JTCP: una implementación de TCP orientada a la evaluación de técnicas de control de congestión

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

TCP is a widely spread transport protocol on the Internet; great part of the traffic comes from applications that use TCP. This feature indicates the importance of the rate to which TCP introduces their data in the network. The rate of transmission must adapt to the requirements of the application, without saturating the network. Data rates that do not adapt to the conditions of the network could make it collapse. TCP has been adapted successful to fulfill their objectives: to obtain an efficient use of the network and to offer a suitable service to the users. This adaptation has been consequence of a considerable research activity, consistent mainly in the development of congestion control mechanisms. This work presents the development of JTCP, a protocol with the basic functionality of TCP, portable and easily modifiable. JTCP is totally implemented in Java, and is independent of the operating system that supports it. JTCP design allows to easily interchange congestión control techniques used by the protocol. Its portability facilitates the test of those congestion control techniques between any pair of hosts connected to the Internet. The objective of the present work is to make possible the evaluation of alternatives for congestion control in TCP, in different operation environments, such as those with high bandwidth (optic fiber links) or those with a high rate of errors (wireless links).

Keywords: TCP, TCP congestion control, TCP performance.

Resumen

TCP es un protocolo ampliamente difundido en la Internet. Gran parte del tráfico de la Internet, proviene de aplicaciones que lo utilizan. Esta característica indica la importancia de la tasa a la cual TCP introduce sus datos en la red. La tasa de envío debe adaptarse a las necesidades de la aplicación, sin saturar la red. El envío de datos a tasas que no se adapten a las condiciones de la red podria hacerla colapsar. Desde que fue definido, el protocolo TCP ha sido adaptado exitosamente para cumplir con sus objetivos de lograr un uso eficiente de la red y ofrecer un servicio adecuado a los usuarios. Esta adaptación ha sido consecuencia de una considerable actividad de investigación, consistente principalmente en el desarrollo de mecanismos de control de congestión. Este trabajo presenta el desarrollo de JTCP, un protocolo con la funcionalidad básica de TCP, portable y fácilmente modificable. JTCP esta totalmente implementado en Java, y es independiente del sistema operativo que lo soporta. El diseño de JTCP permite intercambiar fácilmente las técnicas de control de congestión utilizadas por el protocolo. Su portabilidad facilita la prueba de esas técnicas de control de congestión entre cualquier par de equipos conectados a la Internet. El objetivo de este desarrollo es posibilitar la evaluación de alternativas de control de congestión TCP en diferentes medioambientes de operación, tales como vínculos de banda ancha (fibra óptica) o con elevada tasa de errores (vínculos wíreless)..

Palabras claves: TCP, control de congestión TCP, performance TCP.

1. Introducción

El nivel de transporte en el modelo TCP/IP cumple un rol importante en dos aspectos: la adaptación del servicio provisto por IP a las aplicaciones, y el control de la tasa de datos que éstas introducen en la red. Este último aspecto, de poco peso en los comienzos de la Internet debido a su reducido tamaño, resulta en la actualidad de vital importancia para posibilitar el funcionamiento de la red

TCP (Transfer Control Protocol) [1] ofrece un servicio de transferencia de datos confiable y orientado a la conexión, lo cual lo hace apto para aplicaciones que requieren transferencias de datos de cierto volumen y sin errores. Considerado desde el punto de vista de la red, posee mecanismos de control de congestión, que impiden que las aplicaciones saturen a la red con información, y por lo tanto contribuye a evitar su congestión. Estos mecanismos, ausentes en la especificación original, se han ido agregando a TCP, debido a la creciente necesidad de controlar el tráfico en la red, motivada por el crecimiento de la misma.

UDP (User Datagram Protocol) [2], no ofrece conexiones ni garantías del envío de los datos, adaptándose de esta manera a aplicaciones que realizan transferencias de bajo volumen de datos, y soportan errores. UDP no presenta mecanismo alguno de control de congestión, lo que lo hace peligroso desde el punto de vista de la red, ya que puede saturarla con paquetes.

En la actualidad, y como consecuencia de las necesidades de comunicación de nuevos tipos de aplicaciones y condiciones de la red, han surgido otros protocolos que completan el servicio ofrecido por TCP y UDP. Entre ellos cabe mencionar DCCP (Datagram Congestion Control Protocol) [3]. por su estado de avance en lo que se refiere a standarización. Provee un servicio orientado a conexión pero no confiable, adaptándose a aplicaciones multimedia en tiempo real. Este protocolo, al igual que TCP, hace énfasis en los métodos de control de congestión, atendiendo a la necesidad de no saturar la red.

La motivación de este trabajo surge de la observación de la continua actividad de investigación y desarrollo que se ha llevado a cabo sobre los métodos de control de congestión utilizados por TCP, y la convicción de que estos métodos podrían ser mejorados a través de mayor experimentación. El objetivo principal de este trabajo es constituir una base para desarrollar y evaluar nuevas estrategias de control de congestión que deben ser puestas en juego como respuesta a la aparición de nuevas tecnologías de transmisión, que por sus características hacen inadecuados los métodos existentes, por ejemplo vínculos de comunicación de gran capacidad y vínculos propensos a errores, como las redes wireless.

El presente trabajo consistió en el desarrollo de JTCP, un protocolo con un subconjunto de la funcionalidad de TCP. Se utilizó lenguaje Java, y se dio fundamental importancia al soporte para la implementación y evaluación de diferentes métodos de control de congestión. Se hizo énfasis en permitir la incorporación de diferentes métodos de control de congestión en forma de módulos independientes, de manera de tener un banco de pruebas portable y versátil que permita desarrollar y evaluar rápidamente diferentes estrategias de control de congestión.

Al ser JTCP una versión simplificada de TCP debido a que no debe ocuparse de problemas inherentes a su implantación en sistema operativo alguno y a que maneja solo una conexión, resultará de utilidad para la experimentación con el protocolo, permitiendo la realización de pruebas en escenarios reales (pares de equipos conectados a través de Internet)

El resto del trabajo se organiza de la siguiente manera: en la sección 2 se describe brevemente la evolución de los métodos de control de congestión en TCP, la sección 3 enuncia el objetivo de este trabajo describiendo las características principales del mismo, las cuales permiten determinar los alcances y limitaciones del resultado obtenido. En la sección 4 se hace un breve resumen de los mecanismos standard de control de congestión utilizados por TCP, ya que de ellos se derivó la estructura en módulos para el control de congestión en JTCP. La sección 5 describe la estructura de JTCP, considerando el aspecto dinámico (threads que intervienen) y los módulos que lo componen. La sección 6 describe brevemente la interacción entre JTCP y las aplicaciones que lo utilizan. La

sección 7 contiene las conclusiones derivadas del trabajo realizado, los trabajos en curso que continúan el descripto, y otros usos del prototipo desarrollado. Las referencias bibliográficas de mayor relevancia se citan en la sección 8.

2. Evolución del control de congestión en TCP

TCP es un protocolo de nivel transporte que se originó junto con la arquitectura que lleva su nombre (TCP/IP). Una característica que debe destacarse de TCP es que ha sobrevivido hasta la actualidad sin modificaciones en su diseño, adaptándose a cambios significativos en su medioambiente de operación a través de sucesivas mejoras en sus mecanismos.

A continuación se mencionan algunos de dichos cambios y las versiones más relevantes de TCP.

TCP fue originalmente definido en [1], donde se tratan sus aspectos básicos del protocolo:

1-El formato de frame, que se ha mantenido hasta la actualidad sin modificaciones aunque se le han agregado nuevas opciones [4] ,2- La interfaz genérica con las aplicaciones, que define requerimientos básicos realizados por las aplicaciones (open, read, write, close, etc) que en la práctica se encuentra condicionada por el sistema operativo, y se implementan a través de la interfaz sockets, y 3- la operación del protocolo, definida como una maquina de 11 estados que se activa por eventos, los cuales pueden ser primitivas del nivel aplicación, segmentos recibidos del TCP remoto y vencimiento de timers. Los estados definidos contemplan la totalidad de la operación del protocolo, teniendo en cuenta en detalle los aspectos de establecimiento de conexión (three way handshaking) y terminación de conexión. La transferencia de información, que se lleva a cabo fundamentalmente en el estado ESTABLISHED[1], no se trata dentro de la FSM, sino que por su simplicidad – en aquella primera especificación[2]– solo se definen ciertas variables propias de las partes emisora y receptora y algunas indicaciones sobre la operación.

El aspecto en que más ha variado TCP está relacionado con mejoras en el control de congestión, motivadas por el crecimiento de las redes; más tarde aparece el desafío de operar eficientemente sobre diversas tecnologías, tales como satélites y wíreless [5] [6].. Todo esto ha dado lugar a los cambios que brevemente se mencionarán a continuación.

En la versión original, no se realizan consideraciones acerca del control de congestión protocolo/red, debido a que la homogeneidad de las redes y la baja cantidad de equipos conectados no requería de este tipo de control (se especifica sólo un control de congestión emisor/receptor, basado en ventanas deslizantes de longitud variable). Estas caracteristicas las podemos encontrar en el TCP Berkeley (1983) que puede considerarse como la implementación del RFC 793.

A partir de 1988, en TCP Tahoe y posteriores, aparece un control de congestión orientado a no saturar la red, compuesto de dos mecanismos independientes pero que trabajan en conjunto en las implementaciones: Slow Start y Congestion Avoidance. Con estas modificaciones, el emisor, que antes podía inyectar paquetes hasta el límite de la ventana del receptor, pudiendo saturar la red, está ahora restringido por un limite adicional, que se obtiene de medir la respuesta de la red (considerando llegada de asentimientos) a la carga inducida. Ambos mecanismos se aplican al iniciar la transmisión, y luego cada vez que el emisor detecta síntomas de congestión. Mientras que Slow Start aumenta la tasa exponencialmente, Congestion Avoidance lo hace linealmente [7].

Posteriormente se incorpora Fast retransmit, mejora que permite a un emisor retransmitir un paquete que detecta como perdido de manera inmediata, sin esperar el tiempo de vencimiento del timer de retransmisión, Está basado en la característica de TCP de emitir un ACK cada vez que llega un segmento. Por lo tanto, si un segmento se ha perdido, se comenzará a recibir ACKs duplicados. Se tiene en cuenta la posibilidad de recepción de ACKs duplicados como consecuencia de un reordenamiento temporal de paquetes en la red.

---

[1] Es posible transferir datos en los segmentos que se intercambian para el establecimiento de conexión, así como también, debido al esquema de desconexión asimétrica soportada, recibir datos luego de haber solicitado un CLOSE.

[2] En ese momento no se tuvo en cuenta el control de congestión.

En 1990 aparece TCP Reno, que presenta las caracteristicas del Tahoe, más la incorporación del algoritmo Fast recovery, mejora que permite que el TCP emisor, ante la pérdida de un paquete reaccione con el mecanismo de Congestion Avoidance y no el de Slow Start, con lo cual no se baja drásticamente la tasa de transmisión. TCP asume que cuando la pérdida del paquete se detecta por la recepción de ACKs duplicados, no hay congestión en la red, ya que se reciben ACKs, y que la pérdida ha afectado sólo al paquete en cuestión [8].
Posteriormente aparecen modificaciones tendientes a tratar nuevos problemas, surgidos del aumento de la capacidad de la líneas de transmisión a las cuales se las denomina "long fat networks" (LPN) aludiendo a su gran capacidad de transmisión y a su considerable demora de propagación (en relación a su capacidad). Estas modificaciones, demasiado extensas para describir aquí, consisten entre otras cosas en la posibilidad de incrementar la longitud de los segmentos TCP, para explotar adecuadamente ese tipo de líneas, en el uso de ACKs selectivos (SACKs) para que el emisor retransmita sólo los paquetes no recibidos, en la medición más refinada de tiempos utilizada para estimar tiempos de retransmisión (timestamp option), etc. Para detalles referirse a [9] [10] [11].
TCP Vegas [12] es una mejora que se caracteriza por incrementar el throughput y disminuir las pérdidas que sufre Reno a través de algoritmos mejorados. Sólo involucra cambios de implementación en el lado emisor, resultando compatible con cualquier otra versión válida de TCP.
Actualmente TCP continúa en evolución, no sólo para mejorar su performance, sino para enfrentar nuevos desafíos surgidos de su operación sobre nuevas tecnologías de transmisión.
En resumen, TCP se ha adaptado a importantes cambios en su medioambiente de operación: desde sus comienzos, operando sobre redes homogéneas con pocos equipos conectados, donde no era necesario el control de congestión entre el nivel de red y el protocolo, hasta la actualidad, operando en redes heterogéneas con gran cantidad de equipos en las cuales cobran importancia fundamental dichos mecanismos.
Como respuesta a esos cambios, TCP ha incorporando diferentes algoritmos de control de congestión, que han ido mejorando su performance y adaptándolo a los nuevos requerimientos. Además, la operación sobre tecnologías diversas, que implica la combinación de vínculos altamente confiables y de gran capacidad de transmisión (fibra óptica) con otros de menor capacidad y considerable tasa de errores (vínculos wireless), ha producido importante actividad de investigación tendiente a adaptar el protocolo a esas características.

3. Objetivo, alcances y limitaciones

Se describe a continuación el objetivo principal del trabajo presentado, sus posibles usos alternativos, y las características y limitaciones del mismo.
Como fue comentado previamente, TCP es un protocolo que ha sido definido originalmente en el RFC 793. En este documento, se hizo énfasis en aspectos generales del protocolo, tales como su estructura general, representada por una máquina de estados, los mecanismos de conexión y desconexión, y el formato de segmento. Se dejó sin definir totalmente, sujeto a posterior investigación, el aspecto de transferencia de datos, el cual ocurre fundamentalmente en el estado de conexión establecida (ESTABLISHED) de la FSM, y tiene como principal componente el aspecto de control de congestión, En los trabajos posteriores ya mencionados, se incorporaron diversas estrategias relativas a dicho aspecto, resultando en la actualidad un tema objeto de significativa investigación.
El objetivo principal de este trabajo es el desarrollo de una versión de TCP (a la que denominamos JTCP), tal como se definió en el RFC 793 [1], a la cual pueda incorporársele, de manera simple y modular, la funcionalidad (relativa a control de congestión) de las diferentes versiones existentes de TCP (Tahoe, Reno, etc),
De esta manera será posible evaluar el comportamiento de diferentes alternativas para el control de congestión en condiciones reales, es decir, comunicando las dos partes del protocolo a través de la

Internet. De igual manera, es posible incorporar modificaciones a estos mecanismos ya existentes y desarrollar nuevas heurísticas, las cuales podrán ser codificadas y agregadas a JTCP sin mayor esfuerzo, para finalmente ser evaluadas en su ambiente real de operación.
Un aspecto de este desarrollo al cual se le dio fundamental importancia, es la portabilidad. Esto implica obtener un código que pueda correr en cualquier plataforma, independientemente del sistema operativo. Esto constituye una diferencia significativa con las implementaciones de TCP, que por razones de eficiencia, residen en el sistema operativo, resultando así no portables y peligrosas de modificar por las consecuencias que podrían tener los errores. Para lograr la portabilidad, sacrificando eficiencia, el desarrollo fue realizado en Java. Las consecuencias de esta elección se consideran más adelante.
Se describen a continuación las características principales del desarrollo, resaltando sus alcances y limitaciones.
1-Programación simple, modular y código fácil de modificar a quien desee incorporar nueva funcionalidad en el protocolo: es una característica que limita en cuanto a performance, ya que el uso de estructuras de datos y procesos que operen de manera eficiente, muchas veces disminuye la legibilidad del código y dificulta la incorporación de agregados y modificaciones. La decisión tomada, de acuerdo con el objetivo planteado, fue sacrificar la eficiencia en función de la claridad y modularidad. JTCP está dividido en módulos, resultando clara la división de funciones. Además, se implementó un módulo administrador de eventos relativos al control de congestión, que permite el intercambio de estos módulos conociendo sólo los eventos que producen, los eventos ante los cuales deben reaccionar, y ciertas variables genéricas relacionadas con el manejo de los datos a transferir.
2-Manejo de una única conexión por parte de JTCP e imposibilidad de reutilizar la instancia JTCP en caso de abortar o cerrar la conexión. La consecuencia de lo anterior es que una aplicación debe crear una instancia de JTCP por cada conexión que desee realizar, en contraste con TCP, que es capaz de manejar múltiples conexiones concurrentemente. Por otra parte, para simplificar el proceso, se resolvió no reinicializar la instancia JTCP, lo que implica que una aplicación que cierre una conexión, si desea abrir otra, deberá crear una nueva instancia JTCP. Se adoptó esta decisión debido a que para los objetivos planteados no es importante dicha característica, que contribuiría a complicar el código y a restar eficiencia.
3-Portabilidad. Esta característica es importante en esta aplicación, ya que por su naturaleza, exige ser probada entre un par calquiera de equipos conectados a la Internet.
Dos aspectos fundamentales para lograr portabilidad son: la implementación del protocolo 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 aparejado un problema relacionado con el acceso al nivel de red, que constituye el soporte de comunicación para TCP.
4-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 reside en el nivel de red. Se decidió utilizar UDP, debido a que al estar JTCP 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 JTCP con implementaciones standard de TCP, ya que en nuestro caso el segmento TCP se encapsula en uno UDP y este en uno 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 JTCP con versiones reales3. Sin embargo, previendo la necesidad de esta prueba, se utilizará algún soporte de amplia difusión para acceso al nivel de red, tal como libpcap [13] desde un ambiente Linux, con una librería de acceso desde Java (Jpcap [14]), a efectos de poder interactuar directamente con IP y confrontar este desarrollo con implementaciones de TCP embebidas en los sistemas operativos..
5-Preservación del formato de PDU de TCP. No se alteró el formato de segmento TCP Se contempló también el proceso de opciones. Esta es una característica fundamental para cuando se realice comunicación con implementaciones standard de TCP.
6-Compatibilidad con las invocaciones en Java: JTCP provee un esquema de llamadas a funciones compatible con el provisto por Java. Las invocaciones a las funciones de JTCP son bloqueantes, y se utiliza el mecanismo de excepciones para indicar condiciones anormales. El objetivo de mantener esta compatibilidad es que cuando se implemente la totalidad de las funciones provistas para sockets en Java, las aplicaciones que utilizan TCP puedan adaptarse a JTCP con mínimos cambios.

4. Aspectos de importancia en el mecanismo de control de congestión de TCP

El objetivo de los mecanismos de control de congestión de TCP es ofrecer a la aplicación el mayor ancho de banda posible, sin interferir con otros protocolos que hacen uso de la red, ni saturar a ésta con exceso de información.
En la presente sección se mencionan los diferentes mecanismos que se ponen en juego para llevar a cabo el control de congestión. Estos mecanismos sirvieron como base para definir la modularización utilizada en el desarrollo de JTCP. Algunos de ellos dieron lugar a módulos separados, mientras que otros, por ser interdependientes, se agruparon en un módulo, y por último, aquellos más simples quedaron embebidos en módulos que no integran el conjunto de los considerados “intercambiables” en esta implementación. Estos mecanismos se describen en general [15]., [16], [18] Para una descripción detallada de dichos mecanismos, que no se hace aquí por razones de espacio, referirse as la bibliografía específica.
Podemos distinguir los siguientes mecanismos
Mecanismos que permitan alcanzar en el menor tiempo posible y sin riesgo de saturar a la red, la tasa máxima posible de envío (slow start y congestion avoidance)
Un método para calcular la máxima tasa de envío, sin saturar al receptor (ventana deslizande de tamaño variable).
Reacción rápida ante pérdidas de bloques, antes de recurrir a lo mecanismos de retransmisión (fast retransmit)
Método para regular el tráfico después de la detección de una pérdida, hasta la normalización de la situación (fast recovery).
Una manera de evitar sobrecargar la red con confirmaciones (ACKs) para los bytes bien recibidos, pero que sin embargo no incurra en demoras tales que originen retransmisiones innecesarias (algoritmo de retraso de confirmaciones).
Una manrera de detectar la información enviada que no ha sido recibida por el receptor, y retransmitirla (mecanismos de retransmisión).
Un método para estimar el tiempo de ida y vuelta de los segmentos entre los dos hosts involucrados en la conexión TCP.
Un método que permita estimar, en base al tiempo de ida y vuelta, el tiempo máximo de espera para una retransmisión.

[3] Por “version real” entendemos un TCP implementado en un lenguaje tal como C++ y formando parte del kernel del sistema operativo.

Protección contra el envío de caracteres aislados o la actualización de ventana en pequeñas cantidades de caracteres, lo cual produciría el envío de segmentos con muy poca información (algoritmos silly window syndrome, y de Nagle).

5 Estructura de JTCP

5.1 Dinámica

A efectos de proporcionar una idea general del funcionamiento de JTCP y de su interacción con la aplicación, se describe a continuación la relación entre los diferentes componentes activos que conforman el protocolo, representados por threads de ejecución Java, Los threads propios de JTCP, interactúan entre sí y con uno o más pertenecientes a la aplicación, dependiendo su número de cómo ésta haya sido diseñada. Esos threads y sus interacciones se muestran, en la figura 1.

JTCP, como cualquier aplicación standard de comunicaciones implementada en Java, comprende dos threads relacionados con la comunicación, en este caso con el JTCP remoto: 1- El thread de emisión, que se encarga de enviar segmentos TCP que encuentra en su cola de segmentos y que son generados por el thread de control o por los timers4, y 2- el thread de recepción, que da forma de eventos (ARRIVE) para la FSM a los segmentos TCP recibidos del lado remoto. Estos dos threads encargados de la comunicación interactúan con el soporte de comunicaciones, que puede ser UDP o IP como fue mencionado.

El trhead de la FSM representa la máquina de estados TCP, y a través de él se procesan los eventos, que son extraídos de la cola de eventos. En base a este proceso se cambia el estado del protocolo, incluyendo la conexión, desconexión y transmisión de datos. En este último estado (ESTABLISHED) es donde se ponen en juego los mecanismos de control de congestión y retransmisión, que son objetos de nuestro desarrollo.

Los threads de la aplicación (generalmente un thread de emisión y otro de recepción, aunque pueden soportarse varios), ejecutan código JTCP a través de la invocación a funciones definidas en la interfaz JTCP-Aplicación, y son bloqueados en este código hasta que se produce un resultado al requerimiento solicitado (open, read, etc.).

Otros elementos activos son los timers, ejecutados por un thread background provisto por Java. Las acciones de los timers son diversas, como producir eventos destinados a la FSM o segmentos a enviar.

El acceso a algunas de las variables de JTCP y a los datos, que se encuentran en las ventanas de emisión y recepción, debe ser sincronizado entre los threads descriptos para no producir errores. Esta sincronización se realizó utilizando las facilidades provistas por Java.

La comunicación entre los threads originados en JTCP se lleva a cabo a través de las colas de eventos de la FSM, la cola de segmentos a enviar del thread de emisión, y la comunicación entre los threads de la aplicación y el de JTCP (la máquina de estados) a través de variables destinadas a ese efecto en cada evento producido como consecuencia de una invocación de la aplicación a JTCP.

5.2 Módulos que componen la implementación

A continuación se describen brevemente los diferentes módulos que componen JTCP. En la tabla 1 se indica para cada módulo, su nombre, función, interfaz y tipo.

Se ha realizado un diseño modular que permite reemplazar fácilmente cualquiera de los módulos para cambiar su funcionalidad. Debido a que el objetivo del trabajo es la evaluación de técnicas para el control de congestión, se ha previsto que los módulos relacionados con esta función serán cambiados con bastante frecuencia, y no sólo en cuanto a su implementación, sino en lo que

[4] Los timers son implementados utilizando funcionalidad Java, activándose a través de la ejecución de un timer en background (aclarar esto mejor)

respecta a su funcionalidad. Para que estos cambios resulten simples de poner en práctica, se ha definido un módulo adicional, el administrador de eventos de control de congestión, que posibilita el reemplazo de los demás módulos sin necesidad de conocer sus interacciones.

Modulo	**Función**	**Interfaz**	**Tipo**
Instancia TCP	Variables comunes de TCP Coordinación de los módulos Interacción con la aplicación	TCPAppInterface	IA MV
Máquina de estados TCP	Control de los eventos que recibe TCP	TCPFSM	CT
Envío de segmentos	Supervisión de puerto UDP de comunicación Envío de segmentos TCP de control Construcción y envío de segmentos TCP con datos	EnvioSegmentos	CO
Recepción de segmentos	Supervisión de puerto UDP de comunicación Recepción de datagrams UDP Conversión a segmentos TCP	RecepcionSegmentos	CO
Ajuste de segmentos recibidos	Trunca números de secuencia de los segmentos recibidos para adaptarlos a la ventana de recepción	AjusteSegmento	TS
Generación de segmentos	Genera segmento de datos o de control en base a parámetros recibidos y al estado de los módulos de datos de TCP	GeneradorSegmentos	TS
Almacenamiento de datos a ser enviados	Ventana de emisión, datos y variables asociadas. Datos urgentes PUSH	BufferEmision	MD
Almacenamiento de datos recibidos	Ventana de recepciçon, datos y variables asociadas. Datos urgentes PUSH	BufferRecepcion	MD
Determinación de tasa máxima de envío según condiciones de la red	Ajuste de ventana de congestión según estado de la red, mantiene variables y aplica algoritmos slow start, congesion avoidance, fast retransmit y fast recovery	AjusteCWND	CC
Determinación de tasa máxima de envío según condiciones del receptor	Ajuste de ventana según estado del receptor	BufferRecepción	CC
Decisión de envío de ack	Determina cuando se deben enviar las confirmaciones de segmentos	EstrategiaEnvioACKs	CC
Elección de los segmentos recibidos utilizados para estimar tiempo de ida y vuelta	Discrimina segmento utilizables y no utilizables Mantiene una estimación del RTT	SeleccionSegmentosEstimacionRTT	CC
Estimación del tiempo para el timer de retransmisión (RTO)	En base a los tiempos de ida y vuelta mantiene actualizado el RTO	EstimacionRTO	CC
Retransmisión de segmentos	Mantiene variables que determinan cuando debe retransmitirse	retransmision	CC
Coordinador de los módulos de control de congestión		AdmMod	CM

Tabla 1. Módulos principales componentes de JTCP

Los módulos se agrupan en los siguientes tipos:
Control de congestión (CC): Procedimientos para detectar y reaccionar ante las condiciones de tráfico impuestas por la red y por el lado remoto. Tratan de optimizar el uso de la red, en base al

momento, longitud de los segmentos y confirmaciones enviados. Incluye aspectos relativos a la recuperación de datos.
Control de módulos (CM): Si bien los diferentes módulos que componen JTCP interactúan entre sí, esta función se refiere específicamente a la interacción de los módulos que activamente intervienen en el control de congestión y aquellos relacionados. El motivo de ofrecer este tipo de control es facilitar el agregado y reemplazo de nuevos módulos de control de congestión sin conocer en detalle aspectos de la implementación.
Control de JTCP (CT): Controla el correcto funcionamiento de JTCP en lo relativo a los estados descriptos en el RFC 793. Se ocupa además del manejo de los timers relacionados con estas funciones (tiempo de espera de SYN, Keepalive, etc) y de respuestas producidas por JTCP ante situaciones especiales, por ejemplo, recepción de un segmento conteniendo RST.
Manejo de variables comunes (MV): Mantenimiento de variables genéricas de JTCP que deben ser conocidas por los diferentes componentes, por ejemplo opciones negociadas. Incluye también las variables que referencian a los diferentes componentes de la implementación, que deben ser conocidos por los demás.
Interfaz con la aplicación (IA): Se encarga de ofrecer a las aplicaciones que utilizan JTCP una interfaz similar a la ofrecida por Java, con igual funcionalidad, modo bloqueante y excepciones. Es aquí donde se sincronizan los threads generados por la aplicación con los del JTCP.
Manejo de datos (MD): Se ocupa del almacenamiento de los datos de la aplicación a comunicar entre ambos lados. Implementa los buffers necesarios y las variables asociadas con las ventanas de emisión y recepción. Algunas de estas variables definidas en el RFC 793 (por ejemplo SND_NXT, SND_UNA) deben ser conocidas por los demás módulos para posibilitar su operación. Tiene a su cargo el manejo de PUSH y datos urgentes, y el registro de la señal de FIN.
Comunicación (CO): Su función es simple. Se encargan de establecer un vínculo de comunicación entre ambos lados del protocolo y enviar por él los segmentos TCP. Estos módulos, que operan sobre UDP, pueden ser reemplazados para operar sobre IP, en la medida que el sistema operativo permita dicho acceso y en que se utilicen librerías no standard de Java para acceder a IP.
Tratamiento de segmentos (TS): Tienen por función la generación de segmentos TCP, de acuerdo al estado del protocolo, y el ajuste de segmentos entrantes a la ventana de recepción.

5.3 FSM y Eventos

La FSM está implementada como un thread que procesa eventos almacenados en una cola asociada.. Estos eventos provienen de requerimientos de la aplicación (generados por el o los threads de la aplicación), de la llegada de segmentos TCP (generados por el thread de recepción), y del vencimiento de los timers de control definidos en la norma (generados por el thread backgrounbd provisto por Java, encargado de la administración de timers).
Se agregaron nuevos eventos de entrada a la FSM, producidos por ella misma, con el objetivo de facilitar y mejorar la performance de la implementación. Representan situaciones producidas que deben procesarse en orden, es decir, respetando los posibles eventos en cola en el momento que ellas se producen. Estos eventos, que no alteran el comportamiento descripto en la norma, son CANCEL_INCOMING , que es producido cuando desde la aplicación se cancela un Listen previo, con lo cual se debe volver al estado inicial, y RESET que indica una condición excepcional que hara que la FSM prepare la terminación del JTCP, enviando la causa del reset a todos los eventos pendientes.
Un evento entregado a la FSM no es eliminado por ésta despúés de procesarlo, sino que es accedido nuevamente por el thread que lo produjo para obtener los resultados correspondientes.
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.
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 JTCP: por el momento: open (activo y pasivo) close, send y receive.
2-El generado por el thread de recepción, Arrive, contiene el segmento TCP recibidio del lado remoto
3-Los eventos internos de la implementación, reset, y camcel incoming
4-El evento generado por el timer que controla el tiempo de espera para la reutilización del port involucrado (MSL)..
5-El evento generado por el timer para espera por establecimiento de conexión: un RESET a la FSM (evento agregado en la implementación), en caso de que se exceda el tiempo máximo de espera.

5.4 Interacción entre los módulos de control de congestión

Utilizando como referencia los mecanismos de control de congestión desarrollados hasta el momento para TCP, descriptos brevemente en la sección 4, se dividió la función de control de congestión en diferentes módulos, los cuales interactúan entre sí y con oros módulos de JTCP. Estos módulos fueron descriptos previamente en la sección 5.2..
En función de lograr un medioambiente que permita un fácil intercambio de las diferentes estrategias de control de congestión, representadas por esos módulos, se trató de simplificar la interacción entre ellos.
Las interacciones consideradas son de dos tipos.
1- Consulta, que consiste en que un módulo consulte a otro el valor de una variable, por ejemplo, el número de secuencia del próximo carácter a enviar. Estas consultas son interacciones entre dos módulos, y para que resulten claras sólo se requiere que los módulos tengan una interfaz bien definida través de la cual pueda accederse a dichas variables. Las variables más importantes utilizadas se muestran a manera de ejemplo en la Tabla 2.
2 – Reacción de un módulo ante ciertos "eventos"5 de interés para la función que llevan a cabo. Un evento producido, por ejemplo la recepción de un ACK puede dar lugar a cambios en más de un módulo. Para lograr una interacción simple entre los módulos, se definieron los eventos relevantes, y un módulo adicional encargado de su manejo, los cuales se describen a continuación.

Nombre	Módulo	Función
SND_WND	Buffer Emisión	tamaño anunciado por el receptor para su ventana
SND_NXT	Buffer Emisión	Número de secuencia del próximo carácter a enviar
SND_UNA	Buffer Emisión	Número de secuencia del primer carácter enviado sin confirmación
RCV_NXT	Buffer Recepción	Número de secuencia del próximo carácter a recibir
RCV_WND	Buffer Recepción	Tamaño anunciado para la ventana de recepcion
CWND	Buffer Emision	Número máximo de caracteres a enviar desde el último confirmado
IRS	Buffer Recepción	Número de secuencia inicial para los caracteres recibidos
ISS	Buffer Emisión	Número de secuencia inicial para los caracteres enviados

Tabla 2. Algunas de las variables accedidas por los módulos relativos al control de congestión

5.4.1 Administrador de eventos

El módulo administrador de eventos tiene por función simplificar las interacciones entre los diferentes módulos de control de congestión. La operación se describe a continuación.

[5] Estos "eventos" no son los eventos definidos previamente para TCP, relativos a la FSM definida en el RFC 793, sino que son eventos propios del control de congestión, definidos en base a su funcionalidad,

Cada módulo implementa la interfaz eventosCC, que define las invocaciones genéricas a los módulos al producirse cada evento. Cada evento tiene asociado un método que será invocado en los módulos que lo han solicitado, al producirse dicho evento Por ejemplo, el evento "Envío de un ACK", se asocia al método ACKEnviado, método que deberá estar implementado en cada módulo que desee procesarlo.
La operación puede resumirse en los siguientes pasos:
1-Un módulo de control de congestión, al inicializarse, debe registrarse con el administrador de eventos para informarle acerca de qué eventos desea ser informado.
2-Cada módulo, cuando produce un evento, lo informará al administrador de eventos, enviando los parámetros definidos. La invocación es definida en la interfaz eventosCC. .
3-El administrador de eventos, al recibir información sobre un evento, invoca a los módulos que se han registrado para recibirlo.

Los eventos definidos se enumeran a continuación. Corresponden con el control de congestión standard de TCP, al igual que los módulos afectados por cada uno de ellos. Los módulos involucrados y los parámetros se muestran en la tabla 3. Allí se indica el evento, la función asociada a la invocación, tanto en el administrador de eventos como en los demás módulos, el módulo que origina el evento, y los módulos que son informados6.

Evento	**Funcion**	**Parámetro**	**Generado por**	**Módulos receptores**
(RAV) Recepción de un ACK válido	AckVRecibido	TCPSegment (Segmento TCP)	TCPFSM (Máquina de estados TCP)	AjusteCWND SeleccionSegmentosEstimacionRTT retransmision BufferEmision
(RAA) Recepción de un ACK aceptable para estimación RTT	AckARecibido	double (tiempo estimado RTT)	TCPFSM (Máquina de estados TCP)	EstimacionRTO
(RND) Recepción de nuevos datos	DatosRecibidos	TCPSegment (Segmento TCP)	BufferRecepcion (Buffer de recepción)	EstrategiaEnvioACKs
(ED) Envío de un segmento con datos	envioSegmento	TCPSegment (Segmento TCP)	EnvioSegmentos (Emisor de segmentos)	SeleccionSegmentosEstimacionRTT retransmision
(EA) Envío de ACK	ACKEnviado	Sin parametros	EnvioSegmentos (Emisor de segmentos)	EstrategiaEnvioACKs
(VTR) Vencimiento timer retransmisión	toutTransm	Sin parámetros	Retransmisión	EstimacionRTO

Tabla 2. Eventos definidos para los módulos relativos al control de congestión

(RAV) Recepción de un ACK válido, es decir, que confirma nuevos datos ya enviados
(RAA) Recepción de un ACK aceptable para ser tenido en cuenta en la estimación del tiempo de ida y vuelta al host remoto.
(RND) Recepción de nuevos datos
(ED) Envío de un segmento con datos, ya sea la primera vez que se envían o una retransmisión.

[6] Esta tabla refleja el caso particular de la funcionalidad Standard de TCP implementada.

(EA) Envío de un ACK
(VTR) Vencimiento del timer de retransmisión de segmentos

6 Interacción entre la aplicación y JTCP

Desde el punto de vista del desarrollo de aplicaciones, el acceso a JTCP se ha mantenido similar al standard provisto por Java, es decir, llamadas bloqueantes y anuncio de situaciones anormales a través del mecanismo de excepciones, en este caso, utilizando el tipo TCPException. Las invocaciones se especifican en TCPAPPInterface, siendo implementadas hasta el momento, las básicas que permiten llevar a cabo una comunicación entre las aplicaciones: open, close, read y write.

Para los casos en que se requiera modificar JTCP, por ejemplo para el intercambio o agregado de módulos de control de congestión, si bien no es necesario, resulta conveniente contar con una descripción más detallada de esta interacción.

En ella están involucrados dos threads: uno de ellos es el iniciado por la aplicación[7] y el otro el que corresponde a la máquina de estados JTCP.

El thread de la aplicación ejecuta código de la aplicación hasta que produce una invocación a TCP (a través de la invocación de un método definido en la interfaz TCPAPPInterface, por ejemplo read). A partir de este momento, este código producirá un evento en la FSM JTCP, 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 en la aplicación quedará esperando la resolución de lo solicitado a JTCP. Dicha espera se produce a través de un testeo continuo 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[8]. Dicha excepción deberá ser tratada por código provisto por la aplicación.

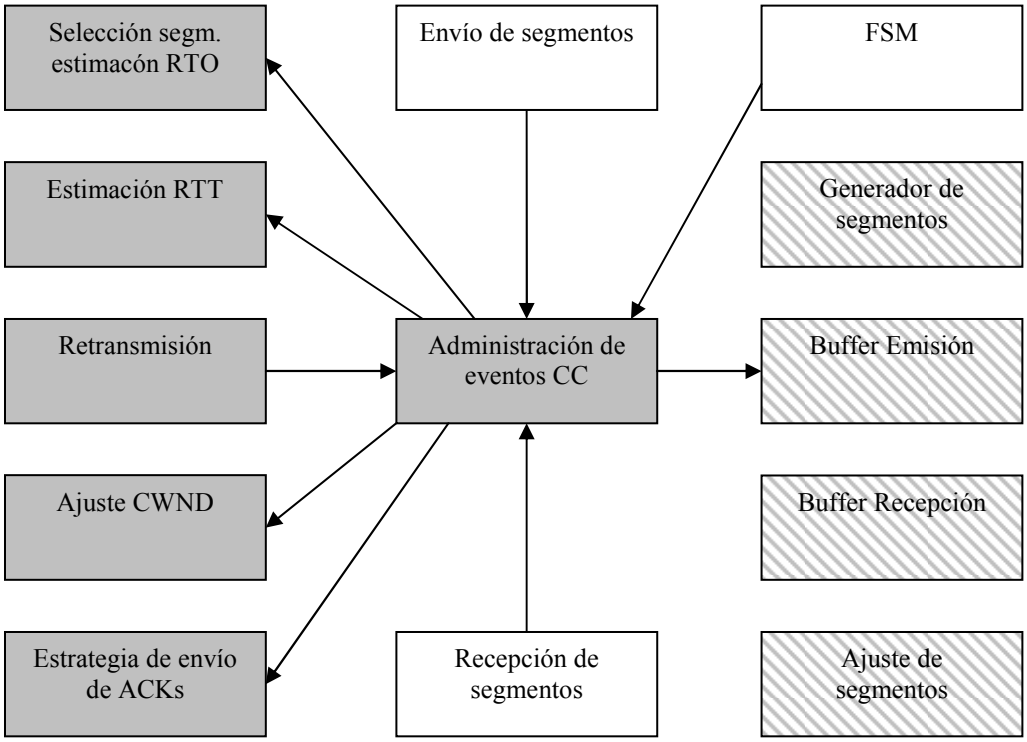
Debemos distinguir los siguientes casos

En el más simple, la llamada se resuelve completamente durante el proceso del evento correspondiente por parte de la FSM. Por ejemplo, en el caso de Status – no implementado aún – que consiste en copiar las variables relevantes de JTCP para entregarlas a la aplicación-.

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 la cantidad de caracteres requerida, que se devuelve a la aplicación, con lo cual finaliza la invocación con el fin del proceso del evento. Sin embargo, podría ocurrir que la cantidad de bytes solicitada no estuviera disponible, y como el llamado es bloqueante, se debería esperar que se tenga dicha cantidad de datos en el buffer o que venza el timer de tiempo máximo de espera de la aplicación por el JTCP. 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 deba procesar. Los posibles casos aquí serian la recepción de los datos, retorno normal, o un error de conexión o vencimiento de timer, lo que produce que el código JTCP invocado por el thread de la aplicación genere una TCPException.

[7] Puede ser uno o más threads. El caso más común es el de una aplicación que cree un thread de lectura y otro de escritura sobre TCP, pero se admite que la cantidad de threads sea variable.

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



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 otro TCP. 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 misma FSM lo anunciará al código de invocación de JTCP, 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 open9.

7. Estado actual, conclusiones y continuación del trabajo

JTCP constituye el primer paso en un proyecto cuyo objetivo final es la investigación relacionada con el control de congestión en TCP. Los objetivos finales son la evaluación de los mecanismos de control de congestión existentes en TCP, y la implementación de nuevas heurísticas que permitan mejorar y adaptar TCP a nuevos medios de comunicación y/o nuevos requerimientos de las aplicaciones.

El objetivo fijado para esta primera etapa, el desarrollo de un TCP portable con un diseño que permite introducir fácilmente nuevos mecanismos, ha sido alcanzado satisfactoriamente con el desarrollo de JTCP. Su efectividad se comprobó a través de la implementación de los métodos standard de control de congestión usados por TCP.

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

El lenguaje utilizado, Java, seleccionado en principio por razones de portabilidad, demostró ser efectivo para este proyecto, tanto por razones de performance como por características tales como manejo de threads de ejecución y soporte para timers.
Es de destacar que debido a la información de carácter general contenida en el RFC 793, fue necesario recurrir a implementaciones "reales" en ciertas ocasiones[15]. Sin embargo, estas ultimas sólo fueron de utilidad para comprender los mecanismos de TCP, ya que el código, contrariamente al de esta implementación, presenta una complejidad considerable debido a la necesidad de lograr un proceso eficiente y a su interacción con el sistema operativo.

Las características salientes de JTCP son las siguientes:
JTCP es portable, independiente del sistema operativo, lo que permite utilizarlo entre cualquier par de hosts conectados a la Internet.
Gracias a su diseño modular y simplicidad en el código, JTCP permite implementar y evaluar rápidamente nuevas heurísticas de control de congestión
JTCP ofrece una funcionalidad limitada respecto de TCP, siendo igualmente compatible con este último.
JTCP maneja sólo una conexión; esto posibilita una buena performance, pese a estar implementado en Java y en espacio usuario. Fueron realizadas pruebas con aplicaciones con varias conexiones (varias instanciaciones de JTCP), no notándose una degradación de la performace que interfiriera con el correcto funcionamiento del protocolo.
JTCP corre sobre UDP cuando en realidad, de acuerdo a su funcionalidad, debería correr sobre IP. Se tomo esta decisión para posibilitar su portabilidad. La encapsulación UDP no afecta los resultados de las evaluaciones, ya que solo agrega los bytes de encabezamiento UDP.

Los objetivos planteados para la próxima etapa son los siguientes:
Mejoramiento de la interfaz ofrecida a las aplicaciones, para aumentar su compatibilidad con la ofrecida por Java, de manera de poder utilizar aplicaciones ya existentes para evaluar las técnicas en experimentación.
Experimentación y evaluación de nuevas técnicas de control de congestión. Se espera que esta actividad tenga como consecuencia secundaria, posibles mejoras en lo que se refiere a la modularización del control de congestión en JTCP.
Operación de JTCP directamente sobre IP, en sistema operativo Linux y utilizando librerias no standards de Java. Esto permitirá confrontar JTCP con implementaciones standard de TCP.

8. Bibliografía

[1] J. Postel, "Transmission Control Protocol, DARPA Internet Program Protocol Specification", RFC 783, 1981
[2] J. Postel, "User Datagram Protocol", RFC 768, 1980
[3] E. Kohler; M. Handley and S. Floyd, "Datagram Congestion Control Protocol", RFC 4330, March 2006.
[4] K. Pentikousis and H..Hadr, "Quantifying the Deployment of TCP Options", IEEE Communication Letters, April 2004.
[5] N.. Balakrishnan et al, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", Computer Science Division, Department of EECS, University of California at Berkeley
[6] M. Allman, et al "Ongoing TCP Research Related to Satellites", RFC 2760, February 2000
[7] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms", RFC 2001, 1988

[8] S. Floyd and T Henderson., “The NewReno modifications to TCP`s Fast Recovery Algorithm”, RFC 2582, April 1999.
[9] V. Jacobson and R. Braden, “TCP Extensions for Long-Delay Paths”, RFC 1072, October 1988
[10] V. Jacobson; R. Braden and L. Zhang, “TCP Extension for High-Speed Paths”, RFC 1185, October 1990
[11] V. Jacobson; R. Braden and D. Borman, “TCP Extensions for High Performance”, RFC 1323, May 1992.
[12] L. Brakmo; S. O`Malley,. and L. Peterson,., “TCP Vegas: New Techniques for Congestion Detection and Avoidance”, .Proc. of ACM SIGCOMM, October, 1994.
[13] libpcap Official Site, http://www.tcpdump.org/
[14] Jpcap Official Site, http://sourceforge.net/projects/jpcap/
[15] W. Stevens, “ TCP/IP Illustrated. Volume II: the Implementation”, Addison Wesley, 1994.
[16] W. Stevens , “ TCP/IP Illustrated. Volume I: the Protocols”, Addison Wesley, 1994.
[17] V. Paxson and M. Allman, “Computing TCP's Retransmission Timer”, RFC 2988, November 2000.
[18] M. Allman; V. Paxson, and W. Stevens, “TCP Congestion Control”, RFC 2581, April 1999
[19] P Karn. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", SIGCOMM 87, 1987.