgosa, J. García Haro, F. Cerdan, F. Burrull. “MUST, a Multicast Synchronous Transfer Application for Fast Intra-Campus Replication”, IEEE Melecon’02.

[2] T. Speakman, et al., "PGM Reliable Transport Protocol Specification”, RFC 3208, December 2001.